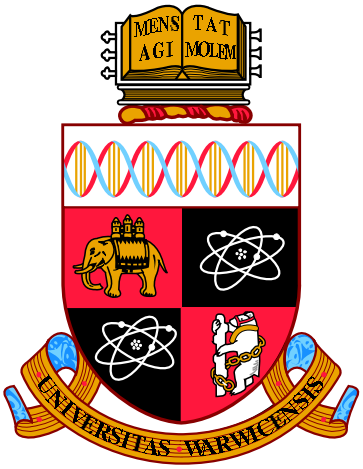


Object Detection for Collision Avoidance from Lidar Point Clouds

by

Martin Rebane

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

School of Engineering

July 2022

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor, Dr. Tardi Tjahjadi for an excellent cooperation and support during my PhD research. You are the best supervisor, no doubt! I would like to thank the reviewers, Dr. Yunfei Chen and Dr. Ke Chen for a valuable discussion, their encouragement and comments. Working together with people from Dynium was the best thing that happened during my PhD. I would like to thank Dr. Christopher Marshall, Charles Kirby, Jaime Conde, Sinan Mutlu, Patrick Harding, Bruno Santos and all the other people from Dynium lab for invaluable discussion, help and fun during my research there! I will miss those research discussions with Chris in different coffee shops! I am very grateful for Innar Liiv from TalTech who helped to reflect many thoughts I had. I am very grateful for Dr. Henn Käärik, Prof. Arvo Krikmann, Dr. Liisi Laineste, Prof. Rein Kuusik and Prof. Leo Võhandu for showing me how much fun it is to do research during my early studies.

My sincere thanks goes to Liisa Jõgiste and Tom who helped me to build the lidar rack for the car. Astrid Pärn was there at the crucial moment where I needed to find motivation to start writing the thesis. Thank you, Indrek Rebane for helping me collect data in the snow storm.

Thank you Karl Amandus, Emma Eliise, Jaan Konrad, Marit and Marita for your support! Thank you, Gunnar Piho for forwarding me the email about the PhD offer from Tardi. Thank you, Marita Lumi for charging my batteries! I am grateful for Google for providing GPU-s for research.

Finally, my sincere thanks goes to the producers of Yorkshire tea for making this research possible at all.

Declarations

The author hereby declares, that the work presented in this thesis is entirely his own unless explicitly acknowledged, including citations of published and unpublished sources.

The author also happily confirms that the thesis has not been submitted for a degree at any another university.

Abstract

This thesis advances deep learning models that are essential for collision avoidance of autonomous vehicles. The first contribution is an advanced multi-component loss function for 3D object detection algorithms where location of the object and dimensions of its bounding box are estimated simultaneously. The loss function penalises model's training process when the prediction does not match an expected ground truth. The proposed multi-component loss function enables to observe the progress of locating objects and place greater penalty on bounding box estimation when the object is well located and vice versa. This speeds up the training process as it helps the model to solve the easier task of locating the model first before solving the difficult problem of estimating its bounding box dimensions. Second, a novel sequential point cloud processing method for semantic segmentation is proposed. This uses a sequence of point clouds to generate a prediction. However, as point cloud processing is computationally expensive, processing sequences makes it even more computationally expensive. The proposed method alleviates this problem by fusing point cloud data in a latent feature space instead of processing all point clouds in the sequence each time a new prediction is made. As a result, the method takes advantage of sequential processing while keeping the computational overhead low. Finally, a practical unsupervised method to detect potential collisions in unlabelled point clouds is proposed. The method allows to test the performance and efficiency of different deep learning models on novel data without having to annotate the data first. It is based on the observation that most potential collision areas are defined by the closest object of interest (e.g., a car, a person). Also, the method provides a more realistic assessment of collision probability than widely used aggregate metrics.

Abbreviations

AP Average Precision

BN Batch normalisation

GPU Graphical Processing Unit

IoU Intersection over Union metric

KITTI Vision dataset and benchmark suite for object detection [49]

Lidar Light emitting sensor to measure depth with high accuracy

MLP Multi-layer perceptron

ReLU Rectified linear unit

ROS Robot Operating System

SGD Stochastic gradient decent

SLAM Simultaneous localisation and mapping

SPVCNN Sparse point-voxel convolutional neural network

Chapter 1

Introduction

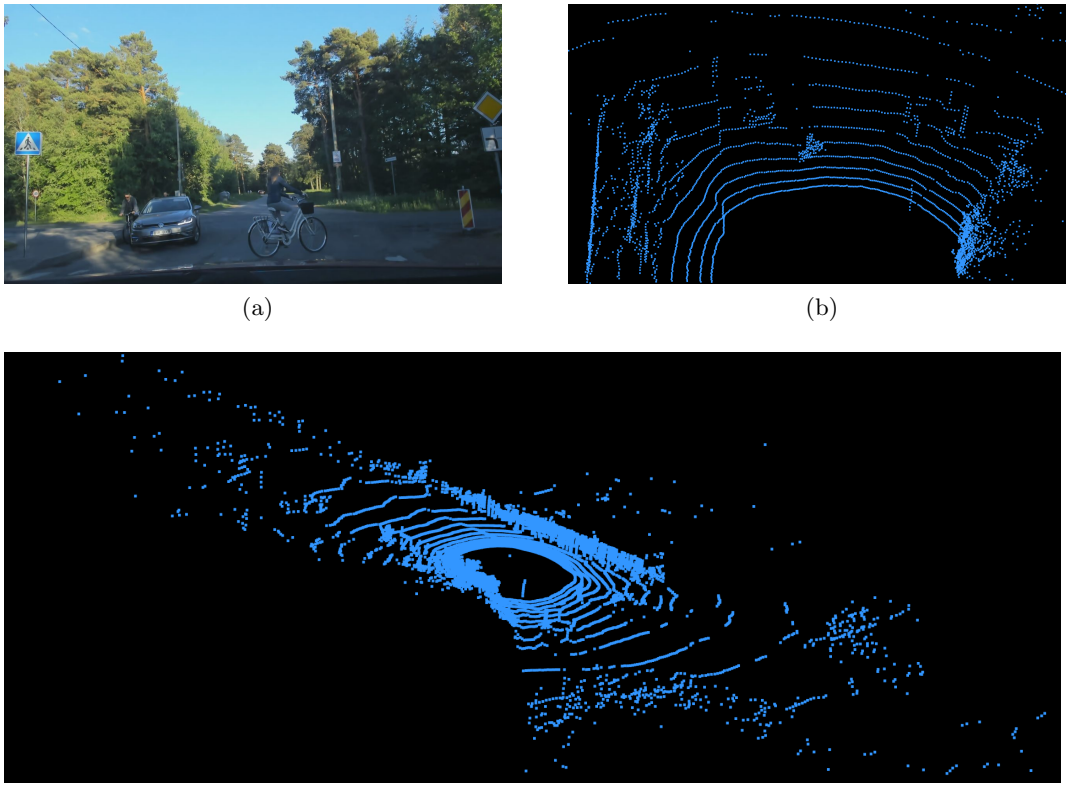
1.1 Introduction and motivation

The goal of this thesis is to advance collision avoidance for autonomous vehicles. More specifically, this thesis focuses on point cloud processing for outdoor navigation systems using deep learning methods.

Point clouds are 3-dimensional scans of the environment. Loosely speaking, point clouds are 3D images. They have an additional depth dimension and are therefore more difficult to process. Point clouds were successfully used for autonomous navigation [25] already in 2005. Their use outside the research community started to grow when open source tool Point Cloud Library [134] became available in 2011. Point clouds are recorded using special sensors. Cameras with a depth sensor (such as Intel RealSense [74]) are often used for indoor recording and lidar scanners (such as Ouster OS-1 [116]) are used for outdoor recording. Lidar is a scanner which sends out laser beams (i.e., pulsed light waves) and records the time when they are reflected back from an object in the environment. Based on the reflection time, a distance from the sensor to the object is calculated. Beams from near objects reflect back quickly while it takes more time to reflect back from far away objects. Unlike a video camera which outputs 2D images, a lidar scan has precise information about the distance of each object from the sensor. This information is crucial for safe autonomous navigation and for collision avoidance.

Most lidars that are used in autonomous vehicles contain rotating lasers. Thus, if such lidar is placed on top of the car, then, unlike a regular video camera, a single lidar is able to record the environment up to 360 degrees around it due to rotation. This enables to develop methods for autonomous navigation that are fully aware of the traffic around the vehicle (e.g., [28], [156]), not only in front of it. Such

awareness further enables to avoid serious collisions such as side impacts. An example of a video image and a corresponding 3D lidar scan are illustrated in Figure 1.1 (a) and (b), respectively. As the figure demonstrates, the point cloud is volumetric, thus, it can be viewed from a different viewing angle than it was recorded. The lidar scan in Figure 1.1 (b) is displayed from a birds-eye-view perspective. Only a fragment that corresponds to the camera image is displayed. The figure also illustrates how the point cloud in (b) makes it easier to visualise the distance between objects (e.g., the distance between the bicycle in the middle of (a) and the car; or distance from the viewpoint of the camera to the cyclist). It is possible as the lidar provides a direct depth measurement. Accurate estimation of distances between objects is crucial for successful collision avoidance.



(a) (b)

(c)

Figure 1.1: An example of a recorded scene. (a) 2D video image and (b) a fragment of the point cloud of the same scene. (c) The same point cloud in full, but from a different view point than (b) and with a different zoom level than (b). There is a circular blind spot (i.e., an empty area) in the middle of (c). The cause of it is explained in Section 4.1

On the other hand, point cloud processing is computationally costly and

requires an expensive hardware. Consequently, working on deep learning models for outdoor point cloud processing was infeasible for many researchers only a few years ago. The first deep learning models that processed large outdoor point clouds efficiently and with sufficient accuracy for autonomous navigation were published around the time this PhD project was initiated (e.g., VoxelNet [183], PointPillars [80]). Even then, handling the full 360-degrees point cloud was challenging. For example, VoxelNet limited the field of view, e.g., similar to Figure 1.1 (b). For reference, the whole 360-degrees point cloud is displayed in Figure 1.1 (c). Evidently, the field of research is young and evolving. There is still a huge potential for new discoveries to be unleashed in point cloud research for safe autonomous navigation and collision avoidance.

1.2 Challenges of collision avoidance

Collision avoidance has several aspects. For mapping and path planning purposes the objective of collision avoidance can be viewed as obstacle avoidance. Obstacle avoidance is a narrower definition and means that a vehicle should choose such a path that it prevents collisions with known obstacles, e.g., the obstacles on the map (e.g., [110]). Some research also considers collisions with moving objects (e.g., [19]). Collision avoidance makes a distinction between collisions that are caused by the vehicle itself, and collisions that are caused by other participants in traffic. Most works on collision avoidance consider both aspects (e.g. [7]). There are also works where the vehicle is learning while moving (e.g., [70]). For the purpose of this thesis, collision avoidance is a goal of an autonomous navigation system to avoid any kind of unwanted collisions. Collision avoidance includes avoiding collisions with known objects (i.e., object avoidance), but also tries to mitigate risks from other sources. These may arise from future actions of other traffic participants.

As opposed to collision avoidance system, this work does not treat collision avoidance as a separate (sub)system of the vehicle, but rather a goal that is being followed over many subsystems of the vehicle. A separate subsystem might observe the environment and upon detecting a potential collision will initiate some action, e.g., stop the vehicle and ask the navigation subsystem to re-route the vehicle. If collision avoidance is an integrated goal of many subsystems, the risk of collision can be mitigated smoothly, e.g., without stopping the vehicle when the collision is not imminent. The research aim of this thesis is to enhance the collision avoidance by designing point cloud processing methods with more accurate perception capabilities. Accuracy of perception is important as collision avoidance is not the only

goal of an autonomous vehicle. A system that overreacts to every minor collision threat is not practically usable. For example, another car that is 200 meters away in a city traffic might pose a danger to an autonomous vehicle if the other car drives recklessly. However, in most cases drivers do not drive recklessly. If an autonomous vehicle would always wait until there are no cars within 200 meters range, then it will be useless in most cities.

In technical terms, there is a conflict of interest at a vehicle level and that conflict must be balanced with care. For example, a collision avoidance goal might conflict with vehicle stabilisation as the work in [46] demonstrates. It is not desirable to avoid a minor collision with low cost which would unstabilize the vehicle such that it would roll over and cause a major damage. There are several ways to address this problem. Authors of [46] and [180] approach this balance as a mathematical optimisation problem on system level. This research is less concerned with mathematical definition of priorities on system level. The main priority is to advance accurate 3D perception methods.

If a perception level of a method is accurate then the system level decision can also be more fine-grained and a possible collision will be easier to predict and handle. The main reason is that all such computations are probabilistic. Humans often tend to express themselves in absolute terms, e.g., "this is safe" or "this is dangerous". Computer algorithms on the other hand utilise probabilistic terms, so equivalent output for "this is safe" could be "0.05% chance of collision". Likewise, every measurement is made with a certain margin of error. Therefore, if a deep learning system outputs that a given object is 5 meters away then its real distance will be within a margin of error of this. For example, if the margin of error is 0.1 meters, the object will be located somewhere between 4.9 and 5.1 meters. Raising the accuracy of perception methods helps to output more reliable probabilities and reduce the margins of error. Therefore, an autonomous vehicle can navigate with more confidence. This in turn translates to safer, faster and smoother driving experience.

1.3 Research background

This research project was initiated by Oxford Robotics Ltd which trades under the Dynium.ai trademark. Although rapidly evolving, by late 2017 lidar technology was rather mature, production ready technology for autonomous vehicles. Several autonomous navigation companies had success with lidar (e.g., Oxbotica, Waymo, Uber). Research on point clouds was evolving rapidly and there were other compa-

nies and universities utilizing the technology in the UK and the Oxford RobotCar dataset was published [99]. Also, the research of autonomous navigation for agriculture made progress (e.g., [10, 129, 130]). This motivated Oxford Robotics to start research collaboration with the University of Warwick on point cloud processing for collision avoidance. The general research objectives for this PhD work were derived from this collaboration.

As Dynium.ai focuses on autonomous off-road vehicles for agriculture, the focus of this research is to advance point cloud processing from practical collision avoidance perspective. Lidar technology is especially suitable as it provides a good visibility even in bad weather conditions (e.g., in rain, in the dark) where the quality of video camera image degrades significantly. Lidar uses laser beams to capture the environment and is therefore not affected by the changes in natural light. More importantly, point clouds that are outputted by lidar are not affected either. While processing an image that is acquired during the daytime is different from processing an image captured during the nighttime, point clouds have the same structure regardless of lighting conditions. This reduces the complexity of processing point cloud data as there is no need to consider natural light.

The emphasis of the research is on advancing methods that are able to work fully autonomously. Such systems will not use any external information, not even existing maps or global navigation satellite systems (GNSS) such as GPS or Galileo. Independence from external systems enables full autonomy. It does not mean that an autonomous vehicle itself would not use GNSS at all, but that point cloud processing models are not dependant on GNSS signal. The vehicle itself will most probably have several concurrent algorithms running, some will be using GNSS and some will not. The optimal performance will be achieved by combining the results of both. However, GNSS will only be useful if both the satellite signal is available and the vehicle can relate location information to existing maps. If there is a problem with either, the vehicle will halt when it misses external data. Dependence on external mapping or positioning is the reason why current autonomous vehicles cannot be easily deployed in new environments. Many systems use self-mapping technology (e.g., [65, 111]) where the vehicle builds the map as it traverses the area. As the research agenda set for this PhD thesis is related to off-road navigation, then methods that require external input or external maps to function properly are not considered.

There is another minor, but still important consideration for this research. Unlike for passenger traffic, the aim of autonomous travel for most other use cases will not be solely the travel itself, but to conduct some task during the travel. For

example, an autonomous tractor needs to cultivate the land. Therefore, the precision and accuracy of point cloud processing models becomes even more important as the route for travel cannot be changed arbitrarily as for passenger traffic, where the exact route is not as important as the final destination.

1.4 Research objectives

The main objective of the navigation algorithm of an autonomous vehicle is to fulfil the goals of the navigation by being aware of the environment around it and plan the navigation according to this awareness. This is different from perception, which just observes the environment. Sensor readings (e.g., images, lidar scans) must be interpreted to understand and utilize them for navigation. This thesis focuses on such interpretation by researching 3D object detection and semantic segmentation deep learning models. Both are state-of-the-art approach for processing point clouds. This thesis aims to advance such deep learning models for point cloud processing. Deep learning models used in this work are all supervised machine learning models, i.e., training of such models requires annotated data.

Building a supervised deep learning model for point cloud processing is not an easy task. First, a lot of point clouds are collected and manually labelled, i.e., someone will annotate where the objects are located in the point cloud, what type of objects those are etc. The result is called a ground truth and such labelled point clouds form a training dataset (e.g., KITTI [49], nuScenes [23], Oxford RobotCat [99] are well-known public datasets). A deep learning model is then trained by showing it the labelled data in a training dataset and tasked to make an estimation, i.e., to repeat the same labelling output that a human labeller produced. The model then outputs its estimation, i.e., labels. Labels are processed by a loss function which is a software component that compares the output of a deep learning model to the human annotated data, the ground truth. Loss function computes the difference between them, i.e., the error. Errors are then fed back to the training process, the parameters of the model are adjusted to minimize the error and the process is repeated until the deep learning model is able to output a similar prediction to the human annotator. However, the objective of the training process is not to re-annotate the training data but to produce a model that is able to generalise on similar, but unseen data. Hence, while each training cycle minimises the error for a specific batch of data, repeating the training process on different batches results in a model that also generalises well on unseen data. Therefore, a learning objective of supervised learning is to minimise the inductive bias, e.g., find a model that on

average works well for many different data examples. As an error on training data does not reflect such ability well, a goodness of a model is mostly validated using a different set of annotated data.

Current research has two deep learning approaches that this thesis contributes to. Thus, the first objective of this PhD research is object detection from point clouds. Object detection was the only practically feasible deep learning method for outdoor point cloud processing when this PhD project started in 2018. The aim of object detection is to output all relevant objects that exist in the point cloud. Relevant objects are those that affect navigation decisions of an autonomous vehicle. Examples of those include but are not limited to cars, pedestrians, cyclists, animals and tractors. As object detection requires outputting object coordinates for each detected object then less relevant objects are mostly left undetected. Estimating the coordinates for every object would be computationally very difficult and also unnecessary.

The second objective is to advance deep learning models for semantic segmentation of point clouds. Semantic segmentation is a process of labelling each point in the point cloud. Instead of object coordinates, such deep learning models (e.g., [181]) output a label for each point. The labels can then be used to further assess the environment around the vehicle. Semantic segmentation of large outdoor point clouds became feasible in 2019 when Choy et al. [28] proposed a computationally effective method for the purpose. Semantic segmentation was incorporated into this PhD project after practical usability was further enhanced by Tang et al. [156] in 2020 when they further sped up the semantic segmentation process significantly for outdoor point clouds.

Object detection and semantic segmentation models are both important components of a modern collision avoidance system designed for complex, non-controlled environments. Their output is be either input directly into a separate collision avoidance algorithm or the model can be plugged into an end-to-end deep learning model where the last block outputs collision probabilities for each part (e.g., voxel, point) of 3D space.

Finally, the output of deep learning models should be evaluated for collision avoidance. The research objective is to develop a method to evaluate models such that the output of the evaluation process is useful for collision avoidance. Therefore, this thesis will assess the models from that perspective, not rely solely on statistics that assess object detection or semantic segmentation as an end goal.

1.5 Thesis outline

The remainder of the thesis comprises:

- Chapter 2 presents a literature overview and introduces important concepts that are essential for later chapters. It also offers a few historical excursions into several topics which aid a reader to comprehend the research field better and also provide context.

- Chapter 3 advances deep learning models for object detection in point cloud. It takes VoxelNet [183] architecture as a baseline and improves the model in several ways. Most notably, it proposes a novel adaptive multi-component loss that enables faster training of the model.

- Chapter 4 advances semantic segmentation for point clouds. Based on the SPVCNN model of Tang et al [156] it develops a novel sequential point cloud processing model. This model uses a novel method that fuses point cloud features in a latent feature space, a practice which considerably decreases the training and running time for the model compared to other sequential models. Both innovations enable to extend the original SPVCNN architecture such that instead of 19 different classes of points, it is able to separate 27 classes of points, including moving objects.

- Chapter 5 proposes a system architecture that is used to run a lidar-equipped test car for data collection during this research.

- Chapter 6 proposes a framework to assess the performance of deep learning models for collision avoidance purposes. It highlights problems that are found using this method and suggests another sequential method to overcome such problems.

- Finally, Chapter 7 discusses the effect and potential of this research and summarises the results of this PhD project.

Chapter 2

Related Research

2.1 Introduction and setting the focus

This chapter gives a high level overview of research related to lidar point cloud navigation and collision avoidance, including relevant methods that are not directly used in this thesis but could be used in alternative solutions. For more intuitive comprehension the chapter is organised such that it presents both the goals of navigation systems (e.g., collision avoidance), and specific designs and methods (e.g., overview of data types, sensors and algorithms).

2.2 Point clouds

2.2.1 Sources of point clouds

Point clouds are 3-dimensional (3D) representation of the environment. There are several ways to collect a point cloud. While lidar, a light emitting laser, is the most straightforward approach, point clouds could also be constructed from RGB-D images. These are RGB images that are enhanced with depth information. Such data is normally collected with stereo camera that is coupled with a depth sensor such as Intel RealSense D435, Microsoft Kinect or similar. However, in this thesis our main contribution is on processing point clouds that are collected with lidar only. The main difference is that RGB-D sensors normally operate in a limited range (useful range of RealSense D435 camera is approximately 5 meters [74]) while lidars (such as Ouster OS-2) can record data within 240 meters [117]. Also, RGB-D cameras have fixed horizontal field of view while most rotating lidars can record 360 degrees.

2.2.2 Types of point clouds

Not all point clouds are the same. While reviewing literature and methods on point clouds it is important to distinguish between methods for different types of point clouds. Some point cloud processing methods are designed for processing only single objects (such as those in ShapeNet [26] or ModelNet [170] datasets). Others work for the whole indoor or outdoor scenes. Indoor datasets (such as SUN RGB-D [148] or S3DIS [5]) are mostly recorded using RGB-D sensors as those sensors are cheaper than lidars and work reliably for most indoor scenarios. However, outdoor point cloud data is collected using lidars as they offer direct measurement for each point. This is essential as outdoor scenes are much more dynamic and also normally cover much larger area than indoor scenes.

Lidar is a rotating laser that emits light waves and is able to measure the distance to an object quite precisely (e.g., Velodyne VLP-16 range accuracy is between 22 to 27 mm [53]). It captures the environment 360 degrees around it (but the range can be capped). Hence it is suitable for mapping (e.g., [179], [65]) and odometry (e.g., [159] [34] [84] [2]). Lidar data are also used as ground truth for training camera-based solutions.

2.2.3 Outdoor lidar datasets

The most labour-intensive part of creating an outdoor lidar dataset is creating the ground truth annotations needed for training and validating the models. Until recently bounding box data (see example in Fig. 2.1) for selected objects types was the most detailed format of ground truth. KITTI [49] is the most used dataset for outdoor point cloud research, providing such data (see also Fig. 2.1). Oxford RobotCar dataset [99] and nuScenes [23], among many others (see more extensive list in Table 2.1), are also widely used. Ford AV dataset [1] lacks official point cloud annotations, but offers data from 4 lidars recording the same scene at once. Since 2019, several datasets have also added point cloud segmentation and panoptic segmentation data, e.g. Semantic KITTI [15] which introduced a ground truth label (i.e., a semantic class) for each point. Panoptic nuScenes [43] and panoptic labels for Semantic KITTI [8] additionally add an instance ID for each point such that all points that are part of the same object have the same instance ID. We use KITTI and Semantic KITTI in this research.

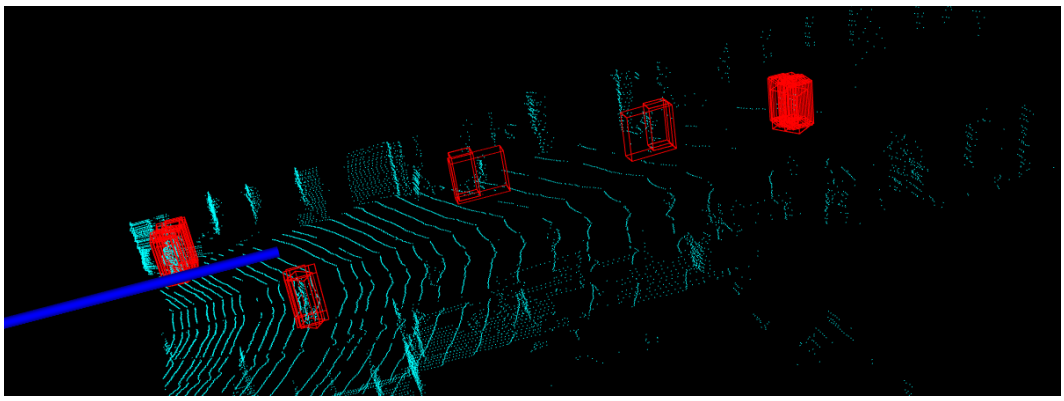


Figure 2.1: An example of a point cloud from KITTI [49] dataset. Lidar points are displayed in light blue. Bounding box candidates generated by our algorithm are displayed in red. Dark blue ray is the x-axis of the sensor. Black is empty space.

Dataset	Bounding box	Semantic	Panoptic
KITTI [49]	Yes	No	No
Semantic KITTI [15]	No	Yes	Yes
KITTI 360 [88]	Yes	Yes	Yes
nuScenes [23]	Yes	Yes	Yes
Ford AV [1]	No	No	No
Oxford RobotCar [99]	Yes	No	No
ApolloScape [149]	Yes	No	No
Waymo Open Dataset [150]	Yes	Yes	No
A2D2 [50]	Yes	Yes	No
Argoverse 2 [169]	No	No	No
PandaSet [59]	Yes	Yes	No
Winter Adverse Driving dataSet [78]	Yes	Yes	Yes
SeeingThroughFog [18]	Yes	No	No
Toronto-3D [155]	Yes	Yes	No

Table 2.1: List of publicly available point cloud datasets and their ground truth features.

2.3 Collision avoidance for autonomous navigation

This section reviews related research for collision avoidance from the perspective of autonomous navigation without external dependency, e.g., offline, off-road and agricultural applications. Furthermore, these methods are also applicable to on-road and urban scenarios.

2.3.1 Operational environment

Off-road collision avoidance differs from on-road systems by the lack of existing maps, and the structure of the environment is less well defined (e.g., lack of road markings, pavements, traffic signs, etc.). Also, this thesis is concerned with an independent system that is able to operate without any external assistance. Hence solutions that are based on global navigation satellite system (GNSS) are out of our research interest.

One of the most prominent cases for such independently working collision avoidance system will be in agriculture. As described in Chapter 1, unlike in most urban traffic scenarios, the aim of autonomous travel is not be solely the travel itself, but to conduct some tasks during the travel. This makes it difficult to apply the results of many existing collision avoidance and off-road navigation research findings as most aim at navigation where the robot either maximizes free space around it (e.g., travelling at the centre of the free space as in [60]) or travels along a safe path (e.g., [25]). Luettel calls this *reactive navigation* [97]. Much of the off-road agricultural navigation is *guided navigation* [97] where the vehicle moves according to a given plan.

2.3.2 Dimensionality

Collision avoidance can be done at many levels of dimensionality. The simplest case is a 2-dimensional (2D) scenario where any spot in the surrounding environment is either occupied or vacant. While suitable for simpler use cases, it omits relevant information for more complicated navigation.

2.5D approach (e.g., [130]) takes into account the height of the obstacle, enabling the autonomous vehicle to drive over a small obstacle without colliding it, but hides information about overhanging objects. For example a 20 cm branch of a tree at 2 meters high will be presented as a 2.20 meters high obstacle.

Finally, 3D collision avoidance will enable all possible scenarios, enabling the autonomous vehicle to drive under the branch or drive both under and above the bridge using the same representation of the environment. Ugenti et al. [160] show that 3D understanding is especially useful on uneven terrain where objects at the same planar surface might be at different heights for the vehicle. They also show that 3D representation significantly improves navigation on such terrain.

2.3.3 Dynamic vs static world

Most research in off-road collision avoidance deals with static world, i.e., the objects are fixed and do not move. This does not mean that moving objects are not considered. They are, but they are mostly viewed as static at any given time instance. This is mostly because off-road vehicles move at slow pace (0.5-2 meters per second) and hence are able to stop or alter their trajectory fast enough to avoid collision. However, if the speed is increased, such approach will not be sufficient as it might lead to collisions. Hence there is a need to predict the trajectory of the moving object. To do that, it is necessary to know about the type of the object. For example, a nearby train is highly unlikely to derail and collide with a tractor, while a child running along the tractor's path poses a great risk. Accounting for the dynamics of moving objects therefore is necessary but also complicates the system significantly.

2.3.4 Mapping

Mapping is used to build a local map of the surroundings. In essence, it provides a memory of what the given vehicle has already 'seen' and hence by using this memory, helps the vehicle to navigate and avoid collision.

While mapping is tightly coupled with the collision avoidance goal, it is also a separate task. Most mapping solutions (e.g., Voxblox [111], Octomap [65]) use probabilistic fusion of new data, meaning that moving objects might not ever appear on a map. Hence collision avoidance can rely on mapping mostly in understanding the static environment, but needs less processed data to assess the existence of dynamic objects.

Modern mapping solutions often regard collision avoidance as one of their goals. Skimap++ [29] is a map that simultaneously does object recognition, so map also contains semantic information. Voxblox [111] enables to look up the distance to the nearest object in full 3D space, and thus it is unnecessary to query all the points in the planned trajectory. Nanomap [42] does not build an integrated map at all - it stores recent measurements and integrates over them when the data is being queried. This has the advantage of not suffering from state estimation errors that would otherwise affect the map construction and hence enables operation under uncertain conditions (e.g., lots of sensor noise and imprecise localisation).

Ort et al. [114] question the need for precise metric mapping in rural environments as precise maps will require huge storage space and will still lose their precision due to constantly changing environment (e.g., vegetation). They demonstrate that local sensing in combination with topological maps will suffice in rural

environments. However, this makes the system dependent on outsourced topological maps.

Although mapping is required for many operational tasks, e.g., navigation and path planning, collision avoidance, conducting the task, the requirements for the task differ. Tasks that consider the immediate surroundings of the vehicle will require more precision during navigation, while path planning can operate using less precise maps.

2.3.5 Odometry and SLAM

Odometry is a means of computing a position estimate from sensor data. SLAM stands for simultaneous localisation and mapping. Both are essential components of an autonomous navigation system which does not use global navigation. Poor odometry will significantly reduce the quality of mapping and collision avoidance. The most widely used algorithm which utilise odometry and SLAM is LOAM: Lidar Odometry and Mapping [179]. In fact LOAM is the commonly used in many lidar-based navigation systems. It has an open-source implementation. While LOAM has a successor, V-LOAM [178], incorporating the measurements from cameras, the successor is less known and researched due to the lack of reference implementation.

The key ideas in visual odometry are keypoint matching and loop closure. The former refers to tracking the same point from one image or scan to the next. Such tracking enables to compute the self-movement of the vehicle. Loop closure refers to a process that will correct the drift in position estimate when a previously encountered point in a scene is re-visited. Given that on the first occasion a certain object is at location (x,y) in 2D space and the same static object is predicted to be on ($x+e_1$, $y+e_2$) on the next visit, then the introduced errors e_1 and e_2 will be corrected by a loop closure algorithm.

Recently, new algorithms have been introduced that outperform LOAM. The key idea is to do keypoint matching and loop closure more efficiently. Both PCE-SLAM [2] and LeGO-LOAM [141] make use of feature extraction from the point cloud. This essentially means keeping only points that are parts of edges and planes. Feature extraction reduces the point cloud size significantly and enables faster and more precise processing. PCE-SLAM further divides the resulting point cloud into batches for motion estimation and processes batches in parallel. LeGO-LOAM segments points to ground and non-ground. Roll (rotation around the forward directional X-axis), pitch (rotation around the Y-axis) and z are computed from ground segments while x, y and yaw angle (rotation around the vertical Z-axis) from non-ground segments.

Another related issue is that most odometry algorithms also build maps (but not vice versa, mapping algorithms mostly do not compute odometry). On one hand, mapping enables loop closure and improves the precision of odometry. On the other hand, although odometry and mapping are run in different processes, mapping slows down the system. This is because the speed of precise, error-adjusted odometry is still bounded by the speed of mapping. Further, loop closure (as seen from the examples of PCE-SLAM and LeGO-LOAM) benefits from the significant reduction of the noise, i.e., needs less data than mapping. For example, PCE-SLAM [2] and DeepCLR [64] compute odometry without mapping. As there are many computationally fast mapping algorithms [65, 111], it might be possible to speed up the whole system and improve the quality of odometry by separating mapping from odometry as the latter usually only requires very recent data points (please see Section 2.5.7 for discussion of practical issues of combining SLAM and mapping).

Finally, mapping an environment by algorithm means incorporating all objects, including those which might evade someone's privacy, especially when 3D point cloud has been constructed using 2D cameras. There is research that is aimed towards preserving privacy [144], but while the contribution of this thesis does not use such direct mapping we are not concerned with the issue here.

2.3.6 Object detection

After data collection it is necessary to interpret the data to understand the environment around the vehicle. While simple driving on a lane might be a relatively easy task, any additional driving requirement will complicate the navigation system. For example, it is possible to separate certain surface from a point cloud by its statistical parameters. Ort et al. [114] amplify the noise in the point cloud and then build a discriminator between the road and the grass as a function of noise. But to separate any further objects would require more layers of data processing. This would ultimately make the system too complex, too slow, or both. Thus, research has moved away from manually programmed control model and shifted towards using deep learning methods. The key idea is that there are far too many edge cases to account all of them manually. As a result, deep learning methods are used to train models that are more robust and able to handle very complex cases of object recognition, object segmentation, depth estimation, image correction and other tasks that are relevant to collision avoidance.

For scene interpretation in 2D, deep learning classifiers (e.g., SSD [92], Faster R-CNN [131]) have been used to classify objects with high precision. Detectors such as Yolo3 [128] have been optimized for very fast real-time usage while sacrificing

some precision. 3D research followed this trend with a delay as the computational complexity is much higher due to the size of the point cloud.

For 3D odometry and keypoint matching, PPFNet [30] is a deep learning model that is capable to match keypoints using modest computation power while offering state-of-the-art performance. They achieve this by finding local features for each patch of the point cloud and then use those features for keypoint matching. Relevant methods for 3D are presented in more details in Section 2.6.1.

2.4 Errors, reliability and uncertainty

2.4.1 Measuring the error

The performance of a classifier can be measured by high level measures as precision, recall, or F1 metric. Using the example of object classification, Intersection over Union (IoU) metric is used to determine if an object is detected correctly against the ground truth. IoU measures the ratio of overlap and union between detected and ground truth bounding boxes [38], i.e.,

$$IoU = \frac{\text{overlap}(prediction, groundtruth)}{\text{union}(prediction, groundtruth)}.$$

A prediction over certain IoU threshold is classified as correct. Average precision (AP) or mean Average Precision (mAP) (mean AP over several object types) are then used to assess the quality of algorithms over standard benchmark datasets. When AP or mAP is measured at certain IoU it is usually referred to with corresponding index, e.g. AP at 75% IoU is referred to as AP_{75}. Authors of Pascal VOC [38] dataset introduced a version of AP that is averaged over many levels of IoU between 0.5 and 0.95 with step size 0.05. Unindexed AP usually refers to this value. They originally used AP metric to benchmark the detection performance of different models on their data, but their AP metric is now widely adapted.

Many widely used classifiers such as RetinaNet [90], SSD [92], Faster R-CNN [131] or YOLO3 achieve Pascal VOC [38] style AP score between 31.2 to 40.8% [90] on COCO dataset [91]. Newer algorithms like YOLOR [164] push this to nearly 60%. It is important to note that AP_{50} or AP for only large objects is normally significantly higher than AP. This means that some objects (mostly nearby and fully visible) are predicted with high precision while some others (mostly smaller, distant, occluded etc) less so.

2.4.2 Uncertainty and errors

Uncertainty is a major issue in autonomous driving. Detecting a human is a solved problem in computer vision, but detecting a human who is harvesting some crops, partly occluded by vegetation is still a challenge for a real time detector. Sudden change of environment is also problematic as the overlap between different sensor scans is minimal [120, p.51,80].

Less research has been done to investigate how to build robust systems in the presence of uncertainty. Kendall and Gal [72] claim that detailed analyses should consider the origin of the uncertainty. They differentiate between aleatoric (noise from observations) and epistemic uncertainty (uncertainty of the model) and show that the distinction enables to build models and loss functions that count for uncertainty [72] (discussed in Section 2.4.3).

Understanding the confidence of a prediction is important for safe navigation. Errors might accumulate at many different levels. Sensors have their error of measurement, and algorithms will introduce another layer of errors - especially when optimisation is used to speed up the computation. However in most cases, errors can also be accurately measured, allowing the vehicle to navigate safely. For example, a mapping algorithm Voxblox has a maximum error of 8.25% due to several approximations to calculate distances [111]. Velodyne VLP-16 lidar, as mentioned earlier, has an 30mm upper bound for the error. Considering both, we can add margins to the vehicle's navigation goals and navigate safely.

As mentioned in previous sections, modern algorithms that are used in collision avoidance make use of deep learning methods. Computing the confidence interval of a prediction made by a deep learning algorithm will be order of magnitude more difficult. While high level metrics like precision, recall, F1 score, and similar indicate the overall quality of the algorithm, there is less research on assessing the quality of any given output. For example, it is easy to say that some deep learning model has high accuracy on a specific dataset but when we use the same model to make a prediction on an unseen output, it is much more difficult to assess how confident the method is on its prediction.

Deep learning models mostly output a prediction and a probability of that prediction. Most common classifier in object detection is some variant of softmax classifier where all predictions made by the model will add up to 1 - so any model might also output several predictions. Yolo3 object detector [128] makes use of independent logistic classifier instead of softmax (e.g., as in [92]) to allow multiple labels with high probability [128] (e.g., "human"', "woman" and "doctor"). This illustrates well that giving any error bounds for the prediction is much more difficult

or even impossible, because we would have to consider at least the following aspects:

- Deep learning classifiers for 3D point clouds are trained on large datasets, containing vast amount of different data. Their ability to deal with unseen data is limited and often unpredictable due to data constraints. Available datasets represent only a small number of real life situations and label fixed number of classes (e.g., 19 classes in KITTI [49]), making it difficult for a model to learn to generalize well.
- Deep learning models are trained using randomisations and optimisations, i.e., even two models with the same architecture might compute their output differently.
- There is no uniform real-world interpretation for the probability of the estimation.
- For practical purposes there is still a need for a binary decision to trust or not to trust the output of a model.

These considerations force researchers to deal with much more complex approach to measuring confidence. While a prediction of a deterministic algorithm could have a well bounded confidence interval, a prediction of a deep learning algorithm has an associated uncertainty which is a lot more difficult to interpret and use.

2.4.3 Bayesian approach

The issues raised in the previous subsection are serious concerns for collision avoidance and navigation generally. One of the first publicly discussed fatal accident with a self-driving car (i.e., Uber Inc. self-driving car killing a pedestrian that pushed a bicycle on her side) [85] was allegedly caused by failing to identify the object correctly while the sensors detected the object in a timely manner [135]. Any threshold on the prediction probability of a deep learning algorithm is either determined empirically or decided arbitrarily, but more importantly - there is little understanding of how to interpret the probabilities outputted by such classifiers.

The importance of understanding and using the uncertainty information in autonomous driving has been recently highlighted by Kendall and Gal [72] for Bayesian deep learning, and by Kahn et al. [70] for reinforcement learning cases. The reason that the literature about uncertainty-awareness is focused around Bayesian learning and reinforcement learning comes from the fact that these approaches have

the means to compute and use such information. Conventional (discriminative) neural network/deep learning approaches are not well suited for capturing uncertainty as normalised score vectors of deep learning models do not necessarily measure uncertainty [72]. Measuring uncertainty becomes especially relevant when processing novel, unseen information. The model may then make erroneous prediction [70]. Uncertainty-aware solutions can provide the level and nature of uncertainty for their estimation, and a vehicle control algorithm can then vary the behaviour accordingly.

Detailed analyses consider the origin of the uncertainty. Kendal and Gal differentiate between aleatoric and epistemic uncertainty [72]. Aleatoric uncertainty originates from the noise in the data and epistemic uncertainty raises from the model parameters. This approach works by modelling different probability distributions over model's weights (for epistemic) or outputs (for aleatoric) [72]. While authors of [72] mostly concentrate on using this information to improve the model itself, an important implication for collision avoidance is to gain knowledge of the source of uncertainty. For example, a situation where a model cannot fully distinguish if some object is a cat or a dog due to uncertainty in a model, is different from a situation where there is too much noise to understand the nature of the object. In one case, a system might operate with a knowledge of a nearby animal, while in the other case it should either use different sensors or halt the operation if this is not possible.

A related approach that can be used to asses the model uncertainty of existing deep learning models (i.e., does not require special architecture nor re-training) is to use a widely used regularisation technique dropout. Gal and Ghahramani show that using dropout can be interpreted as an approximation of a Gaussian process [47], where uncertainty can be captured by collecting estimates during randomised executions of the model [47].

2.5 Perception and sensors

Perception is the combination of prior knowledge and current sensor readings [44]. For on-road navigation, the most obvious form of prior knowledge is a street map. Although an off-road vehicle can use maps based on satellite imagery, the spatial resolution is far coarser than for road maps, e.g., minimum resolution for Sentinel-2 satellite starts from 10 metres [33]. Although there is no global map, it is still possible to map an area for a specific project. More general form of prior knowledge is to use pre-trained models, such as deep learning models to process and interpret sensory data.

2.5.1 Selecting sensors for collision avoidance

First, the selection of sensors must support the type and application of the system. Secondly, it must provide a comprehensive data about the environment under different conditions. Systems that deal with rather uncomplicated terrain and conditions, usually only use some form of vision system. The most popular is some combination of video camera, lidar (e.g., [130]), radar, ultrasonic sensors or infrared sensors (essentially a restricted form of radar). Camera choice is often made between monocular camera (i.e., regular digital camera), stereo camera (i.e., two or more cameras installed on a fixed baseline), and omnidirectional camera [4].

Systems that need to work on rough terrain (e.g., muddy, uneven, etc.) also pay attention to sensors that capture vehicle movement (e.g., [60]), e.g., using gyroscope or inertial measurement unit (IMU) to determine the angular rotation of the vehicle, and wheel encoders to measure the slippage of wheels.

Each sensor has its strengths and weaknesses, and it provides different data about the surroundings. For example, radar directly provides relative velocity in addition to detecting an object [97], lidar can "see" well in the dark [97] and camera can provide colour information. Different combination of sensors is needed under different circumstances.

In this PhD project, collision avoidance is considered as a goal of the autonomous vehicle which depends on the sensors and other components of autonomous navigation. Hence it is important to consider all aspects of the system, including the hardware. The following gives an overview of the importance of different sensors for avoiding collisions, and their importance to this research.

2.5.2 Vision

The importance of vision lays in two main categories. In addition to detecting the objects in the environment, vision is the main source of odometry, i.e., position estimation (if GNSS is not used). There is a variety of sensors that are able to provide vision data. Our research focuses on lidar but for completeness we will give an overview of alternative approaches, each of which has its benefits and drawbacks.

Monocular camera

One of the simplest is a digital camera (either monochrome or colour) that is often referred to as *monocular camera* to contrast it to *stereo camera*. A decade ago the usage of monocular camera data was limited to detecting and classifying segments in an image [25]. Only limited progress was made in understanding the semantics of the

environment (i.e., nature of terrain, types of objects etc.). Ollis et al. [113] observed that certain object are associated with same colours and used neural networks to train a model that computes the cost of traversal for each colour in the image, and were successful in using single image for off-road navigation. Current research [54] [100] is now able to use monocular camera for depth recognition, an essential part of safe navigation. The models in those approaches are mostly still trained using stereo camera or lidar data for ground truth.

Stereo camera

Stereo camera is composed of two or more monocular cameras. These cameras are normally mounted on a same rigid bar with a fixed baseline. It has been in the focus of autonomous driving research for more than a decade because it is possible to reconstruct depth information from a pair of images taken at the same time. Works until recently [25] [24] relied on computing disparity (i.e., difference in distance for the same real-world spot between images). Disparity information allows to reconstruct the properties of a full 3D space and hence compute the real-world geometrical properties of any object in the image (some solutions [24] directly operate in disparity space without constructing the 3D scene). The research on finding computationally effective means of calculating disparity information has relied on direct mathematical approach. Recent research [54] [100] has moved to reconstruct depth information using deep learning method, mostly based on convolutional networks that take stereo camera images at the training time and then use monocular camera on run-time. The advantage is significantly simpler system architecture at run-time [147] (i.e., no need to keep stereo cameras on board) and hence reduced cost. Lidar is a comparable alternative to stereo camera and provides depth information directly, but lacks RGB (colour) data.

Event-based sensors

Event-based sensors are not new, but still emerging. Instead of a whole image they output the change in intensity for each pixel and do it asynchronously and very fast (up to 1 MHz [184]). Hence motion information comes without additional computational overhead and with high frequency. These have been used in prototypes to steer autonomous robots [107] and to predict the steering angle [101].

Lidar and radar

Lidar is laser scanner that emits light rays and measures the distance and intensity of their reflection (different materials have different reflectance). Most lidars used in autonomous navigation are rotating and hence have up to 360 degrees horizontal field of view. Because of this lidars are also called lidar scanners. Their output, a point cloud, is a combined measurements within one turn of rotation, so there is a slight time difference between first and last points in the same point cloud. In this project, we use Ouster OS-1 lidar (see Figure 2.2) which rotates at 10Hz or 20Hz. Thus, at 10Hz the points in the same point cloud can be recorded with 0.1 second difference. Their main benefit with lidar is that it is not affected by darkness and work well in difficult weather conditions.

(a) Vehicle view

(b) Close up view

Figure 2.2: Ouster OS-1 lidar mounted on our research vehicle.

Radar is not widely used as a main sensor in published autonomous vehicle research, but notably Tesla Inc. uses radars on their autonomous vehicles. Radar emits radio waves which have much longer working range than light waves emitted by lidar. Unlike lidar, it is directly able to measure the speed of the moving body. Generally radar data require more complex interpretation than lidar. Thus, they are used for collision avoidance in scenarios where precision is less important, e.g., in maritime collision avoidance [36]. On the other hand, due to radio waves reflecting from surfaces and changing their direction, it is possible to perceive fully occluded objects (e.g., pedestrians around the corner) [136].

2.5.3 Non-vision sensors

In addition to vision, most autonomous vehicles have some kind of inertial measurement unit (IMU) which measures the movements of the vehicle using gyroscope,

accelerometer, magnetometer or other means. Wheel encoders can be used to measure the speed and compute the slippage of the wheel.

2.5.4 Sensor fusion

Sensor data can be fused on raw data level, feature level and object detection level [97]. The motivation for fusing sensor data is to: (a) level out the weaknesses of individual sensors by combining data from several sensors; and (b) to offer a wider range of view as different sensors have different ranges and fields of view. Non-linear versions of Kalman filter and particle filter [60] have been used to achieve these objectives. In recent studies however, the fusion of data often becomes a question of designing an appropriate deep neural network architecture that takes input from different sensors and outputs a unified result. There is one exception though - a fusion of several recordings from the same sensor, mostly for mapping (e.g., [111]) - where algorithmic approach still prevails.

Every sensor has its strengths and limitations. While lidar is very precise, its data are very sparse compared to a camera. While images lack depth data, they contain colour information to make object segmentation easier. So it is useful to rely on many sensors and to fuse their output for better quality and more certain outcome. This is the architecture that most autonomous vehicles adopt.

Non-vision odometry allows to fuse position estimate with that computed from vision to account better for the difficult maneuvers. Mostly either non-linear Kalman filter [16] [140] or particle filter [10] [60] has been used to fuse the odometry information.

Many vision-based algorithms rely on keypoint matching (see Section 2.3.5) to associate consecutive images or scans. This works very well for smooth trajectories, but a sudden change of environment is problematic as the overlap between scans is minimal [120, p. 80]. Sensor fusion can be used to address this (e.g., additional information from a wheel encoder).

2.5.5 Sensor calibration

Most sensors need intrinsic calibration to assure that they function properly. Effective collision avoidance systems also need calibration between sensors to bring all data into a common reference frame [119] for comparison and fusion. Static calibration where parameters are set and then fixed for the system is the easiest and most obvious way as the calibration can be done in laboratory conditions. The need for online calibration, i.e., calibration that takes place while the vehicle is

operating, arose because static calibration is not able to correct calibration errors that happened during the operation of the vehicle [86]. Such calibration is based on comparing data from different sensors and finding the most precise transformation as in [86, 87]. Additional external equipment can also be used, e.g., Pereira et al. offer a very precise method, but it requires special environment setup by placing a calibration ball in front of the vehicle [119].

In addition to comparison and fusion, sensor calibration is also necessary to convert 2D data to 3D in stereo vision. 3D depth image can be generated from a pair of 2D images by matching features from both images when the pair of cameras is calibrated intrinsically. Real units can be associated with depth values when extrinsic calibration has been performed [130]. To optimise the stereo reconstruction, a rectification matrix can be computed upon calibration which will be used to transform images so that any row of pixels from the rectified image of one camera can be mapped to the corresponding row on the rectified image of the second (or third) camera [106]. In addition to spatial calibration, this process needs a very precise time synchronization so that the shutters would operate at the same time [4]. Most systems use a camera with global shutter to make the operation easier as such camera exposes the whole scene at once. But such cameras are more costly than cameras with rolling shutter which record the image row by row, hence motivating research [83] to remove the distortions caused by rolling shutter [67, 83].

2.5.6 Processing and filtering raw data

Why process data before using it? First, modern sensors can output more data than are feasible to process on board the vehicle. Second, some data are most useful in its accumulated form. For example, trajectory estimation requires information accumulated at different time instances, path planning benefits from mapping, stereo images can be turned into disparity or 3D data to get the depth information. But input data are also noisy and it is sometimes beneficial to remove the noise before further processing.

Current research on sensor data filtering for autonomous vehicle systems mostly concentrates on effect removal. Deep learning models are trained to remove shadow [127], rain [45, 174], reflection or other occlusions. There are also some research on mathematical methods to remove effects, e.g., [68] to remove rain from images, but they are constrained to selected scenarios. For example, [68] is generally fast and effective, but struggles to remove rain that is falling under acute angle to the ground.

2.5.7 Mapping and localisation

Mapping is more a necessity than an option when 3D sensors are used. The amount of collected data is way too large to operate with lengthy sequences of raw data in near real-time applications. Iterative Closest Point (ICP) algorithm and similar solutions are commonly used to fuse several measurements into a map. The main challenge is to find a transformation between measurements as the vehicle moves[1]. This can be done in real time, for example an early solution back in 2006 ran ICP solution at 3 Hz [120, p. 74]. As localisation or odometry (i.e., determining position and orientation) also requires the computation of transformation, then mapping and localisation are often done simultaneously and referred to as SLAM - simultaneous localisation and mapping [20]. The main concern is that the output of SLAM solutions tends to drift in time [105]. One of the effective ways to determine and compensate for the drift is to detect loop closure, i.e., new arrival at the already visited place [61]. As localisation with ICP is an optimisation problem, the solution is prone to reaching local optima [105]. The benefits and deficits of using different ICP variants are well summarised in [121], where the baseline for comparison is established, and the very broad conclusion is that different ICP variants are suitable for different conditions.

LOAM [179] is a widely used SLAM algorithm which decomposes the problem into two parts: one part of the algorithm runs fast and computes transformation at 10Hz; and another part compensates for the drift, but is much slower and runs at 1Hz [179]. As LOAM is only good for 3D point clouds, V-LOAM that has a similar working principle, but additionally uses monocular camera data to speed up the initial motion estimation is proposed in [178].

Map update finally takes place after the transformation between a new measurement and the reference has been found. The complexities in mapping are mostly related to updating, storing and fetching data efficiently [65], i.e., about the data structure and update policy. OctoMap [65] is a widely used solution and suitable for collision avoidance as it offers efficient 3D mapping. It is also able to differentiate between empty and unmapped areas, and takes into account the noise and uncertainty in the system [65]. Thus, it does not use boolean occupancy status, but probabilistic status, allowing errors to fade and states to change over time [65]. Octomap is a mapping solution and does not compute a transformation itself. For an integrated solution, Asvadi et al. [7] proposed a voxel-based approach that integrates scans using ICP and includes object detection, achieving the speed of 0.3

[1]For 3D lidar data the process of finding such transformation is often called point cloud registration

fps on a 3.4GHz quad-core computer. This is many times slower than Octomap combined with the LOAM algorithm [179] to provide transformations. Voxel size also affects performance: smaller voxel size provides better accuracy but is slower to compute. Octomap is superior here as it offers multi-resolution approach, optimising the areas where there is no need for smaller resolution [65].

Mapping is an active field of research (e.g., [29, 35, 42, 111, 112]). It is often combined with deep learning methods that are replacing ICP to achieve point cloud registration (e.g., [64, 96]).

2.6 Understanding the environment for navigation

Perceived data can be interpreted on different levels of detail and on different number of dimensions. The simplest form is to separate traversable ground from non-traversable on a specific moment of time. From there, the interpretation of the data can be expanded in time and space as well as in detail.

The early successful attempts of map-free autonomous navigation on unmarked routes were built for off-road navigation. These were a combination of several deterministic methods to find a traversable path and prevent collision. For example, a DARPA challenge winner in 2005 applied path detection and object detection for navigation [25], Milella and Reina built similar navigation system for outdoor vehicles [106] as did Ollis et al [113] and many others. Object detection in all of them was understood as separating any object from traversable ground without determining the type of the object. Such approaches often assume static world, i.e., they operate with a snapshot of data at a given time and without the ability to predict events. Some map their environment (e.g., [25]), so they have a limited memory.

Restricted and controlled environments also allow to use simple approaches to avoid collision. Vougioukas [163] equipped each orchard worker with a radio transmitter to localise them with respect to the vehicle for collision avoidance. Automated vehicles can also follow a human-driven master vehicle [108]. While effective, such approaches bring along a need to maintain additional equipment and have a limited use.

Deterministic and probabilistic algorithms that use human-engineered features have their limitations. The fundamental problem with direct algorithmic approach is that while collision avoidance is based on the laws of physics, the properties necessary to estimate a safe path are inferred from very indirect sources [71]. Additionally, probabilistic approaches based on statistical modelling (e.g., Extended

Kalman Filter) depend heavily on correct feature extraction [60]. Human-engineered feature extraction quickly becomes a very difficult problem as the complexity of the data grows. Therefore, direct algorithmic approach is infeasible and current research has moved towards methods that are able to extract complex features and hence generalise better. The most dominant technology in computer vision is deep learning, especially the use of convolutional layers of neural networks. These allow to automatically discover features from images, and those features can then be used to approximate suitable functions for navigation and collision avoidance.

2.6.1 Using convolutional neural networks

Although the idea dates back to 1960s [81], convolutional neural networks became popular in 1990s when it was demonstrated how to use them in practice. LeCun and Bengio showed on handwritten digits that one can eliminate manual feature extraction [81]. The idea of a convolutional layer is to apply a learned kernel to an input image and the output is considered to be a feature vector. This feature vector can be further reduced in size by pooling, i.e., keeping only one value for every small area [82]. Finally the output of many such convolutions is most often fed into a fully connected neural network which classifies the images [82]. This approach allows to reuse existing convolutional neural network models that have been trained on a huge dataset (e.g., by Google, Microsoft, OpenAI etc.) and only retrain some layers to achieve the goal of any particular navigation or collision avoidance system. An example of such re-engineering approach is [123] where Provodin et al. used a convolutional network that was trained on ImageNet dataset and adapted it for separating ground and objects in off-road conditions. A standard convolutional layer is a rather simple building block, while most point cloud processing research concentrates on finding the best architecture to combine those building blocks.

Standard convolutional neural network layers need dense data like images to provide an acceptable solution [44]. They do not work well for sparse lidar point clouds. This was first addressed by converting 3D to 2D as in [153], but developments in hardware made it possible to perform on 3D data directly [102, 103]. Although the sparseness limit still applies - these approaches are using an accumulated 3D map, not a single scan as an input.

The next phase was dominated by solutions that divided point clouds into local regions and processed aggregated information densely using convolutional networks. For example, Voxelnet [183] and SECOND [172] divide a point cloud into equally sized voxels, computed a feature for each voxel and then processed those features in a dense form as for the image. PointPillars [80] is a similar approach

but uses vertical columns (pillars) instead of voxels. Those methods work reliably for finding coordinates and bounding boxes of objects in a point cloud. But they are not suitable for semantic segmentation, i.e., are not able to classify each point as they work with local areas (voxels, pillars), and hence lose the ability to track single points.

Recently, the work in [28] introduced sparse covolutions that are able to process sparse data densely by mapping data in GPU memory. Further, the work in [156] offered a similar but improved version that combines sparse convolutions with voxelisation, keeping both local area features and point-based features. Finally, all those approaches deal with classification. At the end of each such deep learning network there is normally a simple neural network which outputs the prediction based on computed features (which are represented by vectors or matrices). It is also possible to learn a characteristic feature for each object type, called embedding. For example, self-driving car company Waymo uses embeddings, i.e., dense vector representations of features that carry some semantic meaning [44] and preserve pairwise similarities of original features [63].

2.6.2 Structure from motion

Determining the structure and motion is important to assess the safety of traversal and to make decisions on stopping or changing the movement of the vehicle. Important concepts that are used to understand the structure of the environment are object detection (to determine the type of object), instance segmentation to differentiate between different instances of objects, e.g., two persons), and flow of the objects in time.

These objectives are interlaced as research shows that it is efficient to combine their computation. Structure from motion (SfM) derives 3D structure from several monocular images. It has been proved that given three unique views of four non-coplanar points, it is possible to determine the structure and motion for the views [161]. State-of-the-art open source solution is COLMAP [137] which can be integrated into navigation systems to improve the performance. For example, the V-LOAM [178] SLAM algorithm, integrates SfM to increase speed and robustness. Combining SfM with lidar data, it achieves the effect of images that have depth information for each pixel (RGB-D) with special hardware (such as Kinect sensor). A significant drawback of deriving 3D information only from RGB-D data is that moving objects must be filtered out [138] to get the correct results.

2.6.3 Optical flow and scene flow

Dynamic objects are of great interest to avoid collisions, hence the research fields of optical flow (from 2D data) and scene flow (from 3D data) are determining the motion between images. Current research takes advantage of the need to acquire both flow and structure, so combined approaches are developed. Wulff et al [171] provide depth structure, optical flow and segment image into dynamic and static regions; Taniai et al. [157] are able to predict scene flow, camera motion and detect moving objects from moving stereo images. The difficulties lie in texture-less surfaces and very large displacements [13], so approaches to track objects while calculating scene flow have been proposed [13]. But optical and scene flow are not the only possible solutions.

The state-of-the-art solutions are now using event-based cameras. These output the change in intensity for pixels in the field of view asynchronously, operate in a very high speed up to 1 MHz and hence have high temporal resolution [184]. Due to the nature of the sensor, motion information comes without computational cost. Such cameras have already been used to steer autonomous robot [107] and to predict the steering angle [101] using deep learning models.

2.7 Making decisions

While it may seem straightforward to decide on the course of action when an obstacle is detected, there are difficulties. Fulfilling the collision avoidance goal might destabilise the vehicle, putting the decision into conflict with stablisation goal [46]. While in some areas of autonomous navigation a certain error rate is tolerated, collision avoidance systems are expected to work flawlessly as can be seen from reactions to the recent incident with Uber car in Texas on March 2018 [85]. Different control algorithms have been developed for this. One solution is to find an optimal strategy over different goals of the vehicle without approximation, prioritising collision avoidance [180]. Alternatively, Wang et al. [167] use approximation to take into account more variables. Higher level solutions include embedding collision avoidance goal into path planning system [17] or end-to-end machine learning models that take sensor readings as input and directly output driving directions instead of object detection [73].

Reinforcement learning, an independent paradigm of machine learning which is based on rewarding desired actions, is promising for collision avoidance [69]. A reinforcement model get rewarded for safe journeys, but also learns from the mistakes it makes while guiding the vehicle via its output. Some research is directed towards

making such learning safer, e.g., by making it aware of uncertain situations where the vehicle can reduce speed [70]. Such approach requires the use of simulations as learning from real life collisions is not an acceptable research practice outside the lab environment.

2.7.1 Performance

It is beyond question that any collision avoidance system must operate near real time, but the performance also introduces other constraints. The range and speed of the collision avoidance system also limit the speed of the vehicle [97]. The speed of the vehicle is affected by the ability to foresee the changes in the environment. If we can track the objects around the vehicle and estimate their trajectory, we can compute more certain trajectories for our own vehicle. Direct algorithmic approach is usually not adequate for this complex goal at higher speeds. For example, a robot that navigates between straight rows of plants - one of the simplest possible scenarios for off-road vehicles - using particle-filter based approach [60] achieved the processing power of 10Hz, which enables operating at the speed of 0.7 m/s or 2.5 km/h. Other algorithmic solutions that mostly do not have memory or prediction ability [106, 113] achieve a similar or slightly higher speed. Deep learning solutions mostly require more computations but have a similar frequency on special hardware. Yet they are able to process more complex cases, covering more territory and hence enable faster speeds for the vehicle. There are several reasons for this. First, deep learning models use matrix and tensor operations which can be optimised to a high level while algorithmic solutions depend on step-by-step sequential processing, e.g., work in [113] uses pixel-by-pixel search heuristic, work in [106] divides sensor input into patches, computes geometric properties for each patch, finds relative distance and distribution specific cutoff values. While a tensor operation might involve much higher number of computations than a direct algorithmic approach, those operations are batch-processed on a specialised hardware (most often on a GPU) and are hence much more efficient than sequential algorithms that require passing values between CPU, GPU and the memory of the computer. Second, algorithmic approaches [106, 113] have both long-range and short-range components with continuous update mechanisms where short-range performance naturally limits the speed of the vehicle. Modern deep learning classifiers (e.g., [27, 156]) may output the prediction for the whole 360-degrees point cloud in up to 150 meters of range in a single pass.

2.8 Conclusions

There is an active research in almost every field that is relevant to autonomous navigation and collision avoidance.

Solving collision avoidance problems with direct algorithmic approach is not an easy task even in structured environment. For off-road, e.g., in agriculture, changing climate conditions and vegetation, and even missing plants in a row can be difficult to deal with [60]. In addition to avoiding collision, the autonomous vehicle should be able to drive at desired speed, keep intended trajectory, etc. As the complexity of the vehicle and the environment increases, collision avoidance goal will be related to more subsystems and tasks. Deep learning methods prevail in such systems as they make it possible to deal with complexities that are far beyond the reach of any direct algorithmic approach. In addition to improved understanding of the software design, hardware also keeps up: 3D solutions are feasible in real time, and event-based sensors are available. Assessing the uncertainty is so far one of the most difficult aspects, which is mostly compensated for by redundant systems and sensor fusion.

While there are many performing systems for map-free collision avoidance, most are meant for a specific task (e.g., see [140] for application in orchard, or [12] for other examples).

There are also many related fields not discussed here, e.g., vehicle-to-vehicle communication can improve the results of operation [97], but our aim is foremost to build a fully autonomous system. Preventing adversarial attacks [48] in deep learning models is also an important and related topic (e.g., how to prevent deceiving navigation systems), but is not in the scope of this PhD research.

Odometry, mapping and collision avoidance are very tightly coupled components as they all require processing of the same data and may use each other's results. Precise mapping needs good odometry data, and collision avoidance relies on mapping. Path planning further requires both maps and collision information. Thus, there is a need to research different components of autonomous navigation. At the end, the perfect system will not be a separate collision avoidance system, but rather an autonomous navigation system for off-road vehicles that is able to avoid collisions.

We also consider uncertainty to play an important role in autonomous navigation and collision avoidance. It is important to recognise that at the low levels of the system one can operate with the notion of confidence and is able to set the error margins for the measured or computed result. As we move up to higher levels

of the system, closer to the layers where collisions will be predicted, we need to talk address uncertainty. Giving exact margins of error will neither make sense nor is it possible for the state-of-the-art deep learning algorithms. There is a need to understand the inherent uncertainty in models that are built with such algorithms because the nature and the amount of uncertainty will affect the reliability of the collision avoidance estimations. However, when as system is built up from many subsystems then the errors tend to accumulate. To minimise this risk, systems that have fewer components (e.g., end-to-end deep learning models) have an upper hand here as those take sensor data as input and output high level information for navigation. Such systems are less prone to error accumulation and to integration errors. However, they are not as easily explainable and shortcomings in middle layer logic are much more difficult to discover during testing as often only the end goal is testable.

Chapter 3

Advancements in Voxel-based Object Detection from Point Clouds

3.1 Introduction

Object detection is a crucial component of autonomous navigation platform. While it is possible to navigate without detecting objects (e.g., by detecting only traversable ground as in [130]), fast and safe navigation requires a more detailed understanding of the environment. Object detection is a versatile approach to understand the environment. Object detection algorithms are able to locate an object and detect it's type or class (e.g., car, person etc). This chapter contributes to voxel-based object detection by introducing:

- Adaptive multi-component loss function to speed up a training process and improve training accuracy;
- A novel convergence metric based on frequency distribution of output probabilities of a deep learning model that helps to compare models and quickly prune poorly performing models.

This chapter is structured as follows. First, it offers an overview of object detection from 3D point clouds. Second, it analyses VoxelNet [183], a 3D object detection model that is used as a baseline model for the rest of the chapter. The chapter then discusses loss functions and data augmentation techniques for 3D object detection. Third, the chapter details the unique contributions of this thesis: an adaptive multi-component loss function, a novel method to asses convergence for

the purposes of pruning under-performing models in the training process, and offers several minor improvements over the baseline model's training process.

3.2 Overview of object detection from point clouds

Lidars have been used to detect objects for over a decade now. However, early works ([21, 122, 153, 177]) used them mostly as auxiliary sensors in combination with a camera and the use of neural networks was limited. Deep learning based object detection from point clouds of large outdoor scenes became feasible for processing with increased computing power provided by graphical processing units (GPUs). While all neural networks hugely benefited from GPUs, previously it was not feasible to train and run sufficiently efficient neural network models for point clouds on regular processors (CPUs). However, even modern state-of-the-art GPUs have insufficient computational power to allow training deep learning models from point clouds by just concentrating on the quality of model. Early works on full point cloud processing [103, 125] worked so heavily on computational optimisation that optimisation was often more important contribution of the research than the actual object detection or classification mechanism. Due to their size, point clouds are computationally an order of magnitude more difficult to process than 2D images. Also, a large part of developing a 3D deep learning model still consists of working with optimisations for speed and efficiency. This is also the reason why successful 2D model architectures (e.g., inception networks [154], residual connections [58]) are not instantly and directly applicable for 3D models - they would be infeasible or very slow to train and use. Many ideas make it from 2D to 3D models after extensive research. Most point cloud methods are first developed for point clouds of limited size. Examples include ShapeNet dataset [26] that contains only shapes of objects; or SUN RGB-D dataset [148] which contains point clouds of indoor scenes. Models using those datasets often use 2000 - 8000 points per point cloud (e.g., [51]). Large outdoor point clouds in KITTI [49] contain 10-20 times more points and methods suitable for small point clouds must be further optimised for processing those. A significant contribution to the small scale point cloud processing was the PointNet [126] architecture which consumes points and processes them using standard multi-layer perceptron neural networks. Those networks learn features and those features are aggregated to higher level. Features can then be further used for object detection, semantic segmentation and for other tasks. A breakthrough in object detection from large outdoor point clouds was achieved by VoxelNet [183] which utilises a point-based feature learning method similar to PointNet [126] but learns features for each voxel. After VoxelNet

was published the field of outdoor point cloud object detection evolved rapidly.

PointPillars [80] used vertical columns, pillars, instead of voxels. Frustum ConvNet [168] proposed slicing the point cloud according to the depth into units called frustums. SECOND [172] used voxels, but replaced dense 3D convolutions with specially designed sparse layers. Some works took a different path. PointRCNN [143] and RT3D [177] find regions of interest and process those for detecting objects.

3.2.1 General approach towards 3D object detection

3D object detection consists of solving two problems: locating the object in the point cloud (i.e., finding its coordinates) and determining the size of the bounding box which encloses the object. While in 2D there are many 2-stage detectors that first generate candidate bounding boxes and then refine and classify them, most 3D models are one-stage detectors. The most prominent approach is to use Region Proposal Network (RPN) [131]. The technical question that RPN answers is how to locate the objects. First, the search space (i.e., the point cloud) is divided into equal regions, the centre of each region is an anchor point. One or more possible anchor boxes are attached to each anchor point. RPN was originally used for 2D but quickly adapted for 3D. Alternatives to dense RPN also exist. PointRCNN [143] dynamically finds foreground points and generates candidate boxes for each such point. Candidate boxes are then refined at a later step. Guided anchoring [165] is a similar approach developed for 2D which first uses semantic features to place anchors only where objects are likely to exist.

This research follows [183] and implements dense RPN for 3D (see Figure 3.1). Before training the model, each anchor box is validated against the ground truth to see if they overlap with any ground-truth bounding box. Overlapping anchor boxes are marked as positive and non-overlapping as negative. These annotations guide the training process. Unlike the original 2D version in [131] this chapter improves anchor box generation by building a full 3D anchor box generation and validation method. Compared to simplified 2D version that is used in many 3D works (e.g., in [172, 183]) it allows more precise height estimation. It also makes possible to utilise height parameter more effectively in loss functions, see also Section 3.4.

During the training phase, a model classifies each anchor box (assigning a class probability for each box) and does a bounding box regression, i.e., estimates the dimensions of the object enclosed by the bounding box. It is not expected that a positive anchor box overlaps fully with a ground truth bounding box. The bounding box regression computes differences between the anchor box and the bounding box

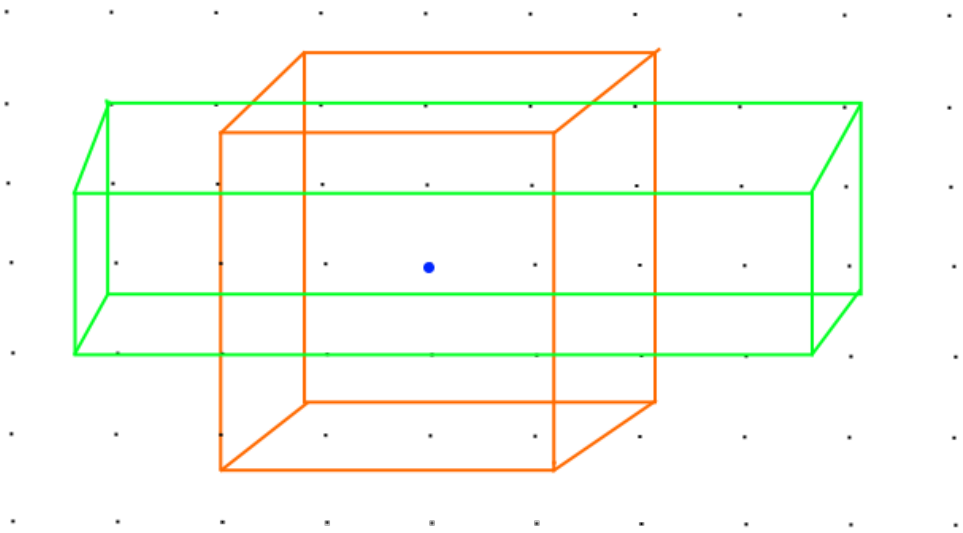


Figure 3.1: Concept of dense RPN as used in this research is based on [131] but adapted to 3D. For illustrative purposes, the space is represented as 2D and anchor boxes as 3D on this image. Grey dots represent anchor points. Each anchor point has the same set of anchor boxes. Here only 2 anchor boxes (orange and green) are drawn for a single selected anchor point (blue). The model classifies each anchor box, i.e., assigns to it a probability of containing an object and then does a bounding box regression, outputting the size and rotation difference between the anchor box and a ground-truth bounding box.

of a detected object. Anchor boxes are needed to guide and provide feedback during the training process.

The output of the model is therefore a tuple of tensors that contain both classification output and bounding box regression results for each anchor box. For example, given that N point clouds are processed at once, each having $L \times W \times H$ anchor points (because there are 3 dimensions), each anchor point having B anchor boxes and each anchor box having 4 properties (length, width, height and rotation), and there are C different object types to be estimated, then the output consists of 6-dimensional tensor N x L x W x H x B x 4 for bounding box regression and 6-dimensional tensor N x L x W x H x B x C for object detection and localisation.

3.2.2 Voxel-based approach

Many point cloud processing models are built by some kind of dimensionality reduction, mostly flattening the point cloud to a 2D multi-view or 2D birds-eye-view (BEV) format [6, 77, 173]. These solutions mostly still keep exact depth informa-

tion for points (i.e., recorded distance from the sensor) but the depth is merely a feature of a 2D pixel. Such models are not able to use neural network architectures that utilize spatial dimensions to fully capture 3D geometry, e.g., 3D convolutions, graph convolutions etc. Of course, some used mechanisms alleviate this by using cuboids [41] or multi-view, e.g., building several 2D views (images) of a single 3D point cloud, or by simply additionally using a 2D image [77].

Voxel grid is a rasterized version of a point cloud. Essentially voxelization decreased dimensions of a point cloud by a magnitude, but still keeps the original dimensions. It makes voxel (which represents many points) the smallest unit of point cloud. Voxels are used to reduce the computational complexity of point cloud algorithms. Efficient use of a voxel grid requires compression of all points inside a voxel, i.e., extracting or learning voxel features. A simplest example is to find a centroid of a voxel by taking the mean of all points that belong to it. Such approach, e.g., [22, 132], uses at least partly hand-crafted features or deterministic methods, not fully learnable deep learning models. Several methods use deterministic algorithm to remove points that are considered less relevant and then process the remainder. For example, Asvadi et al. [6] remove voxels that have points with small variance in height, assuming them to represent ground.

Such an approach has two caveats: voxelization has a computational cost (points in the point cloud are unordered, thus, voxelization requires sorting of the points) and hand-crafted features have limited use. PointNet [124] and Pointnet++ [126] were proposed to overcome the need for voxels and directly learn features from an unordered point cloud. However, Pointnet is trained on relatively small point clouds of about 1000 points, whereas a typical outdoor scene in KITTI [49] dataset has more than 100,000 points. VoxelNet [183] was proposed to overcome this limit while still enabling automatic feature learning.

3.2.3 Loss functions for 3D object detection

The primary aim of a loss function is to provide feedback to the training process and guide it towards a better solution. Loss value in supervised learning is computed by comparing model output to the ground truth. This value is then used to compute partial derivatives for each node in a deep learning network. A partial derivative of a node indicates how much this node contributes to the loss value. The optimisation process then changes node parameters (neural network weights) such that the loss value would decrease for the next step. Most 3D object detection algorithms use cross-entropy loss, Smooth L1 loss [52] or another well-known loss function (or an adaption of one). The advantage of Smooth L1 loss it that it is less sensitive to

outliers than a regular squared error loss (see definition in Eq 3.7).

Lin et al. [90] introduced focal loss for 2D object detection. Focal loss gives more weight to the estimations of more complex objects, i.e., to the objects that are more difficult to learn for the neural network.

α-IoU loss [57] takes a similar approach but instead of balancing classification part it works on bounding box part of the estimation.

Unlike for model architectures where 2D examples are not directly transferable to 3D due to computational limits, most 2D loss functions can be easily adapted to 3D scenario. However, this is a little bit deceiving as those loss functions are often designed for data with different distributions and variances, and hence might not be best suited for 3D. For example, a distant object on a 2D image might be represented as 10x10 pixel patch, having 100 data points while the same object close up might be represented by 100x100 pixel patch having 10,000 data points. The ratio for 3D object might be the same but not due to scale, but due to sparsity. A close object might be represented by 300 points and a far-away object by 3 points, but their dimensions are the same, corresponding to the real world dimensions (e.g., 5m x 2m x 2m). A good model would still localise it but probably would not be able to estimate a bounding box with the same precision due to sparsity. Thus, a good 3D loss function should guide the model to learn the far-away object location but does not have to penalise the model too harshly for not getting the exact object position correct (as this may hinder model's ability to generalise well when it is forced to output a precise localisation when there is not enough data to do so). The drawback of extensive optimisation for finding every possible object is that the resulting model might be overfitting, i.e., it is so specified that its feature space will not generalise well for other tasks. Recent research has shown that loss functions contribute to such [75] overfitting. However, the role of loss function is mostly problematic when a trained model will be later adapted to different tasks (i.e., used for transfer learning where only part of the model is re-trained while keeping most model weights).

3.2.4 Coordinate frame selection

Unlike in the 2D case where all the data are in the same coordinate frame, for object detection from point cloud data it is necessary to transform data from one coordinate frame to another to train and visualise data. KITTI dataset [49] provides ground-truth coordinates in a camera frame while point coordinates in point clouds are in lidar frame. Additionally, visualisation of results is important part of the research process, so it is useful to work such that all data are in camera coordinate frame. Point clouds in a training dataset are transformed using a transformation

matrix T_{velo}^{cam} that is built using the rotation matrix R_{velo}^{cam} and translation vector t_{velo}^{cam} that KITTI provides for each sequence [49]:

$$T_{velo}^{cam} = \begin{pmatrix} R_{velo}^{cam} & t_{velo}^{cam} \\ 0 & 0 \end{pmatrix}. \tag{3.1}$$

3.3 VoxelNet

VoxelNet [183] was the best performing object detection algorithm on KITTI [49] object detection benchmark when this PhD research started in early 2018. Therefore, it is used as a base model for object detection research in this thesis. VoxelNet is not an open source model and was implemented for experiments by the author of this thesis.

The original contribution of VoxelNet is the feature learning network where a feature is learned for each non-empty voxel. Full VoxelNet model is depicted in Figure 3.2. A point cloud is input into feature learning network which learns a feature for each voxel using multi-layer perceptrons (MLP). Voxels are then processed using 3D convolutions with batch normalisation (BN). Finally, 3D space is converted back to 2D and a Region Proposal Network first downsamples and then upsamples the data before it outputs the estimations. The output consists of regression estimate (object coordinates) and boolean classification (object type) for each voxel. Data processing steps are presented in yellow colour while neural network architecture is in green in Figure 3.2.

Conceptually, VoxelNet in its default configuration divides a point cloud into voxels. Each voxel corresponds to 40x20x20 cm in the real scene. Up to 35 points are sampled for each voxel and a PointNet-like feature learning neural network is applied to each voxel independently. Independent feature learing allows to learn features sparsely, skipping the learning process for empty voxels. After learning a feature for each voxel, a dense voxel grid is reconstructed where also empty voxels are now represented. This grid is input to a dense 3D convolutional layer enabling the spatial relations between voxels to be captured. Then, a Region Proposal Network described in Section 3.2.1 is used as the last layer in the model for making estimations. Learned features from 3D convolutional layer are flattened along depth dimensions resulting in a 2D space (birds-eye-view). Finally, an estimation is made for each 40x40 cm area of the resulting 2D space. The model outputs binary classification (i.e., whether an area contains an object) along with the object dimensions.

VoxelNet can be trained to recognise different types of objects. Ground truth data for objects in the training dataset must be in a specific format. Object dimen-

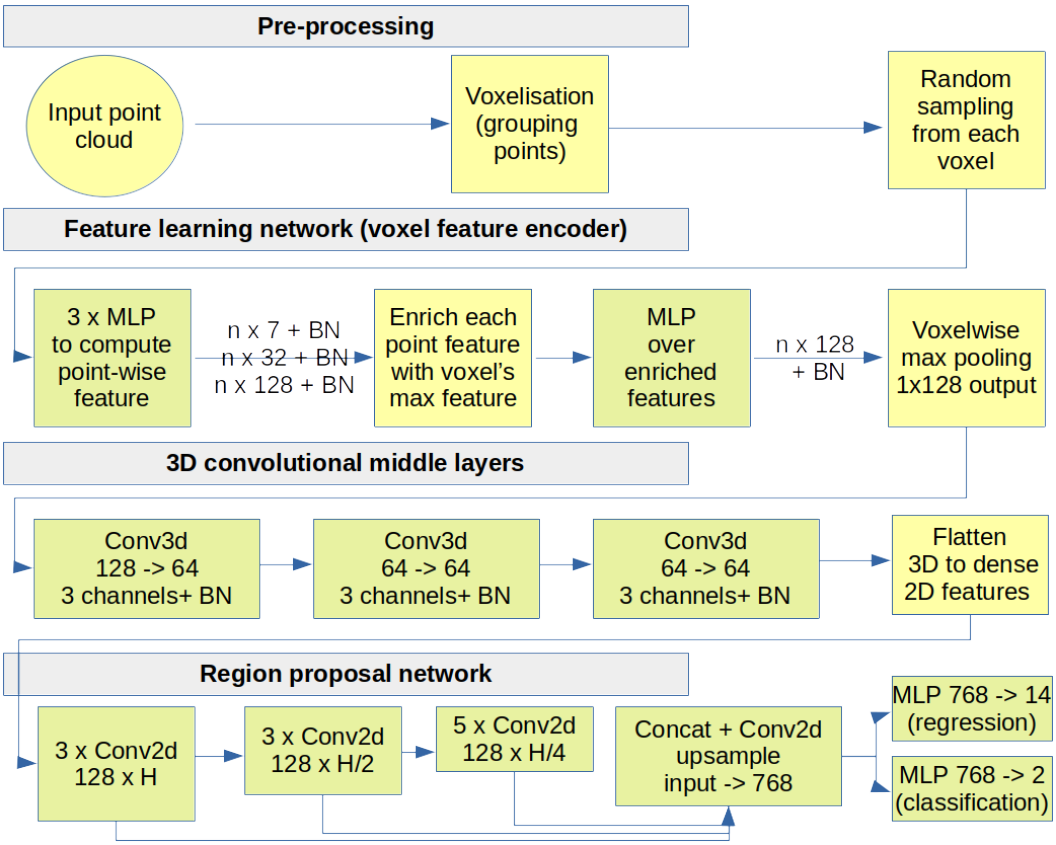


Figure 3.2: VoxelNet model architecture. The model was originally presented in [183]. The figure should be followed from the top to the bottom.

sions are represented as vector consisting of 7 components: x, y and z coordinates of the object center, normalised object bounding box dimensions and the yaw rotation around the height axis. Object bounding box dimensions are given relative to the Region Proposal Network's anchor point a. Assuming central coordinate of a ground truth object is represented by (x_g, y_g, z_g) and a coordinate of a corresponding anchor point is represented by (x_a, y_a, z_a) then relative coordinates are computed as

$$\Delta x = \frac{x_g - x_a}{d}, \Delta y = \frac{y_g - y_a}{d}, \Delta z = \frac{z_g - z_a}{h_a} \tag{3.2}$$

where $d = \sqrt{l_a^2 + w_a^2}$ is a diagonal of the base of the anchor box and l_a, w_a and h_a are respectively the length, width and height of the anchor box. Additionally, bounding box dimensions and yaw angle θ of the object are also represented relative to the anchor box. Also, bounding box dimensions Δl, Δw and Δh are additionally

normalised, such that

$$\Delta l = \log(\frac{l_g}{l_a}), \Delta w = \log(\frac{w_g}{w_a}), \Delta h = \log(\frac{h_g}{h_a}), \Delta\theta = \theta_g - \theta_a \tag{3.3}$$

where l_g, w_g and h_g are length, width and height of a ground object and l_a, w_a and h_a are again the length, width and height of an anchor box. θ uses the same subscript notation. Consequently, VoxelNet ground truth format for each object is a vector $u = [\Delta x, \Delta y, \Delta z, \Delta l, \Delta w, \Delta h, \Delta\theta]$ and regression component of a Region Proposal Network also outputs a 7-component vector $u\prime = [\Delta x\prime, \Delta y\prime, \Delta z\prime, \Delta l\prime, \Delta w\prime, \Delta h\prime, \Delta\theta\prime]$ for each anchor box. Because there are two perpendicular anchor boxes for each anchor point, the regression output is a 1×14 vector for each anchor point, but each bounding box prediction is evaluated independently. Additionally, a model outputs a Softmax classification probability score in a range between 0 and 1 for each anchor box.

3.3.1 Loss function of VoxelNet

VoxelNet [183] uses a combination of a balanced binary cross entropy loss for a classification and Smooth L1 Loss [52] for a regression component. The two components are summed to calculate a final loss value. Classification component is further composed of positive anchor loss and negative anchor loss which represent loss values for anchor boxes that overlap with a ground truth object and do not overlap with ground truth boxes, respectively. The model outputs two vectors, regression estimate $u\prime$ and classification estimate p. Classification estimation vector p is separated into p^{pos} representing estimated probabilities for the positive ground truth anchor boxes, and p^{neg} representing estimated probabilities for the negative ground truth anchor boxes. The loss function that outputs loss value L is then formalised as

$$L = \alpha\frac{1}{N_{pos}}\sum_i L_{cls}(p_i^{pos}, 1) + \beta\frac{1}{N_{neg}}\sum_j L_{cls}(p_j^{neg}, 0) + \frac{1}{N_{pos}}\sum_i L_{reg}(u_i, u_i') \tag{3.4}$$

where α and β are balancing factors for positive classification component and negative classification component respectively, L_{cls} is binary cross entropy loss, N_{pos} and N_{neg} are the number of positive and negative anchor boxes in a corresponding ground truth data. Regression component L_{reg} is Smooth L1 Loss and calculates loss value for positive anchor boxes only.

The balancing factors α and β are used to balance the importance of positive and negative anchor loss. The authors of VoxelNet [183] use $\alpha = 1.5$ and $\beta = 1$.

3.3.2 Inherent limits of VoxelNet

The work in this PhD research uses VoxelNet as baseline model and offers several improvements. As VoxelNet does not have a reference implementation, the model is implemented from scratch. For some parts it is possible to use existing but incomplete third-party implementations. There are many good reasons to select VoxelNet as a baseline model: it is fast, well known, it has a simple and extendable architecture, and it has proved to work well. However, the main reason for selecting VoxelNet as a baseline is that at the time when this PhD research on object detection started in early 2018, VoxelNet was the state-of-the-art model for 3D object detection. At the time it topped the KITTI [49] 3D object detection benchmark.

We observe that the VoxelNet model also has several deficiencies. First, with standard implementation as described in [183], it was not possible to repeat the results claimed in the paper. To author's best knowledge, neither has any other third-party implementation succeeded in repeating them. However, the results have been verified by the official KITTI [49] benchmark, so they must be achievable. Most probably, some minor configuration parameters or implementation details were omitted from the paper. However, with some parameter tuning (especially α and β balancing parameters of the loss function), the replicated results resemble closely the original model performance. Second, while VoxelNet utilises inherent spatial relations in 3D data (e.g., uses convolutions over the 3D voxel grid), it uses 2D data for generating anchors, an important part of the training process. This restrains it from using the full potential of its capabilities as the height parameter is not used for anchors. Third, it treats all objects in a training dataset as equal, limiting its ability to learn complex objects that are either partly occluded or far away. This work aims to address these issues.

3.4 Adaptive multi-component loss for 3D object detection

This section introduces adaptive multi-component loss for 3D object detection which is the main contribution of this chapter. The method adapts the weights it attributes to each loss component as the training progresses, hence we call it adaptive multi-component loss. Our reference model, VoxelNet, uses the combination of cross-entropy loss (for object localisation) and Smooth L1 loss [52] (for object boundary detection). The cross-entropy loss function is parameterised by weight factors $\alpha = 1.5$ and $\beta = 1$ for the relative importance of locations that contain an object and

locations that do not, respectively.

We replace cross-entropy loss for object localisation by adopting focal loss [90], and hence introduce additional parameter γ which is a modulating factor to add weight to those objects that the network has not located well. The basic formula for a cross-entropy loss of single object used by VoxelNet is

$$L = -\log(p), \tag{3.5}$$

where L is the output loss value and p is a estimated probability for a single location containing a object. When a location contains an object then p should ideally be approaching 1 which would minimise the loss value L. Adapting the focal loss we change this to

$$L = -(1-p)^{\gamma}\log(p). \tag{3.6}$$

We use the our convergence assessment metric with Optuna [3] to find the best parameters. At the end we follow [90] and choose $\gamma = 2.0$ for training as this gave the best results. However, we set $\alpha = 0.25$ and $\beta = 0.75$ which give better results than values used in VoxelNet [183] (they use $\alpha = 1.5$ and $\beta = 1.0$). Finally, we include Smooth L1 loss [52] for bounding box regression for all three dimensions and for object rotation. A loss value L is computed for each of them independently. The benefit of Smooth L1 loss is that it is less sensitive to examples with outliers as it uses squared loss term for loss values less than 1 and absolute term otherwise. For an element-wise error e, a Smooth L1 loss is computed as:

$$L = \begin{cases} 0.5e^2, & \text{if } e < 1, \\ e - 0.5, & \text{otherwise.} \end{cases} \tag{3.7}$$

Given that each bounding box has 4 estimated values (length l, width w, height h, and rotation θ) and location (x, y, z coordinates), a loss value L is computed for each of them and seven values are then summed. Bounding box regression loss is only computed for positive anchor boxes that actually contain an object. Similarly to VoxelNet [183] we do not estimate the parameters directly, but differences from the anchor box. Estimations are also normalised. Given that each box and bounding box is described with a vector (l, w, h, x, y, z, θ), each estimation and ground truth vector is normalised. Diagonal $d^a = \sqrt{(l^a)^2 + (w^a)^2}$ is used to normalise length and width coordinates (as KITTI camera frame is used then y is the height coordinate). Superscript ^{a} marks an anchor box parameter and ^{g} marks a ground

truth parameter. The estimated vector δ is as follows:

$$\delta = (\frac{x^g - x^a}{d^a}, \frac{y^g - y^a}{h^a}, \frac{z^g - z^a}{d^a}, \log(\frac{l^g}{l^a}), \log(\frac{w^g}{w^a}), \log(\frac{h^g}{h^a}), (\theta^g - \theta^a)). \tag{3.8}$$

Therefore the bounding box regression L_{box} loss for a positive anchor box is

$$L_{box} = \sum_{param \in \delta} L(param) \tag{3.9}$$

where $param$ is a component of δ. Such loss formulation achieves better results than the original loss setup for VoxelNet. However, the gain is not significant. The most important contribution of this thesis is that we make the loss components dependent on each other and use the value of the box localisation probability as a dynamic modulating factor for bounding box regression loss such that

$$L_{box} = \left(\sum_{param \in \delta} L(param) \right) * p \tag{3.10}$$

where p is the output probability of a given anchor box. The motivation of using probability p from classification output is to allow the regression component loss L_{box} to focus on objects that are already classified and not penalise the model on dimensions of such bounding boxes that the model is not able to locate yet. Also, there is no need to deal with class imbalance here. As class imbalance only concerns the classification component for the loss function, it does not concern the regression component which only penalises bounding box dimensions for a positive class.

We also test several more complex setups with p and an additional modulating factor γ, similarly to the focal loss:

$$L_{box} = \left(\sum_{param \in \delta} L(param) \right) * (p)^{\gamma} \tag{3.11}$$

and

$$L_{box} = \left(\sum_{param \in \delta} L(param) \right)^{p}, \tag{3.12}$$

but find no configuration which would yield better results than the simple formulation in Equation 3.10.

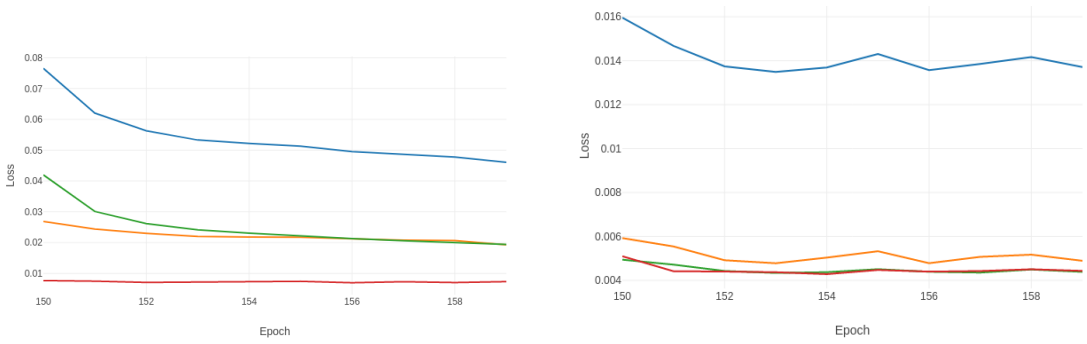
3.4.1 Detaching modulating factor from a computational graph

While the mathematical definition of our adaptive multi-component loss is rather straightforward, it is still very effective. However, the most important part of the novel loss function is a correct implementation. As mentioned in Section 3.2.3 loss values are used to compute partial derivatives for each node. Those indicate how much each node contributes to the total loss. However, if we directly use the localisation probability p in computing L_{box}, then L_{box} also additionally contributes to the loss value that is attributed to those nodes in the deep learning model that are responsible for localisation. Our experiments show that this is not a desirable side-effect as it amplifies the role of localisation and will diminish the loss attributable to the bounding box regression. As a result, the accuracy of bounding box regression falls. Also, the localisation itself is already well-balanced using a version of focal loss in Equation 3.6 so there is no need to change that. Therefore, the crucial step is to detach p from a computational graph of the deep learning framework. PyTorch [118] framework is used in this thesis to write the models and train them. The role of the framework is to provide auto-differentiation, i.e., there is no need to explicitly compute the partial derivatives as PyTorch handles it. PyTorch observes the computations in the model and automates the computation of partial derivatives. It is essential to detach probabilities p from the computational graph before using the values in computing L_{box}. The effect of detaching p from a computational graph is to enable the computation of L_{box} to take advantage of information about localisation progress without affecting it. As a result, positive anchor points that are well localised, contribute more to bounding box regression loss L_{box} than those that are not yet localised (i.e., model has attributed a low probability of them containing an object). Such setup ensures that localisation is considered first and when the boxes are well localised then more emphasis is put into estimating their parameters.

The rationale behind the adaptive multi-component loss is easy and difficult to grasp at the same time. From a human perspective it is very simple: one has to localise the object before assessing its parameters. It is more complex from a computational perspective. Localisation and bounding box regression are parallel tasks. Therefore, the model does not localise the object first. However, localisation and bounding box regression in Region Proposal Network both use the same latent feature space that the convolutional middle layers of VoxelNet output. Commonly used loss functions (such as cross-entropy loss, Smooth L1 loss) force the feature space in Feature Learning network and in convolutional middle layers to adapt to both tasks at the same time. Although if the feature is not sufficient to be used for localisation, it most probably is not sufficient for bounding box regression

either. The complexity of the tasks is significantly different. As a result the relative importance of a localisation penalty decreases if box regression penalty is also high.

Such situation might lead to complications. For example, it is a common problem that the point cloud is too sparse to estimate the bounding box parameters correctly, but sufficiently dense for localisation. With a standard loss function, the problem is not solvable. Even if the localisation succeeds, then the model fails to find good bounding box parameters and starts to rearrange the feature space. This might lead to a situation where the model is unable to localise the object using a new feature space. It then uses localisation loss to arrange the feature space back. Then the cycle repeats. Observations of loss value fluctuations confirm this assumption. Figure 3.3 displays how loss values decrease for adaptive multi-component loss in (a) and for cross-entropy loss combined with Smooth L1 loss in (b). Total loss value is represented in blue color, while individual component of the loss are below it. Positive localisation loss, negative localisation loss and regression loss are orange, green and red respectively. As can be observed, there are more fluctuations in (b) meaning that the optimisation process is rather coarse. Attempts to localise new objects will result in losing some existing progress. For both loss functions, exactly the same VoxelNet implementation with exactly the same setup is used. Both use stochastic gradient descent (SGD) optimiser. The learning rate for the SGD optimiser is set to 0.001 for both loss functions.



(a) (b)

Figure 3.3: Comparison of loss value smoothness for cross-entropy based loss and adaptive multi-component loss. Adaptive loss in (a) has a much smoother curve than the combination of cross-entropy and Smooth L1 loss in (b). Total loss is displayed in blue, positive localisation loss in orange, negative localisation loss in green and regression loss in red colour. Both (a) and (b) use SGD optimiser with 0.001 learning rate.

Most probably the novel loss function works well because it forces the model

to learn the easy task first. Later in the training when model performs better, the loss function adapts to the progress. Due to changing weights of the components, i.e., due to adaptive behaviour, same features can be refined to solve the more difficult task of bounding box regression without losing progress that was made on localisation.

3.4.2 Results

As mentioned in Section 3.4 we determine the best model configuration for our VoxelNet implementation and for both loss functions by using hyperparameter search using Optuna [3] and our convergence assessment metric. The effect of adaptive multi-component loss is measured by training two identical VoxelNet models. One is using the adaptive multi-component loss and the other is using the setup in VoxelNet paper [183]. The model setup is kept fixed for two loss functions. We use all four forms of data augmentation from Section 3.6.3, SGD optimiser with a learning rate 0.01 for the first 150 epochs. The learning rate is decreased to 0.001 for the last 20 epochs. The results are then evaluated on the KITTI [49] validation set using official KITTI validation methods. Loss parameters are $\alpha = 0.75$, $\beta = 0.25$ and $\gamma = 2$. γ is only used for adaptive multi-component loss.

The difficulty levels are defined on KITTI 3D object detection benchmark website[1] and shown in Table 3.1. The dimensions of an object are given in pixels (px) and corresponds to the 2D size of a 3D object if it was projected into a corresponding 2D image.

Difficulty level	Min. bounding box height	Max. occlusion	Max. truncation
Easy	40 px	Fully visible	15%
Medium	25 px	Partly occluded	30%
Hard	25 px	Difficult to see	50%

Table 3.1: KITTI 3D object detection benchmark difficulty levels as defined on their website.

Table 3.2 compares the average precision (AP) [38] of the combination of cross-entropy loss and Smooth L1 loss to the proposed adaptive multi-component loss. The equation to compute AP [38] is

$$AP = \frac{1}{11} \sum_{r \in 0, 0.1, \ldots, 1} \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}). \tag{3.13}$$

[1] `http://www.cvlibs.net/datasets/kitti/eval_object.php`

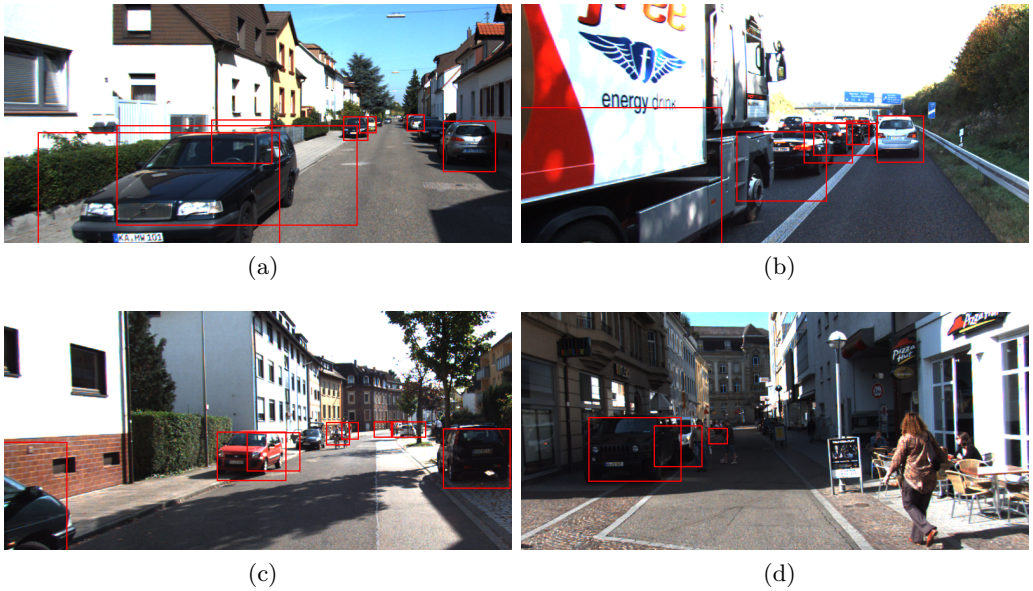
In Equation 3.13, the results of object detection are separated into 11 equally spaced buckets r between 0 and 1. At each bucket, a maximum precision $p(\tilde{r})$ is found among all measurements where the recall exceeds threshold r. Precision values are then averaged. The results are validated on KITTI [49] validation set using 'Car' category. As is evident from Table 3.2, the novel loss function outperforms its alternative by a large margin at each difficulty level. An interesting observation is that both setups achieve higher AP on more difficult objects. This is attributable to the full volumetric comparison of estimations and the ground truth where the model is performing poorly on very close objects. Figure 3.4 (a) and (b) have examples of such close objects. While the object is correctly detected, its dimensions are falsely estimated.

KITTI difficulty level	CE + Smooth L1 loss	Adaptive multi-component loss
Easy	13.02	27.83
Medium	14.60	27.55
Hard	19.17	31.01

Table 3.2: Average precision (AP) results (in %) of training VoxelNet on KITTI validation set in 'Car' category with the novel adaptive multi-component loss and with the combination of cross-entropy loss and Smooth L1 loss.

The results presented here are significantly lower than those in [183]. This is due to a shorter model training period due to high cost of training. However, clearly the model is converging and estimating well. The relatively low AP score comes from a high number of false positive estimations (e.g., a model estimates a car where there is none), imprecise bounding box estimations for close objects, and also from missing very sparse objects. While this should be improved for better accuracy in real traffic conditions, it does not interfere with the comparison of loss functions as the nature of false positives is similar and caused by the model architecture and training level, not the loss function. Model can be forced to reduce the number of false positives. Most probably tuning the balance of α and β further would overcome this problem. β is used to penalise the estimation of false positives. Further tuning was not done for this research as 1) there were not enough funds to pay for the computing time 2) the models are well comparable as the nature of false negatives is similar 3) the qualitative assessment of results confirm the model is usable. Figure 3.4 illustrates several typical error cases. As can be observed, false positives are mostly placed very close to the real objects (in (a), (b) and (d)) or are far away from the sensor (in (c)). Hence for practical navigation and collision avoidance they do not pose a significant problem. Future research should address this problem to eliminate false

negatives.



(a) (b)

(c) (d)

Figure 3.4: Examples of false positive estimations of VoxelNet model trained on KITTI dataset [49] with adaptive multi-component loss. Estimated bounding boxes are presented in red colour. As can be observed, false estimations are either placed close to the real objects (in (a), (b) and (d)) or far away (in (c)).

Figure 3.4 also demonstrates that all important objects are found. All cars that might cause a collision are detected. The errors are not critical. The truck in (b) is detected smaller than it is, but facet closest to the sensor is well detected. In (a), the model confuses a pair of pedestrians with a car. Chapter 6 further analyses the effect of such errors and ways to alleviate those from collision avoidance perspective. However, it is clear from Table 3.2 that the novel adaptive multi-component loss function performs much better than the original loss function in VoxelNet, having much fewer false positives and false negatives.

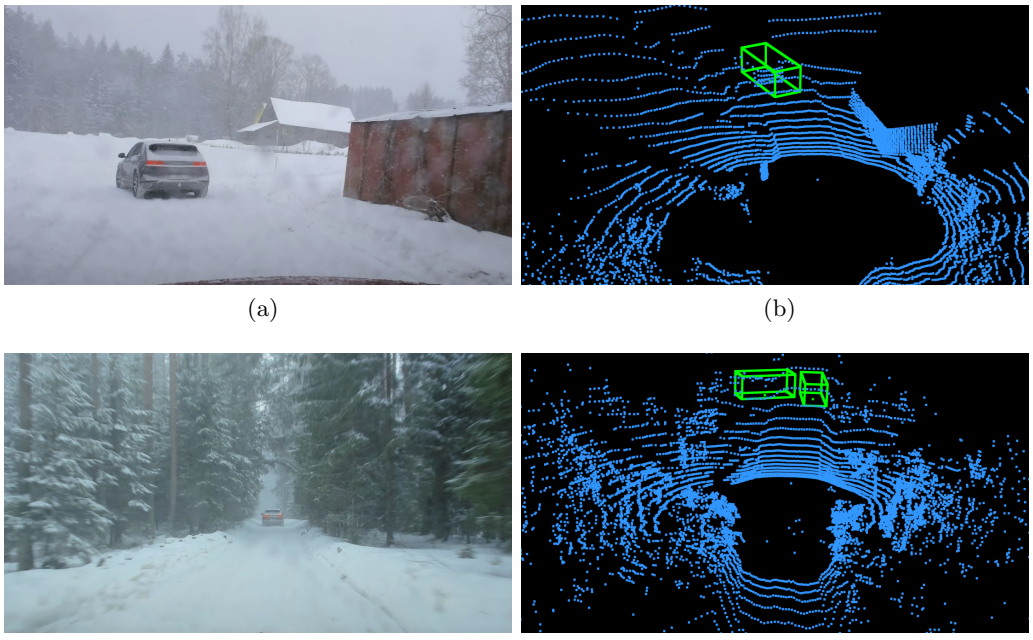
3.4.3 Qualitative evaluation on rural winter data

Finally, we collect data in a rural environment and use it to evaluate the model trained with adaptive multi-component loss qualitatively. The reason for using only qualitative assessment and not quantitative assessment is the lack of rigorous 3D ground truth data which would be too costly to create for a small experiment. Dataset consist of 2 hours of recorded data (5 different continuous recordings).

The data is collected using Ouster OS-1 lidar [116] in Southern Estonia near Vellavere and Vapramäe. The collection took place during the winter. In addition

to regular winter scenes we also chose scenes with rather heavy snowfall into the test to see how the model performs in a noisy environment. Some of the laser rays reflect back from the snowflakes and create a point is an otherwise empty space. To our surprise where were no false positive detections or other disturbances which we could address to the noisy points reflecting snowflakes (some point clouds contained hundreds of such points).

The performance of a model is very good considering that the model itself was trained on KITTI dataset [49] which contains mostly city scenes and no winter scenes. It excels at spotting sparsely represented cars in an otherwise unstructrured environment as shown in Figure 3.5. However, due to sparsity it is not able to position more distant objects well and tends to make several different proposals for the same object (Figure 3.5 (d)).



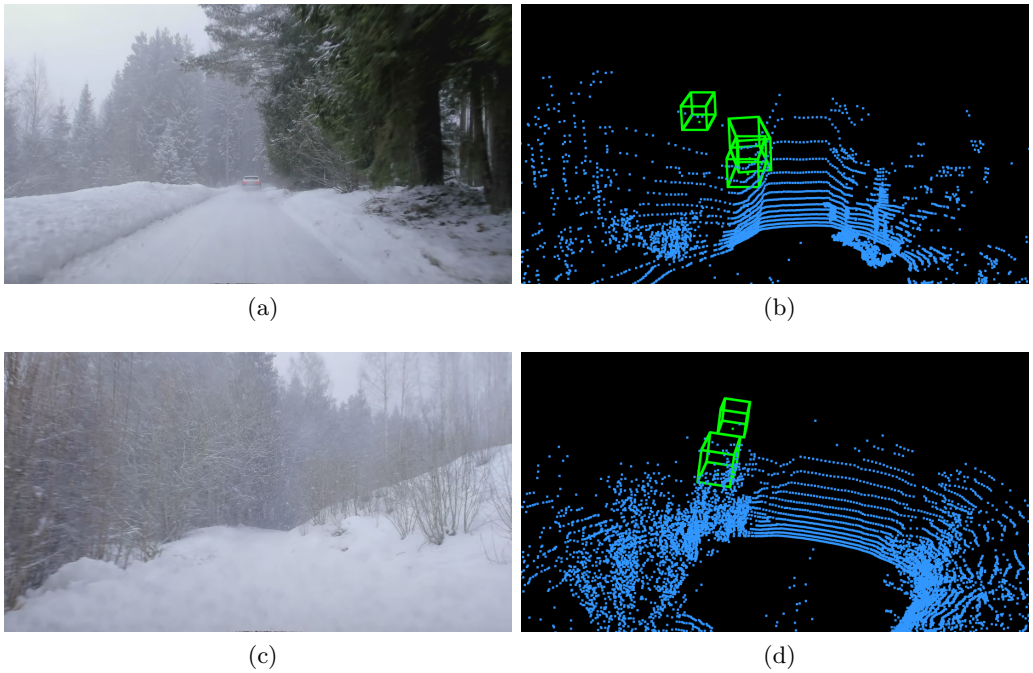
(a) (b)

(c) (d)

Figure 3.5: Examples of object detection on rural winter data using VoxelNet model trained with adaptive multi-component loss on KITTI dataset [49]. Photos of a scene are on the left and detection on 3D point clouds on the right. As can be observed, the model works well in rural winter conditions (in (b) and (d)) although some objects are often represented with several bounding boxes (in (d)).

However, the model also makes false positive predictions. In Figure 3.6 two typical situations are covered. First, in Figure 3.6 (b) the model confuses a large snow wall to a car and in (d) it falsely detects car in bushes. There were several

other similar occasions where snow walls or trees and bushes were detected as cars. In Figure 3.6 (a) and (b) there is also a real car which it not detected. This it not due to the model fault, however. As can be observed in (b) there are no recorded points where the car should be. This is because the weather was relatively mild (-2 degrees Celsius) and the falling snow was a bit wet. As a result it fell onto the lidar and started to freeze there. The points are not recorded because the front upper part of the lidar surface was already covered with ice.



(a) (b)

(c) (d)

Figure 3.6: Examples of failed object detection on rural winter data using VoxelNet model trained with adaptive multi-component loss on KITTI dataset [49]. Photos of a scene are on the left and detection on 3D point clouds on the right. As can be observed, the model treats snow walls as cars (in (b)) and confuses bushes to a car (in (d)).

A VoxelNet model trained with adaptive multi-component loss on city summer scenes is surprisingly robust when used with very different rural winter data. It is reasonable to infer that the model is able to learn characteristic features of an object type irrespective of the context. Further, robustness to the noise is also remarkable as there were no false detections due to random points caused by large and heavy snowflakes in an otherwise empty space.

3.5 Method to assess convergence of a model for faster pruning

Loss value is a good indicator of model convergence in a typical hyperparameter tuning scenario. However, if hyperparameter search space includes loss function parameters, loss values render incomparable and other methods are needed. This section offers an alternative method that is not using a loss value. It is based on assessing and clustering model output probabilities.

Training deep learning models requires many experiments which are costly. Engineering a well-performing model requires a lot of hyperparameter tuning where different configurations of model parameters are tested and evaluated. In some cases, it also requires changing other setup and configurations like augmentation methods, loss components or data format, to name a few. Configurations that might work for some dataset, model setup or even for an object type might not be optimal for another [9]. As training 3D models takes a lot of time (days or even weeks depending on the available resources) and data, it is important to observe the quality of a model setup during the training such that bad configurations that do not converge could be pruned early. It is especially important if an automated hyperparameter search is conducted (e.g., using Optuna [3], Tune [89] or other methods). This requires observing some selected metrics related to the model. The most obvious indicator of goodness is usually a direct measurement of model goodness on the validation data. However, a loss value output by the loss function and averaged over all the training examples at the end of each training epoch (i.e., after a single pass over the data) can also be used as an indirect indicator of the convergence of the model. The model converges if the loss value decreases. If the loss value does not decrease or fluctuates then it is likely the indicator of non-performing model configuration. When using an adaptive loss function it changes its parameters, such as a modulating factor or balancing factors, while training, it is mostly not possible to compare loss values directly between runs and epochs. This is especially so when a training process utilizes a non-deterministic routine (e.g., data sampling, augmentation, etc). On the other hand, to save time and cut research costs, there is a need to assess and compare model quality for very different training configurations such that bad configurations could be pruned (i.e, stopped) early to save time and resources.

Figure 3.7 illustrates a typical case where different loss functions output different loss values for the same model state. All three loss functions receive the output values of the same 3D object detection model at each epoch. Each loss

function then outputs its loss value by comparing model output to the ground truth. As can be observed from the graph, when the model output changes on each epoch, loss functions tend to change towards the same direction. This can be explained as when the output is closer to the ground-truth value, the loss value decreases and vice versa. However, the precise amount, how much the loss value changes, varies by loss function as they penalize errors differently. Therefore, as can be derived from the experiment in Figure 3.7, it is not possible to compare training progress by comparing loss values from different experiments if different loss functions or loss function parameters are used for training. This is a contradictory observation to a typical hyperparameter search where different model configurations are usually trained using the same loss function and where the average loss value would be a good indicator of model convergence.

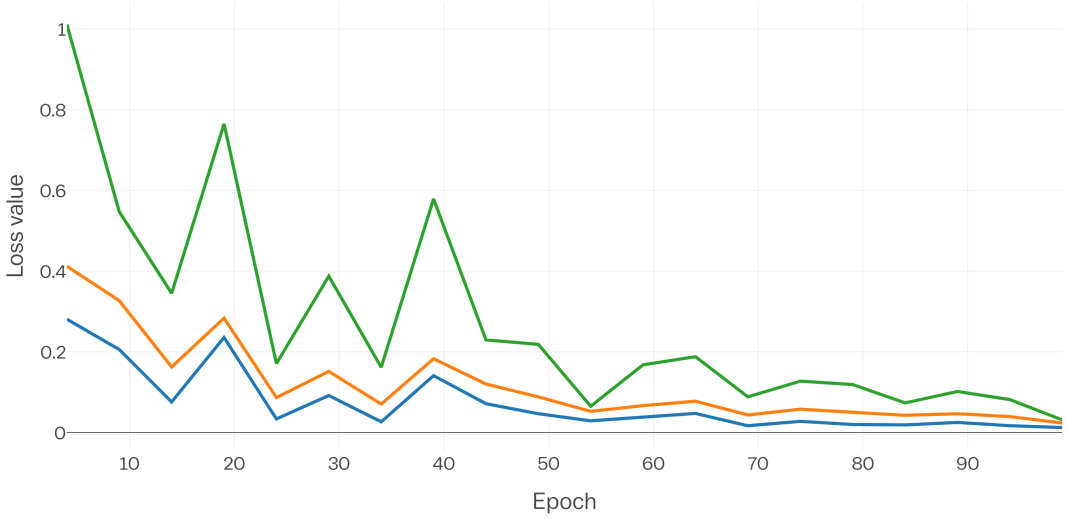


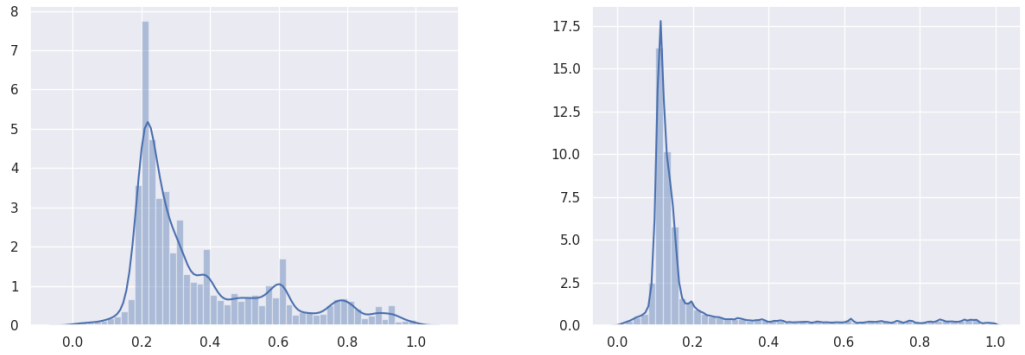
Figure 3.7: Loss value is an indicator of the convergence of the model. If several model configurations are evaluated, then loss value is comparable if the same loss function is used. However, as the figure illustrates, the reverse is not true. The same model is evaluated with 3 differently parameterised loss functions. Model quality is the same, but loss function values are not comparable for pruning purposes. Hence other methods are needed for evaluation if hyperparameter search includes loss function parameters.

While the goodness of a model could be evaluated by validating the estimation against the ground truth, this is feasible only for simple tasks, but computationally costly for complex cases. For example, a precise validation of 3D object detection result includes computing volumetric overlap between the ground truth box and the estimated box which are both rotated along the yaw axis. Such validation either requires additional computational resources or on shared resources

increases training time. To avoid such extra cost while still being able to monitor a rough estimate of the progress in the training, it is necessary to use a metric that is cheap to compute. This work proposes a simple metric based on the probability distribution of a binary classification output. This work uses it to prune ineffective configurations while training 3D object detection models.

3.5.1 Assessing convergence from a probability distribution

A classification layer at the end of a deep learning model normally outputs a probability of an item belonging to a given class (with a value between 0 and 1), likewise for the model introduced in this chapter. A converging high-quality model outputs high probability for a correct class (e.g., class "car" for any real car) and low probability otherwise. On the other hand, a model that is either in an early phase of learning or not able to learn well will have a much more uniform frequency distribution. Example is shown in Figure 3.8 where the same point cloud from KITTI [49] dataset is used to output model estimations at epoch 20 and at epoch 50. Compared to a converging model output distribution in Figure 3.9, the distribution here is much more uniform. The model that outputted the results was later trained until 10 000 epochs to make sure that it will not converge to a good solution (it probably was stuck in a local optima). This section uses this observation to introduce a



(a) Epoch 20

(b) Epoch 50

Figure 3.8: An example of a probability distribution of a poorly converging model. An example frame from KITTI [49] dataset. Model output at epoch 20 is displayed in (a), and model output at epoch 50 in (b).

metric that allows to compare different hyperparameter and model configurations without using loss value, and without having to spend computational resources on

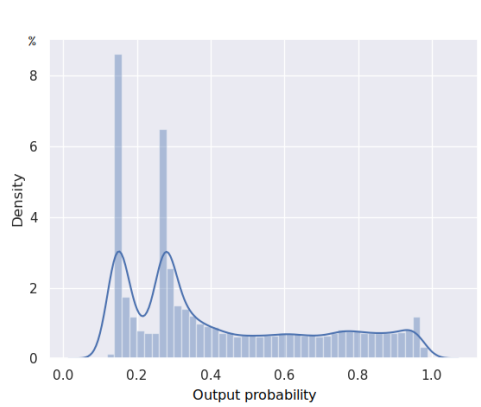
exact evaluation of the model. Using frequency distribution of output probabilities enables to measure the convergence in a stable manner across different model configurations. However, it should be noted that the method is mainly tested on KITTI dataset, and hence the observation made may be ad hoc and not generalise well.

3.5.2 Empirical evaluation

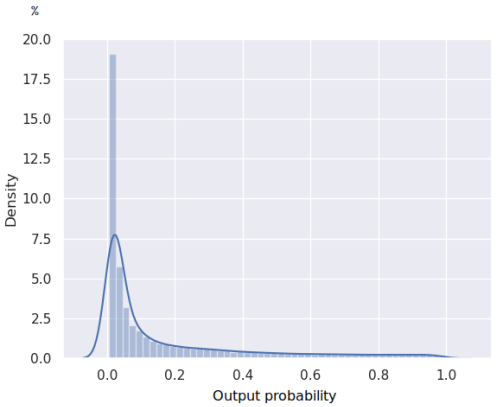
The approach is based on an observation that it is not necessarily a good idea to prune configurations that converge slowly according to a loss value, but to prune configurations that are not decisive, i.e., the output probabilities are too far from 0 and 1. Deep learning models are not convex functions but have many local optima, hence measuring the speed and magnitude of convergence by loss value might also be deceiving and counterproductive. Speed of convergence is dependent on which parts of the optimisation problem are solved first, e.g., training process that tackles complex problems first might converge slower. On the other hand, deep learning models need to discriminate between classes. A binary classifier needs to clearly output negative and positive cases to be practically useful. When a loss value of a 3D object detection model is decreasing during a training process, but there is a high dispersion of output probability values then such model is most probably not optimising well.

Figure 3.9 shows how probability densities change when a model is successfully converging based on our empirical observation. There are over 76,000 anchor points for each point cloud (see Figure 3.1 for the illustration of anchor points). To reduce the computational complexity we discretise output probability values into buckets of 0.01. Each plot represents the distribution of output probabilities for anchor points of a single point cloud. If the probability is close to 0 then the model estimates that there is no object at that location. Accordingly, probability close to 1 means that the model estimates that there is an object. At the first epoch in (a), except for two peaks, the distribution is rather uniform. At epoch 10, most anchor points have probability less than 0.5 and at epoch 20, most are below 0.2. The model is converging. Finally, at epoch 50 the vast majority of anchor points have been ruled out by the model.

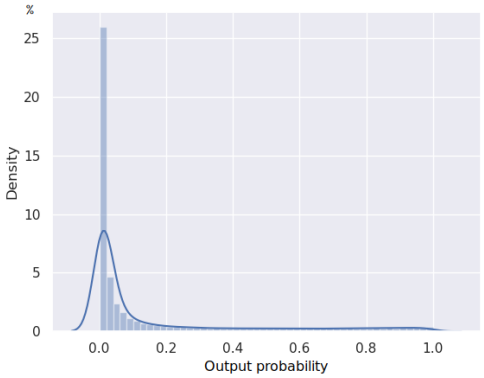
We make use of this observation and notice that instead of plotting the distribution of densities we could as well plot them cumulatively like for cumulative distribution function (CDF). It allows to observe cumulative density D_c for any



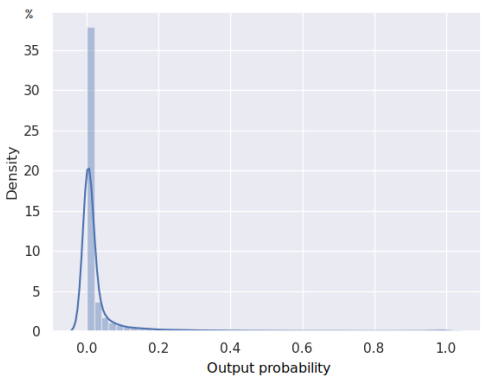
(a) Model makes random bets at epoch 1.



(b) Model starts to converge at epoch 10.



(c) Model converging at epoch 20.



(d) Model at epoch 50.

Figure 3.9: Distribution of output probabilities of anchor points. Scene 000010 from KITTI dataset. The horizontal x-axis represents a discretised probability between 0 and 1 and the vertical axis the percent of probability density falling in any given probability range.

discrete value x in all possible outputs X from a closed interval [0, 1] such that

$$D_c(x) = \sum_{i=0}^{x} D(i), \tag{3.14}$$

where $D(i)$ is density value at i, (i.e., number of outputs at discrete value i). While all possible values of CDF sum up to 1, $D_c(1)$ sums to N, the total number of examples. Further, to make models with different N comparable, $D(i)$ can be normalised

such that its normalised version is

$$D_{norm}(i) = \frac{D(i)}{N} * 100, \tag{3.15}$$

and hence using $D_{norm}(i)$ instead of $D(i)$ we can treat D_c conveniently as CDF such that $D_c(1) = 1$, i.e.,

$$D_c(x) = \sum_{i=0}^{x} D_{norm}(i). \tag{3.16}$$

This normalisation is similar to how data are normalised for calculating a shape parameter λ (count of all observations divided by the number of buckets, i.e., observation space) for Poisson distribution. Unlike in Poisson, the number of buckets here is always fixed as the scale is fixed. Additionally, for object detection, the number of observations is also fixed because of the fixed network output size. This does not allow using algebraic form of Poisson probability function $P(x, \lambda)$ to differentiate the models, hence the use of CDF here.

Our goal here is to find a metric that describes both the variance of the distribution and its centre. CDF enables it indirectly if we set an arbitrary threshold and observe a cumulative value at that threshold.

As mentioned above, the dimension of object detection output is fixed by the network output size. For such models, the majority of estimations are negative. Hence, it can be assumed that the majority of estimations must have a low output probability value. Defining this majority is the configuration parameter α of our metric, where we observe that 0.97 works well. As it is more convenient to use decreasing metric, the final metric γ is the inverse of cumulative distribution at α, i.e.,

$$\gamma = 1 - D_c(\alpha) = 1 - \sum_{i=0}^{\alpha} D_{norm}(i). \tag{3.17}$$

It must be noted that even well decreasing scenario does not guarantee convergence of the model towards global optima. However, this is not of concern here as we are observing this metric across many epochs. It is the role of actual loss function to provide sufficient feedback to avoid or overcome local optima.

3.5.3 Results and Discussion

The implemented metric is stable and can be used to compare training processes of models with different loss functions. Figure 3.10 illustrates how the metric remains relatively stable despite significant fluctuation of the loss function values. While

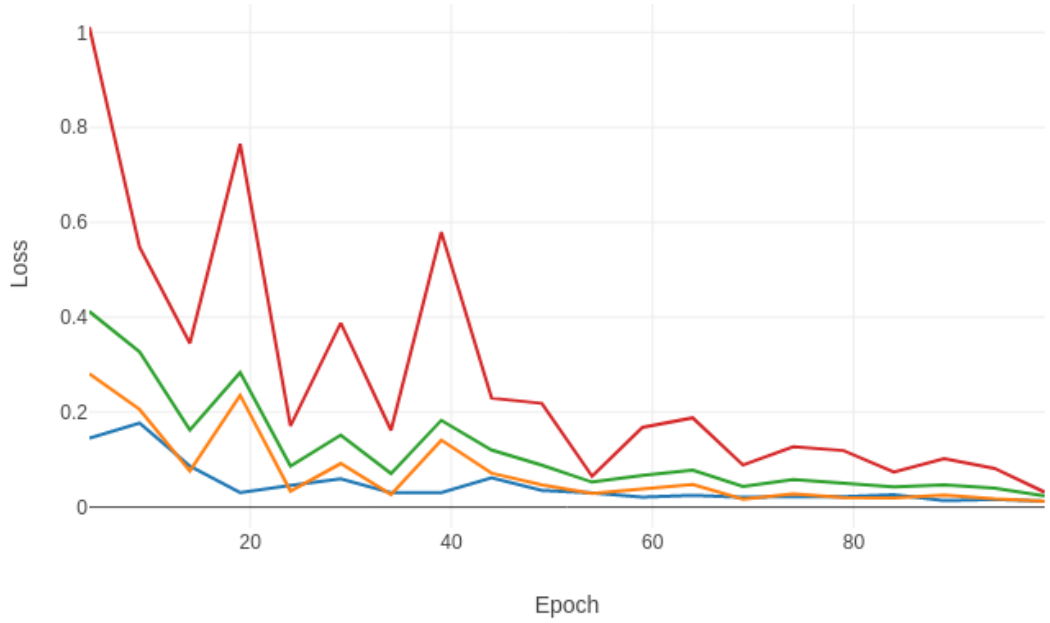


Figure 3.10: The same, already trained model is evaluated with 3 different loss functions (our Adaptive multi-component loss in red, Focal loss [90] in green and Smooth L1 loss [52] in orange) and with the novel convergence assessment metric (displayed in blue). Y-axis value for the convergence assessment metric is the value of γ. Please note that these are not training curves where loss value on each epoch affects the next epoch, but evaluation with model weights fixed for all four experiments.

the loss function values are fluctuating due to many components, the metric value shows how the distribution of estimated values change. Hence, it is not as sensitive to changes in short-term model behaviour, e.g., when a training process might try to overcome local optima. This is also because 3D object detection involves both object localisation and object boundary detection. The probability distribution metric only looks at localisation as this is crucial on early stages of training, where evaluating boundary detection only makes sense when the object has been well localised. As the main application area of this metric is early comparison (for the purposes of early pruning) of different training configurations, it is sufficient to observe only localisation.

Of course, it could be claimed that for binary classification it would be equally easy and computationally cheap to use either precision and recall values, or even

a normal cross-entropy loss for comparing models and pruning despite using other types of loss for actual training. This is not entirely true, and using probability distribution based metric has several advantages.

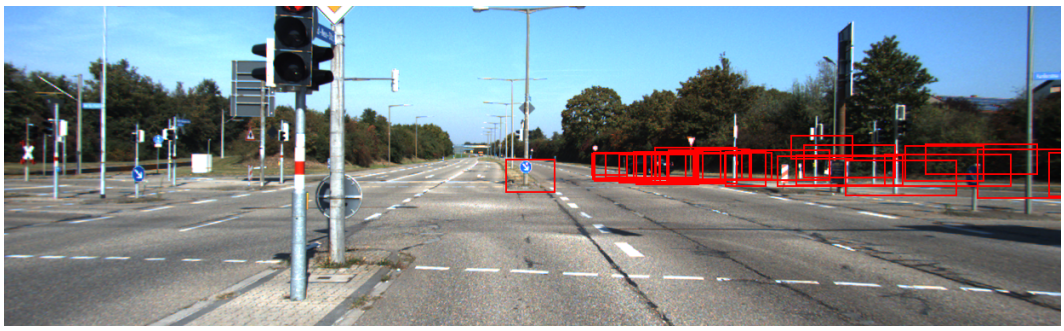
First, for a meaningful comparison against the ground truth there should be a fixed probability threshold that could be used to determine whether a estimation is good. This is problematic as pruning unsuccessful configurations should happen rather early on in the training process, and fixing such threshold would assume that all successful configurations converge at the same speed and manner. This is not a case when the training configurations differ substantially. While our solution measures convergence, it does so by observing the qualities of the whole distribution, not of each single estimation.

Second, both precision metric (measuring how many positive estimations are actually true), and recall metric (measuring how many real objects are actually found) are sensitive to absolute values. For example, it is not meaningful to compare two models on an early training phase where one consistently outputs 2 false positives irrespective of the number of real objects, and the second model consistently outputs 10% false positives depending on the number of real objects. Instead of evaluating estimation quality at the early stage, our experience shows that it is more meaningful to measure the speed of convergence via probability distribution as this metric will not depend so much on specific model behaviour.

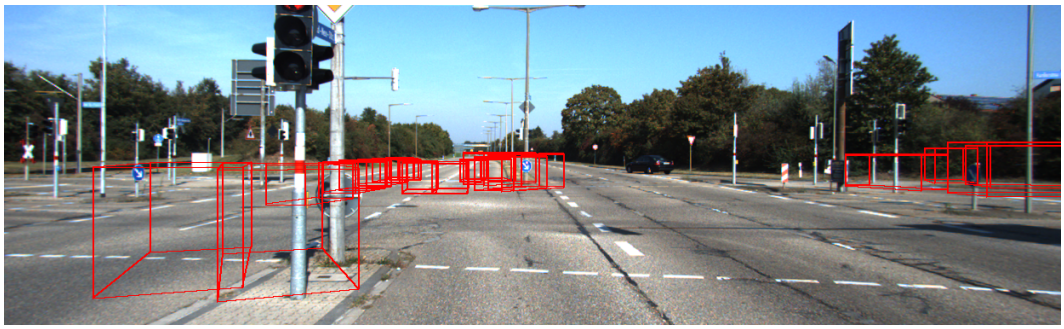
Third, cross-entropy loss value is a better metric but its aggregate value also measures goodness of the model, not its convergence. As a result, a training configuration that tackles complex objects first is disadvantaged and might even get pruned over a model that prioritises easy examples.

Fourth, bounding box detection is a difficult task and metrics that measure the quality of individual estimations might not be useful at all on early stages, e.g., Figure 3.11 illustrates a case where individual estimations are still of low quality, but the models are approaching the solution. Red bounding boxes are estimations made by the model and projected into the image. We observe that model in (a) is much closer to capturing the real object (a black car turning to the main road from the right side) than (b) but there is too much noise to capture this numerically. As a result, accuracy-based metrics like precision and recall are equally close to 0, but the probability distribution metric differentiates the models showing that (a) is closer to a real solution than (b) as model (a) is more confident in its estimation than (b).

The probability distribution based metric manages to capture the difference in quality while precision and recall metrics do not (are equally very close to 0).



(a) Model places majority of bounding boxes (in red) around the actual vehicle.

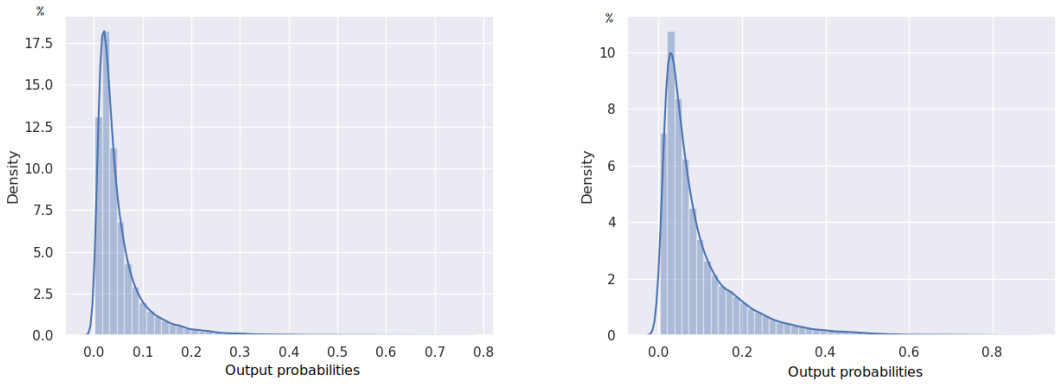


(b) Model places all bounding boxes (in red) away from the actual vehicle.

Figure 3.11: Estimations of two models in an early training phase. The aim is to place a bounding box around the car. There are many false estimations. Accuracy-based metrics like precision and recall are equally close to 0, but the probability distribution metric differentiates the models showing that (a) is closer to a real solution than (b). Scene 000017 from KITTI dataset.

Figure 3.12 displays output probability distributions of the models (KITTI scene 000017). As can be observed, the distribution in (a) is aligned more to left and hence the model is more confident in its estimations. The distribution in (b) for another model has a wider variance and the model is less confident in its estimations.

The probability distribution based metric is developed for practical purposes for pruning model configurations that will likely perform poorly. The main motivation is to save in costs of training. Therefore, there resources to extensively test this model are limited, and the results are described based on the empirical evidence collected during the research. The method was tested as follows. First, a set of configuration parameters were selected for training, altogether 50 different setups. Then, Optuna library [3] was used for hyperparameter search and the λ-value of the novel convergence assessment metric was used as a criteria for pruning the experiment. We take a note of 50% worst performing models (i.e., 25 models) and label them as to be pruned. Then we randomly select 10 of them and train further.



(a) (b)

Figure 3.12: Distribution of output probabilities of anchor points. Scene 000017 from KITTI dataset. Model (a) is more confident in its estimations than (b).

None achieves results that are comparable to the best models in the search space. The precision of those models is between 5-15% poorer than the best performing model. We further use the λ to always prune at least 50% of experiments. There is no absolute value or threshold of λ which is a good indicator of a poor model. Instead, some percentange (e.g., 20-50%) of low performing models should be pruned. We also acknowledge that to verify the method in a more rigorous manner, more experiments are needed. But as each such experiment is very costly (approximately £200-500 worth of computing time depending on configuration) we cannot afford further experiments as this is not the main focus of research.

3.6 Other improvements in 3D object detection

Additionally, this thesis uses other minor improvements over the training process outlined in the VoxelNet [183] paper. This research makes small modifications in 3D evaluation, uses simplified coordinate calculations, and a fourth data augmentation method. Finally, we experiment with using different loss functions to train the same model where one loss function is used for coarse training and then the other model for finetuning.

3.6.1 3D evaluation for best anchor box selection

VoxelNet [183] uses 2D comparison to label anchor boxes as positive or negative. Boxes that overlap most with the ground truth bounding box, are labelled positive

and others are labelled negative for the training. This work implements this process in 3D. Our method uses a full 3D anchor box comparison, including depth and rotation. This enables the method to select the best anchor box for each object volumetrically, and accordingly provide more precise training data.

For this, the method first builds a grid of bounding boxes (which are called anchor boxes) uniformly distributed along the point cloud. It then compares the anchor boxes with actual ground truth boxes. Only after the method detects an overlap in the 2D birds-eye-view, it makes a volumetric comparison in full 3D. Such approach allows the use of anchor boxes with different heights or at different heights (unlike in birds-eye-view). As a result, positive anchor boxes are selected based on comparing their volumetric overlap with ground truth boxes as opposed to flattened 2D overlap. Regrettably, this contribution is mostly useful for more complex data and model setup. KITTI [49] does not contain data where full volumetric comparision would give an advantage. However, real world driving scenarios are more complex and this advancement will be useful when the data contains multi-level intersections, bridges etc.

3.6.2 Removing negative coordinates

In this work, coordinates where one of the axis has a negative value are referred as negative coordinates. It is computationally easier to process data without having to deal with negative coordinates. Namely, should there be points in range (-500, 500) on the horizontal x-axis, it is necessary to create a mapping for every such coordinate. Accessing the point means accessing a mapping entry in a data structure and then finding the point. On the other hand, eliminating negative coordinates allows to access data directly from the data structure where it is stored such that the x-axis coordinate value corresponds to the data structure index value. Hence coordinate shift is implemented for x-axis coordinates over all possible values X such that the x-coordinates for all lidar points are shifted (i.e., axis redefined) by the absolute value of minimal x-coordinate. For example, first find minimum x-coordinate value and then shift all coordinates by this value, i.e.,

$$S_{min} = abs(min(X)) \tag{3.18}$$

$$x \in X = x + S_{min}. \tag{3.19}$$

3.6.3 Data augmentation

Data augmentation is a crucial step in many machine learning scenarious. It allows to train more robust models that are able to recognise objects from different perspectives. Standard augmentation techniques in 2D imaging include cropping and tilting images. This allows, for example, to bring far-away objects closer or to emulate a slightly different camera angle. On the other hand, data augmentation can compensate for the lack of training data which could otherwise lead to overfitting the model to the training data. Augmentation is a cost-effective means to have more training data. VoxelNet [183] uses three forms of data augmentation. First, it rotates and translates the contents of each bounding box within the point cloud independently by a random value from a fixed range. Second, is scales the whole point cloud by a maximum of 5%. Third, it applies global rotation of the whole point cloud along the vertical axis.

We also implement the same augmentation techniques but introduce a fourth one that we consider most beneficial, namely sensor randomness emulation. As described earlier, a lidar sensor sample points from a real 3D space at a given frequency and step size (determined by the number of laser rays and distance of the object). As a result, two scans of the same static scenery are not the same. Points are sampled from slightly different locations within that scene. We choose to emulate such situation by introducing sensor randomness emulation.

While points differ in two scans of the same object, they still record the same scene and the point cloud as a whole has the same statistical properties (i.e., number of points, density and distribution). The aim of the method of sensor randomness emulation is not to change those properties, but still augment individual points. For this, the method augments each point individually by a maximum of 0.01 meters. The augmentation values are sampled uniformly at random for all the point cloud at once such that the probability distribution of augmentation values resembles Gaussian noise with zero mean and variance 0.01 meters across the whole point cloud over all the orthogonal axes. Given the point cloud X with shape $[l, w, h]$ then the sensor randomness emulation is applied to all points x on all three axis at the same time.

3.6.4 Using several loss functions in one training process

There exists a significant amount of research on different loss functions. To the best knowledge of thesis author, there are no experiments, at least in 3D point cloud object detection, that uses several loss functions in a sequence to train a single

model. Changing training parameters is not uncommon at all. Most common trick is to decrease the learning rate as the learning progresses (e.g., in [183]). The effect of changing a loss function is more unpredictable due to gradient update mechanism. Each loss function gives different feedback to the model due to different calculations of loss values. However, this does not necessarily mean that the effect is bad. Changing a loss function during the training process resembles the behaviour of an adaptive loss function. Therefore, a short test is conducted with VoxelNet 3D object detection model as follows. First, the model is trained for 150 with the combination of cross-entropy and Smooth L1 loss using 0.01 learning rate which will be decayed to 0.009 by the 150th epoch. Then the learning rate is decreased to 0.001 for refinement. The same model is further trained for 20 epochs using the adaptive multi-component loss. The results in table 3.3 clearly show that the method does not work. The first column repeats the results of cross-entropy and Smooth L1 loss from the main experiment in Table 3.2, the second column shows the results of this experiment. The results for this experiment are much worse than for the cross-entropy and Smooth L1 loss.

KITTI difficulty level	CE + Smooth L1 loss	CE + S. L1 + Adaptive loss
Easy	13.02	8.81
Medium	14.60	9.57
Hard	19.17	12.66

Table 3.3: Average precision (AP) results (in %) of training VoxelNet on KITTI validation set in 'Car' category using the combination of cross-entropy loss and Smooth L1 loss for the first 150 epochs and the novel adaptive multi-component for further 20 epochs (in the second column). Results for only using cross-entropy loss and Smooth L1 loss are shown for comparison in the first column.

Although the method did not work, it still provides important insight into the adaptive loss function. The fact that adaptive loss function was not able to improve the results of the alternative loss function in 20 epochs shows that while the loss functions are rather similar, they guide the model very differently. The inability of the model to improve in 20 epochs means that its feature space is very different from the version that was trained using only adaptive multi-component loss. To use the information provided by the adaptive multi-component loss, the model has to rearrange its feature space. This supports the hypothesis that adaptive multi-component loss not only provides faster convergence, but also guides the model to build a different, hopefully richer and more representative feature space.

Finally, to complete the experiment, we test the loss functions in a reverse order. First, we take a model trained using adaptive multi-component loss for

the first 150 epochs and then train it further 50 epochs with cross-entropy loss and Smooth L1 loss. The results presented in Table 3.4 only confirm the previous hypothesis that different loss functions direct the model to build a different feature space and not to refine the model. As can be observed, training the same model with a different kind of loss function leads to a significant decrease in performance. While adaptive multi-component loss achieved 31.01% average precision (AP) in hard category, the performance of the same model after 50 further epochs with cross-entropy and Smooth L1 loss fell to 14.19%. This combination was trained for

KITTI difficulty level	Adaptive multi-comp.	Adaptive + CE + Smooth L1
Easy	27.83	9.89
Medium	27.55	11.09
Hard	31.01	14.19

Table 3.4: Average precision (AP) results (in %) of training VoxelNet on KITTI validation set in 'Car' category using the adaptive multi-component loss for the first 150 epochs and the combination of cross-entropy loss and Smooth L1 for further 50 epochs (in the second column). Results for using only adaptive multi-component loss are shown for comparison in the first column.

150 + 50 epochs instead of 150 + 20 epochs as in the previous experiment. This is because the loss value was fluctuating more than 70% after 150 + 20 epochs, i.e., the model was not stable enough for evaluation. The results at 150 + 20 epochs would have been very different from both 150 + 19 and 150 + 21 epochs, but still marginal (e.g., AP for all categories was between 2-3% at epoch 150 + 20). This indicates that adaptive multi-component loss leads for a faster convergence of the model as it was relatively stable, i.e. there were no extreme fluctuations after 150 + 20 epochs when adaptive multi-component loss was used as a second loss function. In fact, the results are similar to a performance of a model that was trained from the scratch for 50 epochs.

3.7 Summary

This chapter introduces advancements in 3D object detection. The work used VoxelNet as a baseline model and offers several improvements. First, we build an adaptive multi-component loss function that combines cross-entropy loss with focal loss [90] and adds a novel modulating factor p for bounding box regression loss. Localisation probability vector is detached from the computational graph of a deep learning model and is used as the factor p. This allows to use more precise bounding boxes and speed up training due to forcing the model to solve a simpler task first. Ex-

periments show that the novel loss function outperforms the standard combination of cross-entropy loss and Smooth L1 loss. Second, probability distribution based metric allows faster training due to fast detection of configurations that are not converging. Third, a new data augmentation method called sensor randomness emulation is introduced for 3D point clouds that is specific to sparse data and not derived from a 2D. This allows the model to learn same scenes as if it was recorded with several different scans, reducing overfitting. Finally, the experiments show that it is not sensible to sequentially combine very different loss functions for refinement as this decreases the performance of the model.

Chapter 4

Semantic Segmentation of Point Clouds

4.1 Overview of semantic segmentation for point clouds

Point cloud semantic segmentation is a process of assigning a class label for each point in the point cloud. An example of a segmented point cloud from Semantic KITTI dataset [15] is shown in Figure 4.1 where point cloud is represented as 2D birds eye-view image for better comprehension. The point cloud contains various objects and surfaces. Altogether there are approximately 20 different semantic classes in the figure. For example, moving car is denoted by light green and road surface by purple colour. Semantic segmentation is a challenging task and there are still many research problems to address. Separating moving objects from static objects is one of them. For example, the car on the other side of the intersection has some blue colour on the roof. This is an error of a semantic segmentation model as such blue denotes a static car which is not moving. The circular white region near the centre of the intersection is a blind spot and not an error. It happens in this example scene because lidar is placed on the roof of the car and no emitted laser ray touches any object in the white area.

By 2019 it became evident that 3D object detection alone was not sufficient for point cloud processing needs of a fully autonomous vehicle. 3D object detection had researched a point where any further gain in accuracy came with a cost of increasingly more complex model architecture. For example, Frustum ConvNet [168] proposed a complex multi-resolution architecture to achieve approximately 0.2% gain over the previous best result of PointRCNN [143]. At the same time, two equally important factors withheld the development of point cloud segmentation

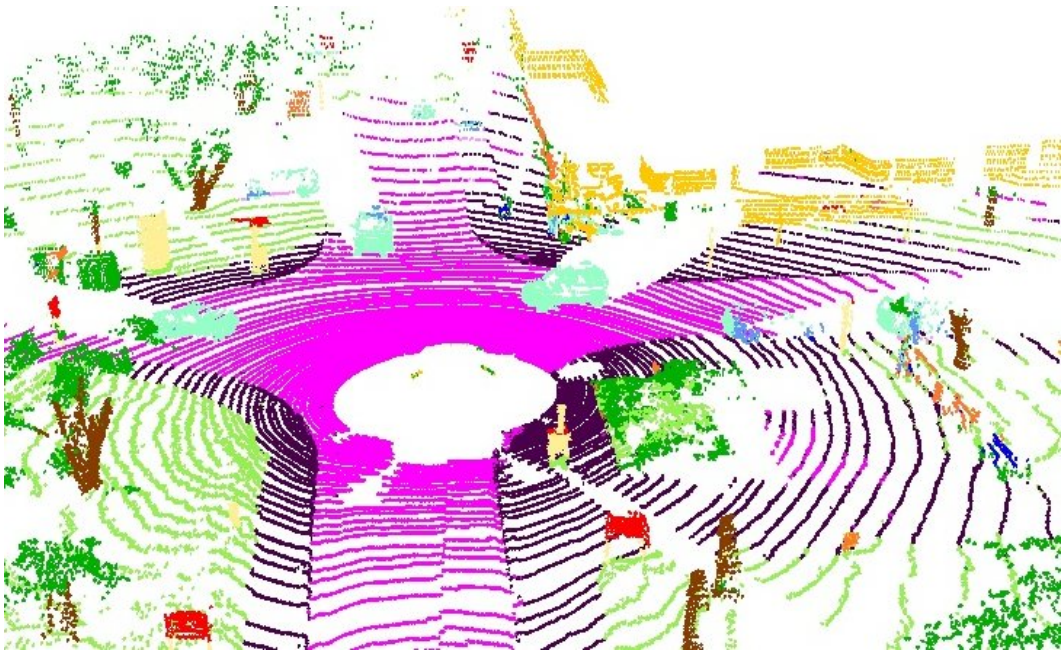


Figure 4.1: An example of a semantic segmentation of a point cloud represented as 2D birds-eye-view of scene 000030 from Semantic KITTI [15] dataset. Each coloured pixel represents a point and each colour represents a different semantic class. White circle in the middle is a blind spot around the lidar.

model for large point clouds until 2019. First was the lack of ground truth data to train such algorithms. A typical outdoor point cloud might contain 100,000 or more points. Assigning a well validated label for each point is a huge task not only because of the amount of work involved, but also because it is difficult for humans to correctly label each point due to point cloud's sparsity. An example is provided in Figure 4.2 where the reader would find it difficult to label each point. As one can observe by comparing Figure 4.2(a) and (b), providing annotations to each point is not only labour-intensive but it is also difficult to determine the correct class. The second reason was the absence of efficient means to process sparse 3D point clouds such as to keep the high resolution that a point cloud natively has. Typically 2D images are segmented using convolutional neural networks [81] (CNN-s, please see Section 2.6.1). CNN-s apply a set of small kernels over the image. The application of such kernels enables CNN-s to construct compressed latent representations of the input data (i.e., features) which can be further used to detect objects and segment pixels. These operations are computationally conducted as tensor multiplications which enables efficient batch processing. Dense tensors are used in the process. As a result, empty space also must have a recorded value in

dense tensor. Figure 4.2 illustrates the order of magnitude difference between space occupied by recorded points and empty space. Dense representation of the sparse point cloud in Figure 4.2(a) would require equally distributed data points for all of the empty space. In comparison, the image (b) is inherently dense, each pixel always has an associated value.

(a) (b)

Figure 4.2: Sparse 3D point cloud (a) and its corresponding image (b) from KITTI scene 000045 (aspect ratio and exact sensor position are different). Key to interpret the point cloud: black - empty space; light blue - recorded point; light green - bounding box showing object location.

Dense tensors work well with 2D images where each possible pixel location has a value attached to it. However, point clouds are sparse and do not have data representation for empty space. Treating point cloud data as dense tensors is computationally infeasible as the size of 3D space is several orders of magnitude larger than the size of 2D space. As presented in Section 3.2, most point cloud methods have hence used some kind of data reduction [172, 183] or other optimisations [143, 146]. As a result of such reductions, most algorithms were inevitably built in way that a final output of a model could not be related to a specific point in the original data but rather to a local area. Alternatively, graph neural networks were proposed [166] as graphs are inherently sparse, but they also suffer from the same problem - large outdoor point clouds are infeasible to process. Hence, semantic segmentation for large outdoor point clouds was not practically possible as it requires making a prediction for each point in the original point cloud. Relating the input to output while having an acceptable computational overhead has been the main challenge for 3D semantic segmentation.

4.2 Sparse point cloud processing

Semantic segmentation of outdoor 3D point clouds has been made feasible by algorithms and software that allow sparse convolutions (e.g., [28, 55, 56, 156]), i.e.,

convolutions that are applicable without having to explicitly represent empty space in a dataset. Since point clouds are inherently sparse, sparse convolution allows using covolutional operator without having to fill in the empty space. While sparse convolutions were already used for some 3D object detection methods (e.g., SECOND [172]), early solutions were specific to the task and hence did not keep a link between input points and the output. Instead, the output was mapped to a specific voxel.

The foundational work on sparse convolutions is from Graham [55] already from 2014 when he worked at the University of Warwick. The work was later extended [56] to submanifold sparse convolution (SSC). SSC allows information to flow between objects that are otherwise disconnected due to sparsity in the hidden layers of neural networks. However, their implementation was able to process relatively small point cloud data of ShapeNet [26] and NYU Depth [145]. A major breakthrough in point cloud processing was achieved by Choy et al. who introduced Minkowski engine [28], a software library that implemented sparse convolutions. They developed auto-differentiation algorithm for sparse tensors to achieve that. Auto-differentiation is not needed for convolution but is required to use such convolution as part of a neural network. While Minkowski engine still uses voxels, the voxel resolution is high enough (i.e., voxels are small enough) to assume that most voxels contain points of the same class. Tang et al. derived from this idea and developed Sparse Point-Voxel Convolution (SPVConv) [156] where they add a point branch for even better resolution. It allows to connect each point from the input point cloud to the output label (as opposed to assigning a label to a larger voxel space). This is important as the point branch preserves the fine details [156] which voxel branch would otherwise lose as it stores a unified feature for the whole voxel. Information from the point branch is then used to make a more precise estimation on fine details such as on object boundaries and small objects (we refer reader to [156] for illustrated examples). Deep FusionNet [181] is a similar but slightly more complex architecture.

4.3 Benefits of semantic segmentation

Semantic segmentation allows much more precise understanding of the point cloud data as it does not limit the shape and size of any object like bounding boxes do in 3D object detection. Namely, finding 3D bounding boxes requires a generation of candidate anchor boxes (please see Section 3.2.1) using Region Proposal Network [131] or a similar approach. As a result each possible shape and size combination

adds significantly to the overall computational requirements. This limits a feasible number of combinations that could be searched for in 3D object detection. Semantic segmentation on the other hand has no such limitation - the search space is bounded by the size of the point cloud, objects can be of any shape and size without affecting computational requirements. Further, as there is no need to pre-define possible shapes and sizes, it is possible to search for many more object types (data classes). Semantic KITTI [15] uses 19 annotated classes for its benchmark in a static version and 27 in a dynamic version.

4.4 Segmentation of dynamic and sequential point clouds

Dyanamic object segmentation differentiates between moving and non-moving objects. For example, instead of a single "car" category there are "car" and "moving-car".

4.4.1 Challenges of dynamic segmentation

In many cases, it is rather difficult even for a human to tell whether an object is moving or not by just observing the object itself. The same applies for semantic segmentation algorithms. A car still looks the same both when it is parked (i.e., it is a static object) and when it is moving (i.e., it is a dynamic object). We can distinguish between dynamic and static objects either by context or by observing them in time.

Assigning a class by context means that we evaluate where the car is located, its surrounding and relations to other objects. This is something humans do when looking at a photo or what essentially can be inferred by a point cloud segmentation model that only uses one point cloud as input. Assigning a class by observing the object in time allows to register whether the object has moved regarding its surroundings. This is the main motivation to build a deep learning model using point cloud sequences. Hence, detecting moving objects inherently also captures similar features as for the scene flow. For example, PointFlowNet [14] predicts scene flow to understand movement (see also Section 2.6.3). Scene flow captures the movement between two (or more) scenes. While it is not the purpose of this work to compute a scene flow, it is reasonable to assume that building a model that would be capable to capture a scene flow would also aid in detecting moving objects, i.e., such model should have equivalent input. Unlike for scene flow, segmenting points is a binary decision - each point either belongs to a static or to a dynamic object. That should simplify the model when compared to other models ([14, 93]) that aim

to output precise scene flow information.

4.4.2 Segmentation of sequential point clouds

As mentioned in Section 4.1 the main limiting factor of 3D point cloud segmentation has been computational feasibility. Hence, processing sequences of point clouds has been even more difficult due to the same limits. Early attempts used recurrent networks that were successful in natural language processing. McCrae and Zakhor [104] combine PointPillars [80] with Long Short-Term Memory (LSTM) recurrent neural networks [62]. Fan and Yang test several configurations of recurrent networks [39]. Huang et al. [66] use LSTM combined with U-Net [133] type sparse convolutions, but also require additional sensor input to compensate for ego-motion of the vehicle.

The effect of processing point cloud data in sequences is to observe movement between the scenes and using that observation to make enhanced predictions. Figure 4.3 illustrates how the static environment (e.g., roads (denoted in pink), trees (i.e., with brown trunk and green leaves)) remains the same while dynamic objects (e.g., cars (light green) on roads(pink)) move. However, the lidar is placed on top of a car near the middle of the intersection) so that the recording sensor moves. As a result all the static objects also appear to move relatively to the lidar. Hence, globally static objects, like trees or buildings, are displaced in a local sensor coordinate frame when the sensor is moving. However, unlike for dynamic objects, the coordinates of the static objects in a global coordinate frame does not change.

Several approaches have been proposed to process sequences of point clouds. MeteorNet [94] generalises PointNet [126] architecture to several point clouds and fuses points from sequential point clouds. MinkowskiNet [28] develops a 4D convolution operator to handle time dimension. SpSequenceNet [142] builds an attention mechanism based on 3D convolutions. After transformer architecture [162] achieved excellent results in natural language processing, transformers have also been successfully used for segmenting 2D image sequences [182]. Point 4D Transformer [40] extends the idea to point clouds and combines 4D convolution with transformer architecture. The problem with using deep learning models from natural language processing is that like LSTM networks, transformer architecture also requires much more computations than convolutional networks do for the same amount of data. Hence points cannot be directly input into transformers and require some kind of patching of the input or other optimisations which further complicate the architecture. Such models tend to be rather slow for real time processing.

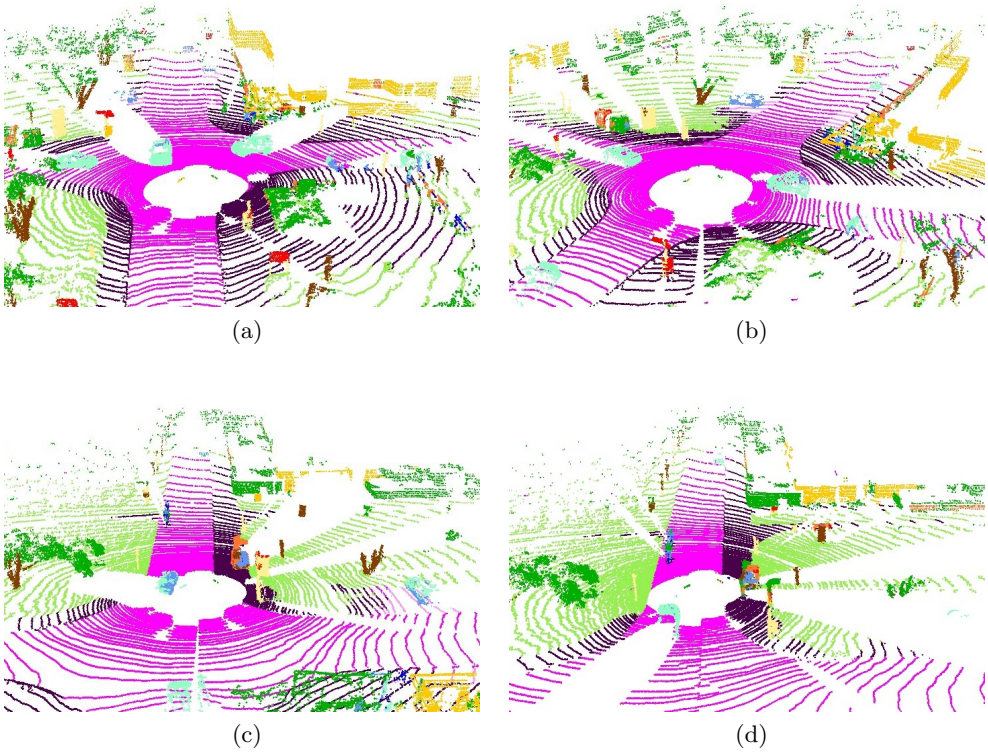


Figure 4.3: Four scenes from the same sequence from SemantiKITTI [15] dataset illustrating how the local coordinates (in sensor frame of reference) of both dynamic and static objects change when the lidar sensor moves. In scene 000040 (a) and scene 000060 (b), the car with the lidar sensor is turning to the left. In scenes 000080 and 000088 the car has completed the turn.

4.4.3 Evaluation of semantic segmentation

The results of semantic segmentation are typically evaluated using the mean value of intersection over union (IoU) with the ground truth data. This metric was popularised and made de facto standard by Pascal VOC classification challege [37, 38]. Semantic KITTI that is used in this thesis [15] also uses the same convention. Mean intersection of union (mIOU) is defined as follows [15, 37]:

$$\text{mIoU} = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FP_c + FN_c}, \tag{4.1}$$

where C represents the number of classes in the scene; TP_c, FP_c and FN_c are the number of true positives, false positives and false negatives for class c accordingly.

While at the training time a ground truth for the training set is used, the

model performance is measured against the validation data which does not overlap with the training data. This thesis uses Semantic KITTI dataset [15] similarly to other research. The full sequences '00', '01', '02', '03', '04', '05', '06', '07', '09', '10' are used for training and the sequence '08' of is used for validation.

4.5 Open problems in sequential point cloud processing

This chapter proposes a novel deep learning architecture for fusing point clouds in a latent feature space. The proposed model aims to overcome several problems that current sequential semantic segmentation models suffer from. To our best knowledge, all deep learning segmentation models that use point cloud sequences take unprocessed whole point clouds as input and use recurrent architecture or even transformer architecture [40]. Models that require unprocessed point cloud input for all scenes in the sequence [14, 94, 142] process each point cloud N times where N is the sequence length. Models with recurrent architectures, e.g., [39, 104, 175], remember their state and process each scene only once. However, recurrent architecture for point clouds is very complex as it requires turning unordered point cloud into an ordered sequence. This requires some form of complex feature manipulation. For example, work in [39] builds and performs spatiotemporally-local correlation of points in sequential point clouds using k nearest neighbour search. For computational feasibility they choose $k = 2$. Transformer architecture [162] has proven to work very well for sequential data and be superior to recurrent networks on most cases, but a trivial transformer implementation requires all-to-all comparisons of all features. This is infeasible even for 2D images. There has been a lot of research on efficient transformers recently (see [158]) to reduce a quadratic computational cost. In [32] an image is divided into 16×16 patches, flattened and the flattened patches are used as inputs to transformer all-to-all comparison architecture (encoder). However, even transformers with best optimisations to date are infeasible for whole outdoor point clouds. Similarly to dense convolutions, the original feature space is too large for the method even after optimisations. Hence, Fan et al. [40] divide point cloud into local areas, learn features for each local area and input the learned features to a computationally heavy transformer part of the model. Lai et al. [79] similarly apply transformers on computed embeddings of the sampled points after dividing point cloud into non-overlapping cubic windows, essentially voxels. They demonstrate their approach on small object and indoor point clouds.

Different approach is taken by the work in [28] where a special 4D sparse convolution is designed to include time dimension. However, they observe that

the computational cost and the number of parameters in the networks increase exponentially as they add a new dimension.

A typical point cloud processing includes reading data into memory, sorting them (as point clouds are unordered, yet most deep learning models require sorted data), learning features and making predictions. Repeating the first two steps are easily avoidable for any model with some engineering effort, for example by storing sorted point clouds in an operating memory. However, learning features is a computationally expensive process and there is no way to avoid repeating it without special model architecture. The work in this PhD thesis observes this and offers an alternative approach which (a) re-uses already learned features and (b) applies 3D convolution to keep the computational complexity low. The architecture is described in Section 4.6.

4.6 3D-SEQNET sequential semantic segmentation

This section introduces the architecture of sequential point cloud semantic segmentation deep learning model which we call 3D-SEQNET. The model makes use of a saved latent feature space of a previous point cloud to keep the computational overhead low while taking advantage of sequential data.

4.6.1 Feature fusion in latent space

To reduce the computational complexity of processing sequences of point clouds, this PhD thesis proposes a deep learning architecture which fuses point clouds in a latent feature space. Latent feature space is a set of matrices, i.e., features, that are computed in the middle layers of a neural network. It is called latent as the features are abstract and mostly not directly interpretable by a human (although some convolutional features can be visualised in a meaningful way [109]). Feature fusion in a latent space allows to skip repeated processing of the same point cloud up until to the point of feature computation.

4.6.2 Backbone network

The solution proposed in this PhD work uses a backbone network. Backbone network is a well-established deep learning network that is able to learn features of a point cloud well. Backbone network is used as a basic building block (i.e., as a core component) of a more specific neural network. Using a backbone network is a widely used technique in deep learning. For example, many specific 2D image pro-

cessing tasks are solved using a backbone network that has been proved to work on a large general dataset, for example, ResNet models [58] on ImageNet [76] dataset. This work uses a modified SPVCNN [156] network as its backbone. The original SPVCNN is optimised for a semantic segmentation of single point clouds. We omit the classifier at the end of the network and use SPVCNN as backbone network to train feature for the models presented in this thesis. A conceptual example of a typical backbone network is presented in Figure 4.4 where the left-hand side of the model learns features and the right-hand side of the model uses the features to classify each point. Only the left-hand feature learning part is used as a backbone.

Figure 4.4: An example of a deep learning model that can be used as a backbone. Mostly, only feature learning part of the model is used.

4.6.3 Setting a baseline and the object classes

A baseline model is a deep learning model that is used for comparison. A result that is better than that of the baseline model is considered an advancement. As the work in this thesis uses SPVCNN [156] as backbone network, it is logical to use SPVCNN also as a baseline. The model architecture of SPVCNN is pictured in Figure 4.5. Lidar points are fed to the point branch and voxel branch simultaneously. Voxel branch computes features downsamples tehm and then upsamples again while sending intermediate results back to the point branch which uses multi-layer perceptron (MLP) architecture to process lidar data pointwise. Finally, two branches are merged to a single feature space and a classifier outputs the estimation. We refer reader to [156] for more details.

However, SPVCNN is built to predict static classes only. This work extends it to include dynamic classes. The selection of classes is the same as in Semantic KITTI [15] moving object segmentation benchmark. For example, where the original SPVCNN version predicts ‘car’, the extended version predicts two separate classes, ‘car’ for non-moving cars and ‘moving-car’ for moving cars. As the extended version predicts 27 different classes instead of 19 as in the original version, we call the

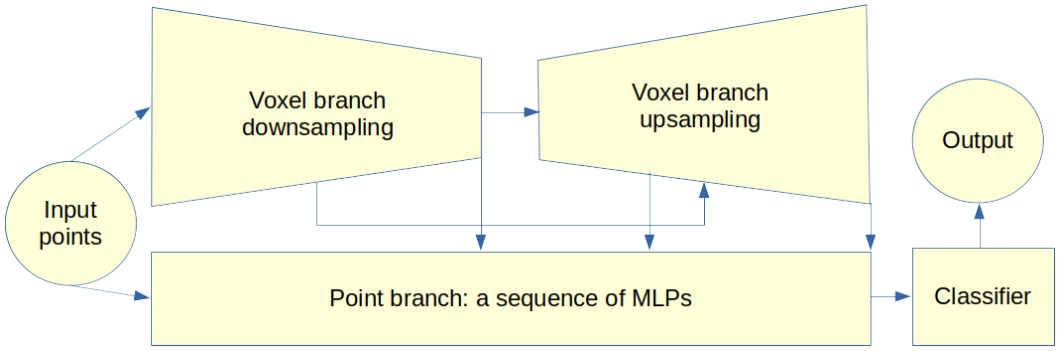


Figure 4.5: A conceptual diagram of SPVCNN [156] architecture. Our baseline model SPVCNN-27 uses the same architecture and only modifies the classifier.

extended version SPVCNN-27. Table 4.1 lists all the classes that are predicted by the extended model. The left column contains classes used by the SPVCNN and right column the classes that SPVCNN-27 uses in addition to them.

Static class	Corresponding dynamic class
road	-
sidewalk	-
parking	-
other-ground	-
building	-
car	moving-car
truck	moving-truck
bicycle	-
motorcycle	-
other-vehicle	moving-other-vehicle
vegetation	-
trunk	-
terrain	-
person	moving-person
bicyclist	moving-bicyclist
motorcyclist	moving-motorcyclist
fence	-
pole	-
traffic-sign	-
-	moving-on-rails
-	moving-bus

Table 4.1: List of classes estimated in dynamic semantic segmentation. 8 dynamic classes with moving-* name patterns are added compared to a semantic segmentation of static classes.

The same set of classes will be used in 3D-SEQNET proposed in this thesis. The first 19 classes in the first column are shared with SPVCNN with the above-mentioned exception that the semantic meaning of several general classes is different and excludes moving objects. Naturally, this only applies for classes of objects that could potentially move, e.g., a car, a pedestrian, etc. There is no dynamic equivalent for classes that are always static, e.g., 'road', 'building', 'pole', etc. Altogether the extended SPVCNN-27 predicts 8 dynamic classes. Those are named with a 'moving-' prefix in Table 4.1. Also, two further dynamic classes are extracted from 'other-vehicle' that do not have equivalent static class, 'moving-bus' and 'moving-on-rails' (the latter contains mostly trams). Similarly to the work in [156], the reason that the list of classes (both static and dynamic) is determined by the SemanticKITTI [15] benchmark it that such list can be easily evaluated against the ground truth provided by SemanticKITTI.

SPVCNN-27 that predicts 27 different classes from a single point cloud is used as a baseline model for comparison in this work. Such selection also enables measuring the effect of processing sequences of point clouds instead of single scans.

4.6.4 3D-SEQNET model architecture

The proposed architecture for 3D point cloud feature fusion model 3D-SEQNET is provided in Figure 4.6. The novel contribution of this thesis is the feature fusion which consists of three parts: non-duplicating batch merge to fuse features of two sequential point clouds, positional embedding to relate each point to the correct sequence, and convolutional feature fusion. This section first explains the concept at high level and then each block of the proposed feature fusion model is explained in detail.

There are two inputs for the model, the most recent point cloud PC_t from the current time step t and features F_{t-1} from latent feature space of the point cloud PC_{t-1} from the previous time step $t-1$. As we use stored features F_{t-1} there is no need to pass previous point clouds through the backbone part of the model. Only the most recent point cloud goes through the backbone network which outputs a set of features F_t. At that point, those features are saved to be used for processing the next point cloud. At the same time similarly saved features F_{t-1} from the previous point cloud are loaded. Both F_t and F_{t-1} are then input to a batch merge block which is the first part of feature fusion. Batch merge block merges features from F_t and F_{t-1} into a single data structure. The model architecture has no design restrictions in adding more previous scene features, e.g., F_{t-2}, F_{t-3}. However, adding them requires more GPU memory and computational power both

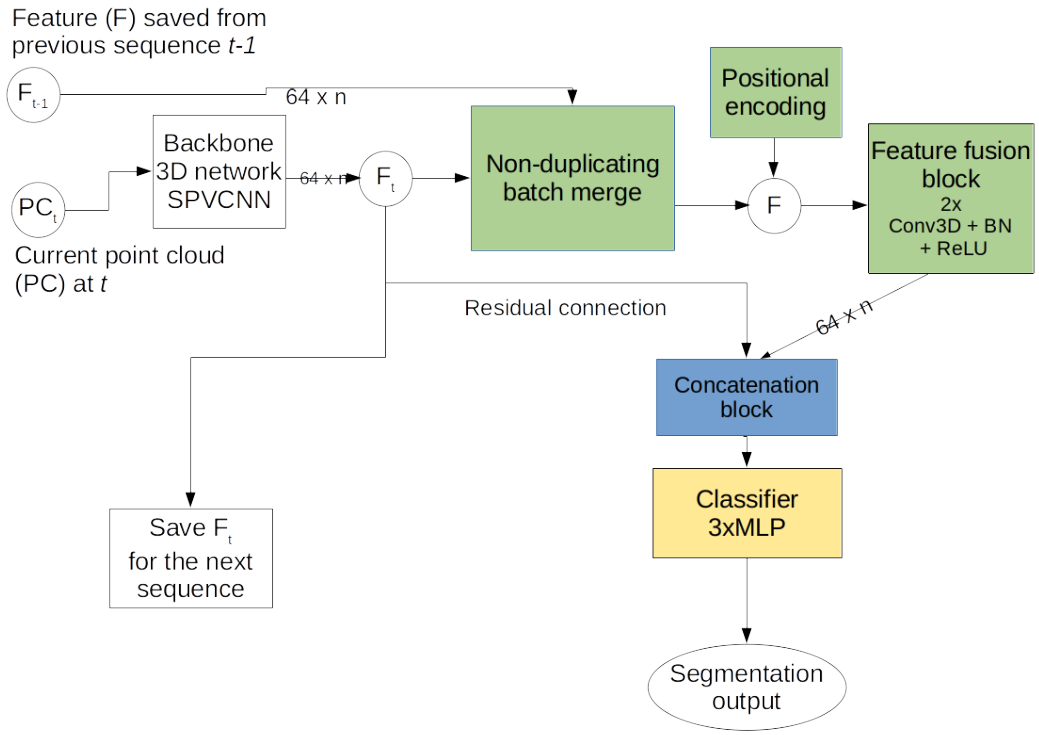


Figure 4.6: Model architecture of 3D-SEQNET, a deep learning model for sequential point cloud processing that fuses point clouds in a latent feature space. Green blocks are responsible for the feature fusion while a blue block is concatenating the original feature space and the fused feature space.

during training and for inference.

Figure 4.6 shows the best performing non-duplicating batch merge block. This and other options for batch merge are detailed in Section 4.7 below. The second part of the feature fusion consists of two stacked 3D sparse convolutions (as in [156], using convolutional kernel size $3 \times 3 \times 3$ and not changing the dimensions of the data), both followed by batch normalisation block and a ReLU activation function. Last two are standard components of deep learning models. While batch merge joins the data, convolutional feature fusion block learns new features from the merged data. To make it easier for the convolutional feature fusion block to learn relevant features, all point features are enhanced with positional encoding similarly to [94]. Positional encoding shows whether a feature belongs to F_t or to F_{t-1}. Learned features from feature fusion are then input to a concatenation block together with a copy of F_t. Such residual connection makes both fused features and unfused features available for the final layer of a model, a classifier. Experiments show that the best performance is gained by applying a linear transformation on F_t

before concatenation. Therefore, concatenation block enhances feature F_t by passing them through an additional multi-layer perceptron layer (MLP). MLP is a standard architecture of neural networks. Finally, the result is input into the classifier which outputs probabilities for each point belonging to each class. The very last step is to unroll the data into the original shape of the point cloud so that each point cloud be directly mapped to the predicted class label.

4.7 Batch merge for feature fusion

The core part of feature fusion is the process of non-duplicating batch merge. The process merges the latent features of the current point cloud with the feature space of a previous point cloud. As sparse point-voxel convolutions [156] are used in the process, each discrete voxel coordinate can only contain a single feature vector. During the development of the model, the following variations of fusion were tested:

- Keeping features from both point clouds;
- Keeping all from PC_t and only features for dynamic objects from PC_{t-1};
- Keeping all from PC_t and unique features from the previous point cloud PC_{t-1}.

4.7.1 Keeping all features

Keeping features from both point clouds PC_t and PC_{t-1} would require overcoming the restriction that sparse point-voxel convolutions require unique coordinates. Providing unique coordinates is not possible for every point as there are almost certainly points with identical coordinates on both in PC_t and PC_{t-1}. The simplest solution would be to double the feature size and fit both features of corresponding points on the same coordinate such that the joint feature space F_j would be

$$F_j = F_t \parallel F_{t-1} \tag{4.2}$$

where $\parallel$ is a concatenation operator. Assuming both feature spaces F_t and F_{t-1} are $n \times 64$ vectors, the dimension of F_j would then be $n \times 128$ where n is the number of points in a point cloud. However, such solution means that features of points on coordinates that are present only in one point cloud will be half empty and should be padded to perform tensor operations necessary to train the model. Figure 4.7 illustrates three options for feature vectors that are present on such fusion. Many feature vectors are padded on either side as they do not have equivalent point in

the other point cloud. Such configuration was tested and the initial tests show significantly (5-10%) lower mIoU with the ground truth data than the baseline model SPVCNN-27. Details are presented in Table 4.2. It is also more than 300% slower to train due to large volume of excessive padding.

Semantic segmentation model	Max mIoU	Epochs required	Epoch run time
SPVCNN-27 baseline	61.397	27	47min
Full padding (best run)	58.29	40	3h 2min

Table 4.2: Mean IoU results of segmentation models on Semantic KITTI [15] validation set for 27 classes; and the number of training epochs required to achieve the result. As can be seen, full padding model performs worse than the baseline, so this method is discarded.

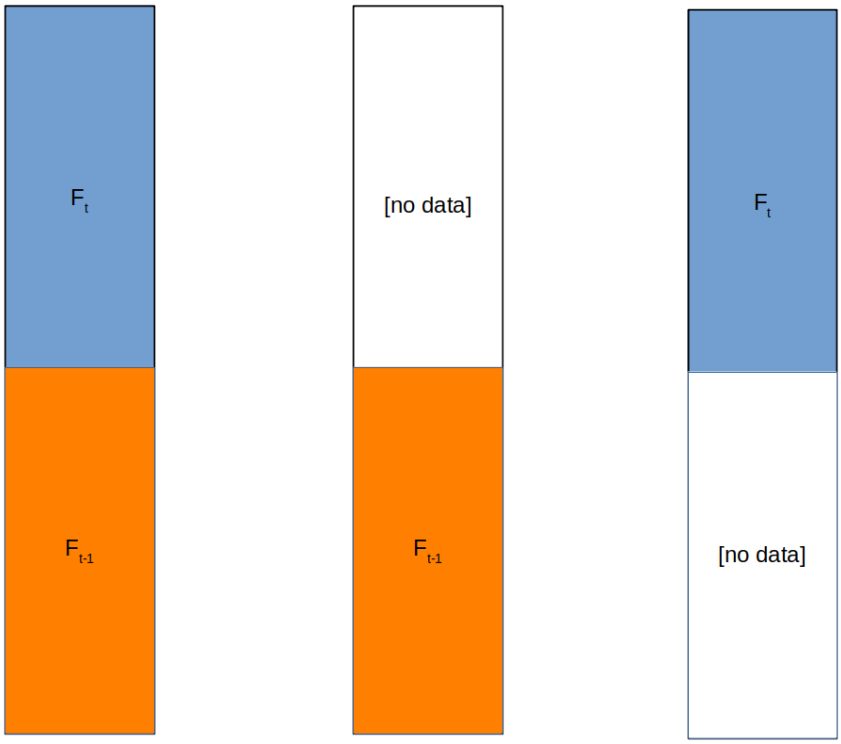


Figure 4.7: Three options for feature vectors when merging all point features from both point clouds. Many feature vectors would be half empty.

4.7.2 Keeping only dynamic points from a previous point cloud

The goal of feature fusion is to capture the dynamics of the scene better than is possible only from a single point cloud. Hence it would be appropriate to keep only

point features that represent potentially dynamic objects from the feature space F_{t-1}. This raises a fundamental question: how to determine and select those points. It would be feasible to achieve when training the model, as there is access to the ground truth values. However, when using the model for the intended purpose, there is no access to the ground truth when segmenting captured point clouds on a vehicle. Hence the only possible approach is to select potentially dynamic points by using the model output (i.e., based on model estimation) made for the previous point cloud PC_{t-1}. Such approach has a significant drawback, namely it most probably causes an accumulation of errors. When a false estimation is made for a time step $t-1$ then the features of potentially dynamic objects that will be input on time step t will also be erroneous. So, even when the model would show superb performance on training, its performance on new data would be more uncertain than for other configurations due to accumulating errors. However, such a model is still trained for the comparison and as can be seen from Figure 4.8 the performance of this model (measured on Semantic KITTI validation set) is indeed worse than for the baseline model. The most plausible interpretation is that a selection based on the model output for a previous point cloud either causes the error to accumulate over runs or a valuable information is not in the dynamic points itself, but in their relation to other points.

It is surprising that features of dynamic points appear to be useless in predicting motion. But it might have two explanations: previous explanation was wrong or the gain is not from finding dynamic points, but from dynamics of the scene which are more appearant on the edge between dynamic object and static world.

4.7.3 Non-duplicating batch merge

The process of non-duplicating batch merge for feature fusion keeps all the features with unique coordinates. It receives all the features F_t from the current point cloud and only features with non-overlapping coordinates from F_{t-1}. It has the best performance among batch merge blocks. Feature fusion is built such that the joint input F_j will be

$$F_j = F_t \cup (F_{t-1} \setminus F_t). \tag{4.3}$$

Compared to a full merge of all features, this approach results in a 50% smaller feature space F_j as can be observed by comparing Figures 4.7 and 4.9. Unlike for full merge, features from the current and previous point cloud feature space are not concatenated but kept separate. The rationale behind this approach is that only

Figure 4.8: Baseline model SPVCNN-27 performs better than a modification that fuses only point of dynamic objects (DYNAMICONLY) and selects its input into feature fusion block based on its own output on previous step.

most important edge features will be merged from the previous step. Figure 4.9 illustrates how three possible feature configurations in Figure 4.7 would be kept in a non-duplicating batch merge block. The first feature fill not contain a feature from a previous time step, while the second and the third feature do not need a padding.

4.8 Training process

The models are trained and training process is observed in epochs. Each epoch is one pass over the whole training dataset. Experiments show that when trained on Semantic KITTI dataset [15] the optimal number of epochs when training 3D-SEQNET, its modifications and the baseline model varies between 25 to 40. After this, the model starts overfitting and loses precision on the validation set.

Similarly to [156] we use cross-entropy loss, stochastic gradient descent (SGD) optimiser with the initial learning rate 0.24 and the weight decay 0.0001. SGD uses Nesterov style momentum [152] for regularisation, a standard approach to make the model more adaptable to different data.

The training process also considers a case that the previous point cloud might not be available, e.g., when booting up the system when used in a car. The

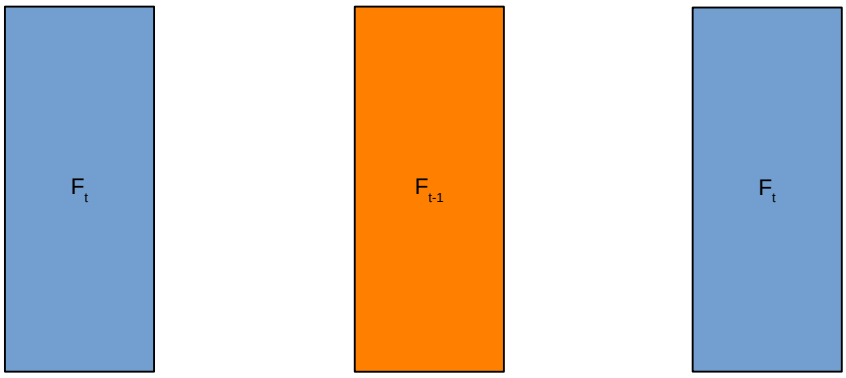


Figure 4.9: Given the same original features as in Figure 4.7, the joint feature set F_j would have smaller dimensions, keeping all from F_t and only those from F_{t-1} where there is no point on the same coordinate in F_t.

model should be robust to such missing data. Therefore, the model is also partly trained without inputting features F_{t-1}. This is to cover the case in real operational environment where there is no access to the previous point cloud feature space.

4.8.1 Two-stage training process

We also test whether it would help to train the model faster if the training was divided into two stages. The motivation is that when we start training the model, the feature space is not descriptive enough to make good predictions. Hence, we aim to test if omitting a previous feature F_{t-1} on early training phase would speed up the training of a model. In the first stage, the model is trained only with the current point cloud until it outputs reasonable estimations (which we measure by loss value). In the second stage, the model is trained further using both the current point cloud and the previous feature F_{t-1}. For this experiment, a threshold for a loss value is set at 0.99 using cross-entropy loss. When the average loss value for an epoch falls below the threshold, the training process switches from the first stage to the second, i.e., starts using both the current point cloud and the feature set of the previous point cloud. However, the experiment fails completely. The 3D-SEQNET model trained using such procedure converges slower, has approximately 30% lower mIoU at epoch 25 and shows no good characteristics. Therefore, we train all further experiments in one manner, using both PC_t and F_{t-1} as input starting from the first epoch.

4.8.2 Validating the usage of previous features

The aim of the training process of a neural network is to learn an optimal solution. If the feature space of a previous point cloud has no value to the model, one would expect the model would just ignore such data. Therefore, it is important to understand if the model actually makes use of the previous point cloud's feature space or is it rudimentary. To conduct a very basic evaluation of the approach, a previous feature space F_{t-1} is omitted for one training epoch to see if it affects the model performance. The experiment is repeated 10 times. Each time omitting a previous feature space decreases the result quality. In Figure 4.10 one such experiment is pictured. Feature space of the previous point cloud is omitted around step 27000 (each epoch takes about 2400 training steps) for one epoch. As can be observed, the mIoU metric (higher is better) instantly drops.

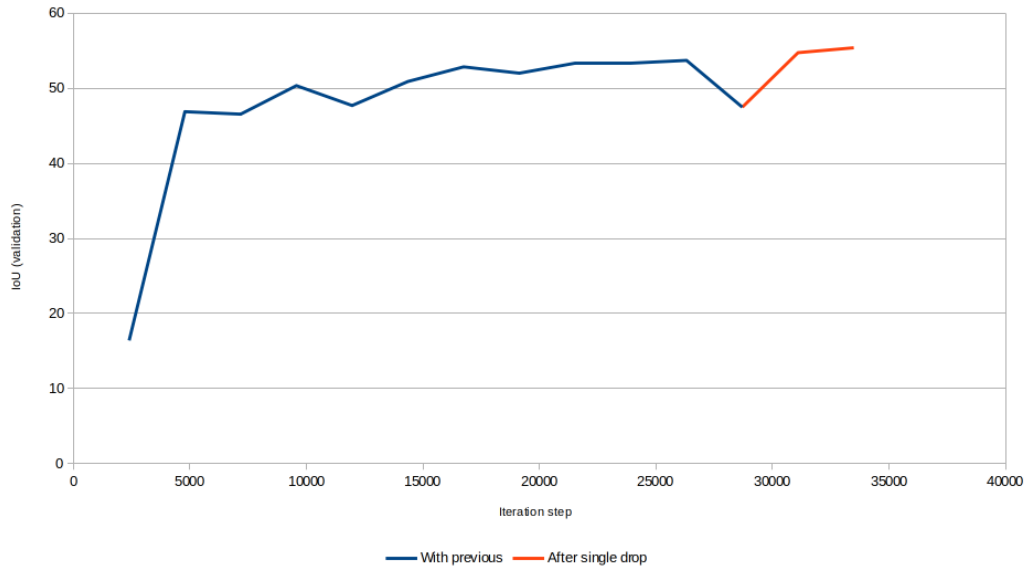


Figure 4.10: Experiment of omitting a feature space of a previous point cloud for one epoch around step 27,000 (equivalent to epoch 11), where the prediction capability falls significantly.

The experiment demonstrates that the features from a previous scene are used in the final estimation that the model outputs. After normal training is resumed and previous feature space is again used then the model performance recovers (around step 30000). However, it does not necessarily mean that the estimation would be so much worse without using previous point cloud features at all. To make sure that features from the previous point cloud indeed have an effect, the contribution of those features is further investigated in Section 4.9.2.

4.9 Understanding the model behaviour

4.9.1 Qualitative assessment of error cases

Models can be validated both qualitatively and quantitatively. Most scientific papers prefer a quantitative validation by measuring some form of precision, e.g., most often using average precision (AP) [38], mean average precision (mAP) or mIoU [37]. This allows to compare the performance of models numerically. However, for practical engineering purposes, it is as relevant to assess models qualitatively to understand where and how they succeed or fail. Figure 4.11 demonstrates a failure case of 3D-SEQNET model where half of the car (enclosed in a blue square) is estimated to be dynamic (denoted in light green) and half of it static (denoted in blue). High level metrics will only show lower metric score but will not be able to capture the essence of the error.

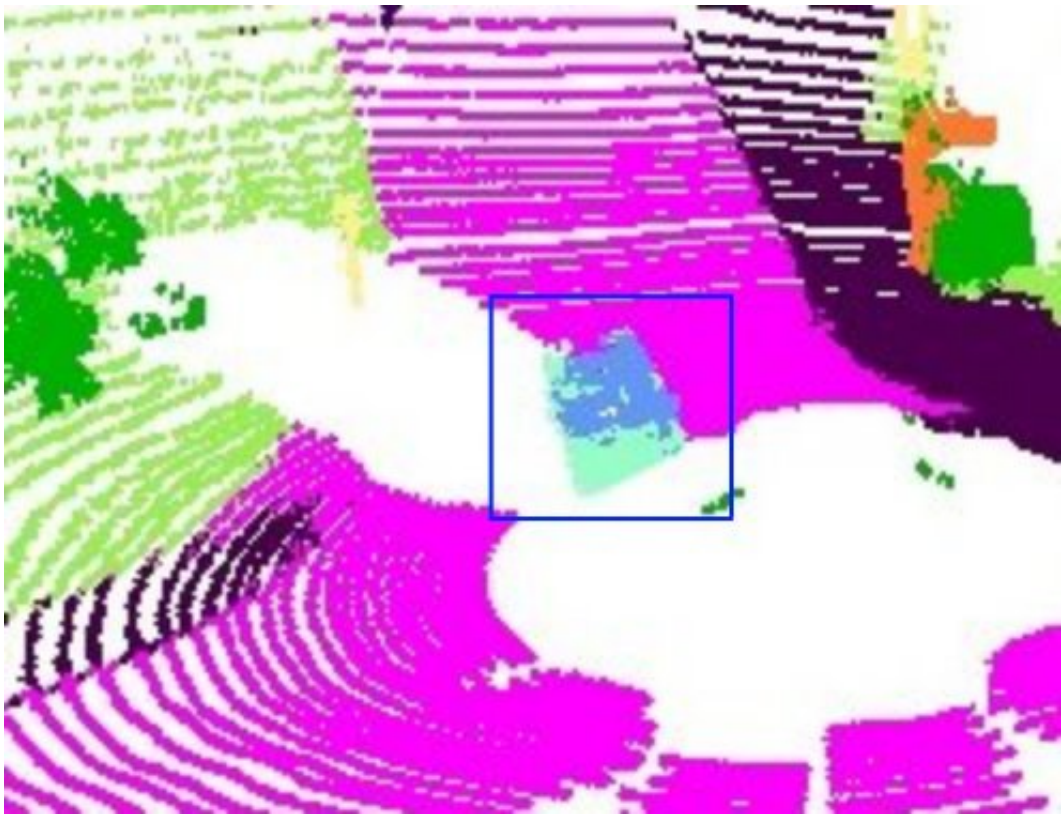


Figure 4.11: Each coloured pixel represents a point and each colour represents a different semantic class. An example of a erroneous estimation where approximately half of the car (inside a blue rectangle) is predicted as dynamic (moving, in green) and half as static (in blue).

The erroneous case in Figure 4.11 is due to two reasons. First, information required for the dynamic segmentation is only present on one side of the car. Second, as the segmentation is conducted using convolutional kernels, the size of the kernel is not large enough to capture the dynamics for all points of the car. This can be mitigated in a future work by using panoptic segmentation. Panoptic segmentation is an advanced version of semantic segmentation where the model also assigns an instance ID for each point such that points belonging to the same object (e.g., to the same car or pedestrian) have the same instance ID.

4.9.2 Model interpretability

Deep learning models are sometimes described as black boxes. It is indeed practically impossible for a human brain to grasp an exact explanation of how a given input leads to a given output, but it is not theoretically impossible. The limiting factor here is not the nature of a deep learning model - such models are mathematically well defined - but the size of the model. A modern deep learning model might include millions (or even billions) of parameters. Such large number of parameters makes it practically impossible for humans to follow the precise logic that the model uses to make an estimation. However, it is still possible to comprehend the decision process of such models.

The essence of deep learning model interpretability is to attribute model's decision to certain parts of the model or input data. If one can observe a certain model output then it is also possible to observe and understand which parts of the model affected the decision most, and which parts of the input data affected the decision most. This enables to understand and interpret the behaviour of the deep learning model. Hence, the main types of attribution are:

- Neuron attribution;
- Layer attribution;
- and Input attribution.

Neuron attribution [31] is perhaps the most accessible method as it evaluates the role of each neuron in the neural network. The method is therefore easy to use and its results are easy to understand. However, if the neural network is rather large, then evaluating each neuron independently would not give a good picture of how the network is operating. One would have to evaluate millions of data points. There, layer attribution and input attribution are more suitable method for interpreting large 3D deep learning models like 3D-SEQNET. Layer attribution

[176] helps to visualise the effect that each layer in the network has in outputting a certain estimation. This is escpecially useful for 2D convolutional layers [176]. This work uses input attribution, also known as primary attribution [151] and implements it for debugging purposes. The aim is to understand how much the previous feature space F_{t-1} contributes to the final estimation.

Input attribution using the integrated gradients (IG) algorithms [151] is tested using Captum[1] software library which offers IG implementation. As the process is rather slow, we only test 50 point clouds. The average contribution of the previous feature space is 5.1%. Given that the previous point cloud should only contribute to the segmentation of dynamic points, this can be considered a reasonable result.

4.10 Experiments and improvements

Several experiments are conducted with slightly different configuration of 3D-SEQNET. First, 3D-SEQNET is trained using the same configuration as the baseline model that is described in Section 4.6.3, including the number of epochs (25), loss function (cross entropy loss) and other hyperparameters. This allows a fair comparison between the models. The results are displayed in Figure 4.12. The optimal training point for the best results is around 60,000th iteration (i.e., 25th epoch). After that the model starts overfitting to the training data and hence loses the ability to generalise well on the validation data. Only 3D-SEQNET is trained past the optimal point. While the baseline achieves its best mIoU value of 61.397 with the ground truth on validation data at 60,000th iteration, 3D-SEQNET achieves mIoU value 61.585 a bit earlier and has 0.188 improvement over the baseline model.

The effect of fusing features raises the mIoU value by 0.188 which can be considered as a good improvement. However, currently the feature fusion provides the model the information about the dynamics of the data, but does very little to help it to interpret the data. However, as 3D convolution is a computationally expensive operation even if it is a sparse convolution, then we look for methods that would help the model without additional computational cost. Convolutional neural network can interpret the data by applying convolutional kernels on the data. The size of those kernels determines how much each kernel can "see" at once, i.e., the size determines its receptive field [98]. We use this knowledge and redesign the convolutional feature fusion block. After many experiments the design depicted in Figure 4.13 offers the best improvement. The improved convolutional feature fusion

[1] `https://captum.ai`

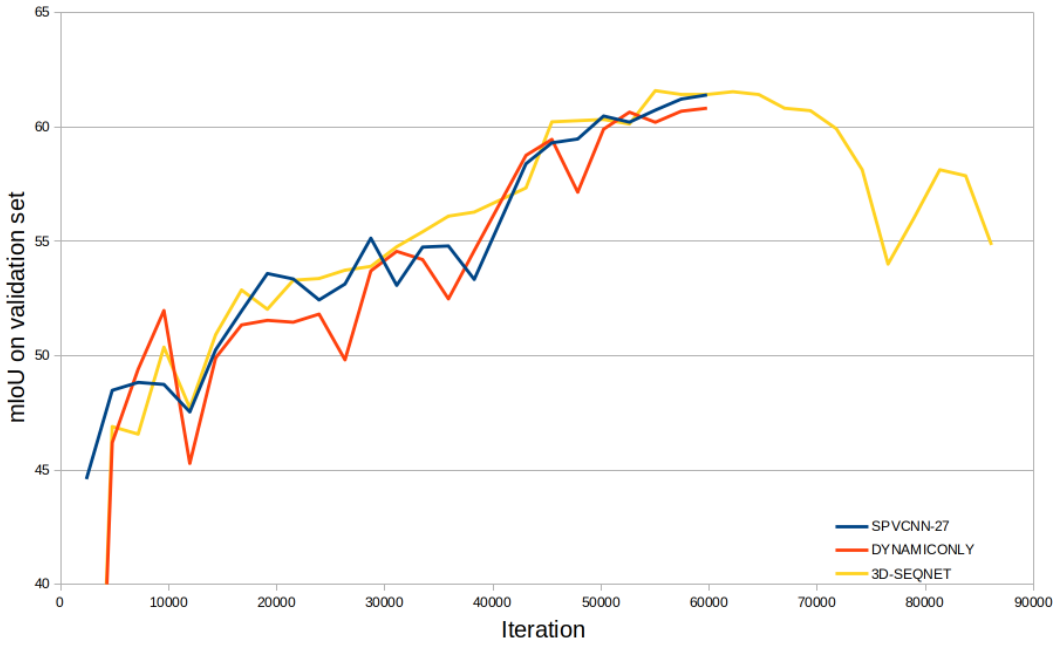


Figure 4.12: Validation results of 3D-SEQNET (in yellow) compared to the baseline model SPVCNN-27 (in blue) and the version of 3D-SEQNET where only features of dynamic points were chosen from the previous feature space F_{t-1}.

in Figure 4.13 first encodes and then decodes the feature space using $5 \times 5 \times 5$ kernels instead of $3 \times 3 \times 3$ kernels in the vanilla 3D-SEQNET version. It enables the kernel to have a significantly larger receptive field. Further, while the vanilla version kept the feature dimensions intact while fusing, the improved version shrinks the data dimensions by half ($n \times 64$ feature to $n \times 32$) forcing the model to encode the information in a smaller space. This allows the network to capture and keep higher level features that are more relevant for detecting dynamics in the scene. The decoder part on the right side of Figure 4.13 then restores the original dimension so that the resulting feature could be concatenated with the F_t that is input from residual connection in the next processing step.

The improvement to the convolutional block of the feature fusion provides a significant increase in mIoU value for the dataset. Figure 4.14 illustrates the progress. As a result of the improvements, the model is also less prone to overfit to the training dataset and achieves its peak mIoU value later than the vanilla 3D-SEQNET.

The improved model, 3D-SEQNET-enc, achieves a 62.816 mIoU on Semantic KITTI [15] validation set which is a 1.231 improvement over the vanilla 3D-SEQNET and a significant 1.419 improvement over the baseline model SPVCNN-27. The

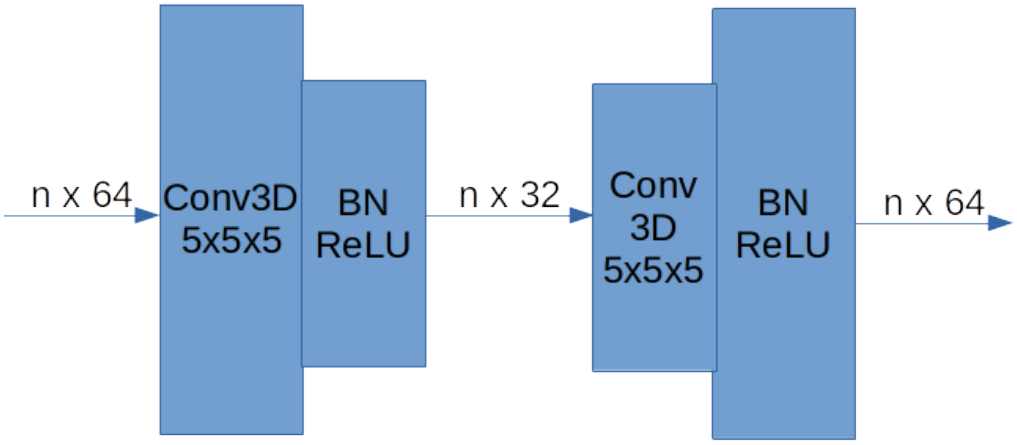


Figure 4.13: Improved convolutional feature fusion which encodes and then decodes the feature space using $5 \times 5 \times 5$ kernels. Convolutional blocks are followed by batch normalisation (BN) and ReLU activation function.

results are also presented in Table 4.3.

Segmentation model	Max mIoU	Epochs to achieve mIoU	Parameter count
SPVCNN-27	61.397	27	5,456,335
DYNAMICONLY	60.819	27	6,053,395
3D-SEQNET	61.585	25	6,053,395
3D-SEQNET-enc	62.816	40	9,298,970

Table 4.3: Mean IoU results of segmentation models on Semantic KITTI [15] validation set for 27 classes; the number of training epochs required to achieve the result; and the number of total model parameters.

In terms of model complexity, deep learning models are most commonly assessed by the number of parameters. The more parameters a model has, the costlier it is to train it. Table 4.3 shows that both DYNAMICONLY and 3D-SEQNET have a modest 10.9% rise in the number of parameters while it is able to process twice as much data as a baseline SPVCNN-27 model. They have the same number of parameters because the difference is in non-duplicating batch merge block only. The best performing 3D-SEQNET-enc however has 70.4% more parameters than the baseline model, growing the number from 5.4 million to 9.2 million. While this is a lot, it is still proportionally less than the amount of data which it is able to process compared to a baseline model. However, most of the parameter growth is attributable to the convolutional feature fusion block. Future work should analyse if the same effect could be achieved using less parameters.

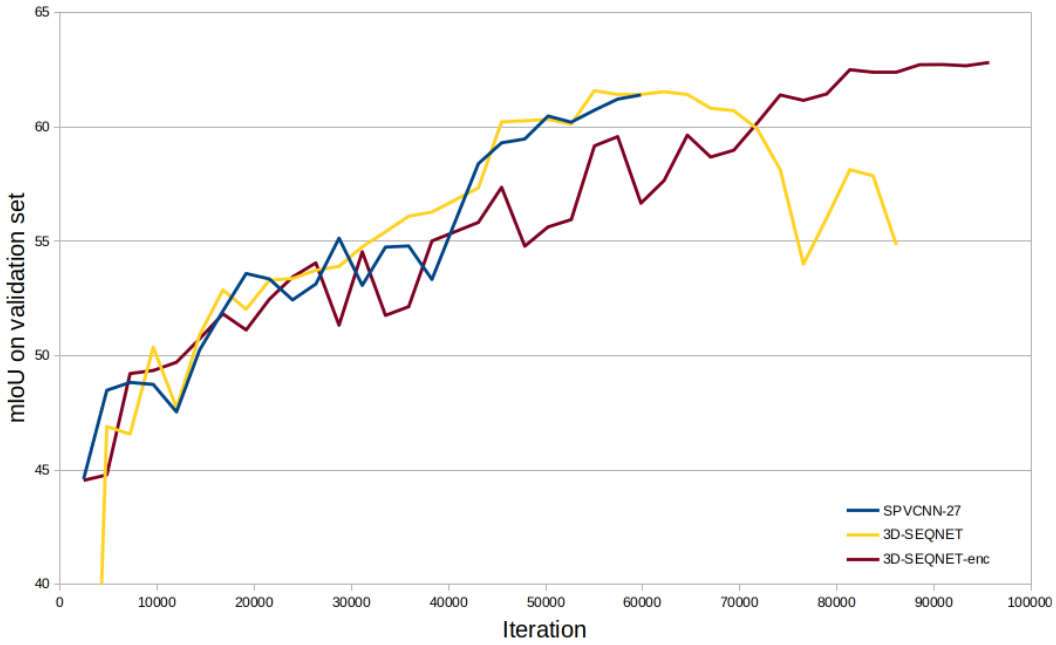


Figure 4.14: Validation mIoU of the improved 3D-SEQNET model (in maroon) compared to the baseline model SPVCNN-27 (in blue) and the vanilla version of 3D-SEQNET (in yellow).

4.11 Summary

This chapter proposed a semantic segmentation algorithm 3D-SEQNET that uses a novel feature fusion method. The method fuses features from the previous point cloud with the features of the current point cloud in the latent feature space. Non-duplicating batch merge block, positional encoding and convolutional encoder-decoder architecture are proposed to achieve the goal. To author's best knowledge this is the only sequential point cloud segmentation model which does feature fusion. Such fusion allows to process and use a previous point cloud without having to compute its feature space again. The method achieves significantly better results on Semantic KITTI validation set that the baseline model.

Chapter 5

Test Car System

This chapter details how to deploy the algorithms introduced in this thesis. First, a hardware description is provided in Section 5.1 followed by software overview in Section 5.2.

5.1 System requirements and limiting factors

Lidar installation is necessary to evaluate and test 3D object detection and semantic segmentation algorithms. However, the installation of lidar needs special equipment, hence a special test car is prepared. A 2014 Toyota Auris ST is used as a test vehicle. Ouster OS1-16 [115] is used as a lidar. The lidar is shown in Figure 5.1 from two perspectives. The lidar has a round shape and is symmetrical except for the type RJ45 cable connector (displayed as a protrusion on the top view) which is used to connect the lidar to the computer. The direction of the connector points to the negative z-axis of the output point cloud. This is important as point cloud coordinates are used for navigation. Navigation algorithms, including object detection and semantic segmentation algorithms, require consistent use of coordinate axis. This lidar emits 16 laser rays which are emitted by lasers from the middle part of the lidar, depicted as white area in the side view in Figure 5.1. Therefore this lidar has a much lower point cloud resolution than Velodyne HDL-64E that was used to create the KITTI [49] dataset. The KITTI dataset was used to train the models in this thesis. The decision to use Ouster OS1-16 is a practical choice as it was provided for this research by our industrial partner Oxford Robotics Inc[1] that sponsors this research. However, using a different lidar from the KITTI also enables to test how the algorithms perform on a different point cloud resolution.

[1]Oxford Robotics Inc. is better known under its dynium.ai trademark.

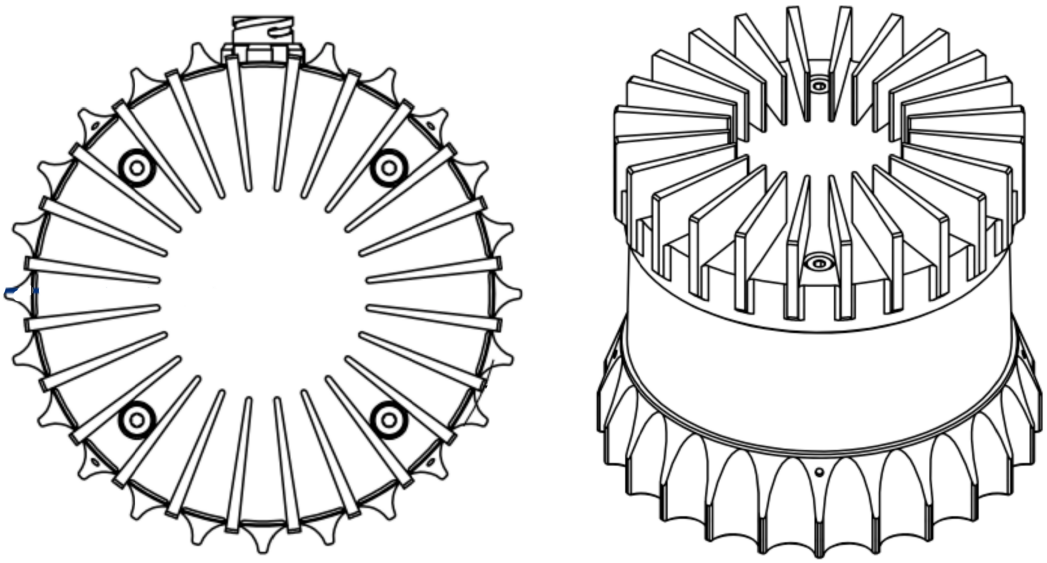


Figure 5.1: Ouster OS1-16 lidar viewed from top (left) and side (right). Protrusion on the left image is type RJ45 connector. Rotating lasers are behind the white clear area on the right hand view. The top and the bottom of the lidar consist of heat sinks. Image is taken from Ouster OS1-16 hardware user guide [115].

The criteria for placing the lidar on the car are set by the following requirements:

- 360 degrees visibility;
- Minimize blind spot;
- Avoid recording the car itself.

360 degrees visibility is required for testing purposes as it is beneficial to have an identical field of view (FOV) for the training dataset. This is not a strict requirement as the algorithms are able to work well on any point cloud shape, but having a similar FOV enables to test the algorithms in all directions. 360 degrees horizontal FOV is achieved as Ouster OS1-16 is a rotating lidar. Fast rotation enables to collect data from 360 degrees around the lidar. The rotation rate is configurable and is either 10 or 20 Hz while a lower rate enables to collect denser point cloud.

Each lidar has a blind spot which depends on their vertical FOV. Ouster OS1-16 has a vertical FOV of 33.2 degrees (16.6 degrees both up and down) [116]. Therefore, the higher the lidar is relative to the ground, the larger the blind spot around the lidar. Figure 5.2 illustrates the vertical FOV of a lidar where α is the vertical view angle and the area between the blue lines is the vertical FOV. Blue

lines represent the highest and lowert laser rays respectively. Between them there are additional 14 laser rays. FOV is by default symmetrical for both upper and lower part of the FOV but could also be reconfigured if needed. As can be observed from the figure, the blind spot around the lidar is between the lower edge of the vertical FOV and the ground level. The blind spot area is shaded with grey slanted lines in the image. Accordingly, the size of the blind spot depends on the installation height of the lidar. However, due to technical limitations, Ouster OS1-16 always has an area 0.8 m around it where it is not able to collect a point cloud. However, this will not be a problem for car installation as the blind spot illustrated in Figure 5.2 is greater than 0.8 due to vehicle height.

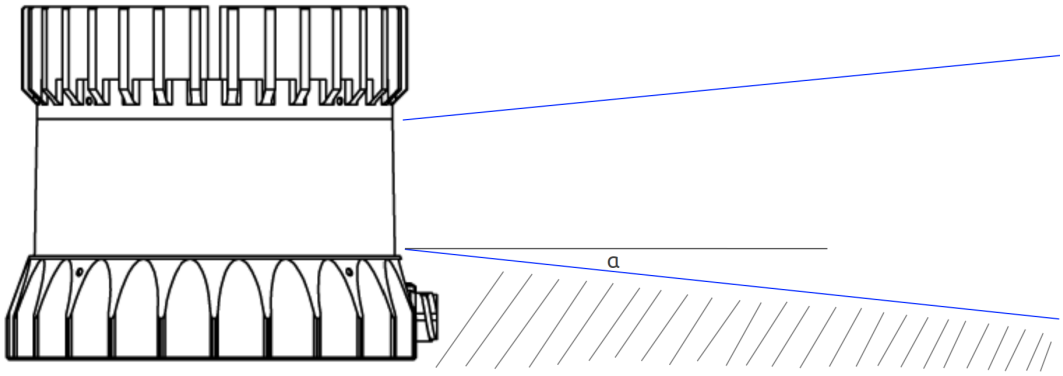


Figure 5.2: Vertical field of view of the lidar is between the blue lines. For Ouster OS1-16 this is $2 \times \alpha$ where $\alpha = 16.6°$. The striped area represents the blind spot. Drawing of the Ouster OS1-16 on the left side of the figure is taken from [115].

However, while the aim is to limit the size of the blind spot around the lidar, it is also necessary to avoid recording the car itself, i.e., a lidar ray should not touch the surface of the car. To achieve this, the lidar needs to be placed high enough so that it does not record the surface of the car but still as low as possible to minimize the blind spot. Potential self-recording scenario is illustrated in Figure 5.3 where two possible locations A and B for the lidar are depicted. The blue line originating from the possible locations is the lowest laser ray. Point B is not a suitable location for a lidar as the laser ray touches the car bonnet. Point A is a good location as the lowest laser ray does not touch the car. To minimize the blind spot, an optimal location would be somewhere between points A and B where the lowest ray does not touch the bonnet, but the gap between the ray and the bonnet is kept minimum. However, the same rule applies for other surfaces of the car too, 360° around the lidar, so the optimal location has to be found considering all areas of the car.

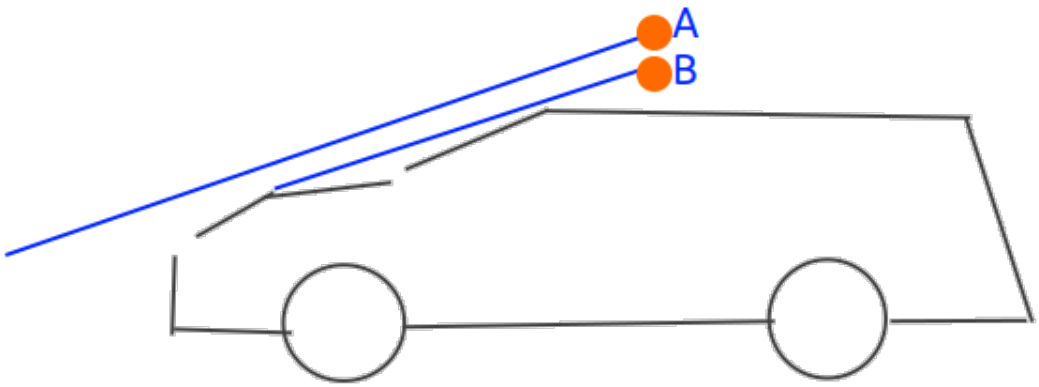


Figure 5.3: Lidar should be installed such that it is not recording the car that carries the lidar. Point B violates while point A satisfies this requirement.

5.1.1 Test car setup and hardware installation

As is clear from the requirements in Section 5.1, the optimal location for the lidar is on the roof of the car. Thus, the installation of the lidar involves the following. First, a pair of Thule roof bars is installed on the car (the silver bars visible in Figure 5.4). The roof bars are a standard equipment and are later used to mount the lidar. Roof bars are the base where it is possible to mount a dedicated lidar mounting rack (an assembled mounting rack can be seen in Figure 5.5 on top of the roof bars). To build the mounting rack it is then necessary to determine the optimal position of the lidar. Once the position has been determined, it is possible to construct a special mounting rack for the lidar. Additionally we require that it would enable to easily mount and unmount the lidar. We also have a requirement to change the angle of the lidar should it become necessary in later research.

To meet the requirements set in Section 5.1 the optimal location for the lidar is determined empirically. We connect the lidar to our computer system and start it. A special software is used to display the captured point cloud near real time. A lidar is then moved around such that its rays do not touch the car in any place, but the blind spot is minimized at the same time. We use arbitrary objects to fix the position of the lidar as illustrated in Figure 5.4 which is a hand sketch that was in fact used to build the mounting rack. Optimal position is then measured as indicated in the Figure 5.4. For this test car, the lidar should be placed 23.6 cm above the roof of the car and the mounting rack should be 12 cm above the roof bars. The distance between the two roof bars is 66 cm. The dimensions of the mounting rack are derived from those measurements. Such empirical process enables to bypass costly and time-consuming 3D modelling and calculations step

which would be an alternative way to determine the optimal position, but would require 3D model of the car. The approach in this work is many time faster and of much lower cost. Also, in Figure 5.4 Thule roof bars are already installed (depicted in silver).

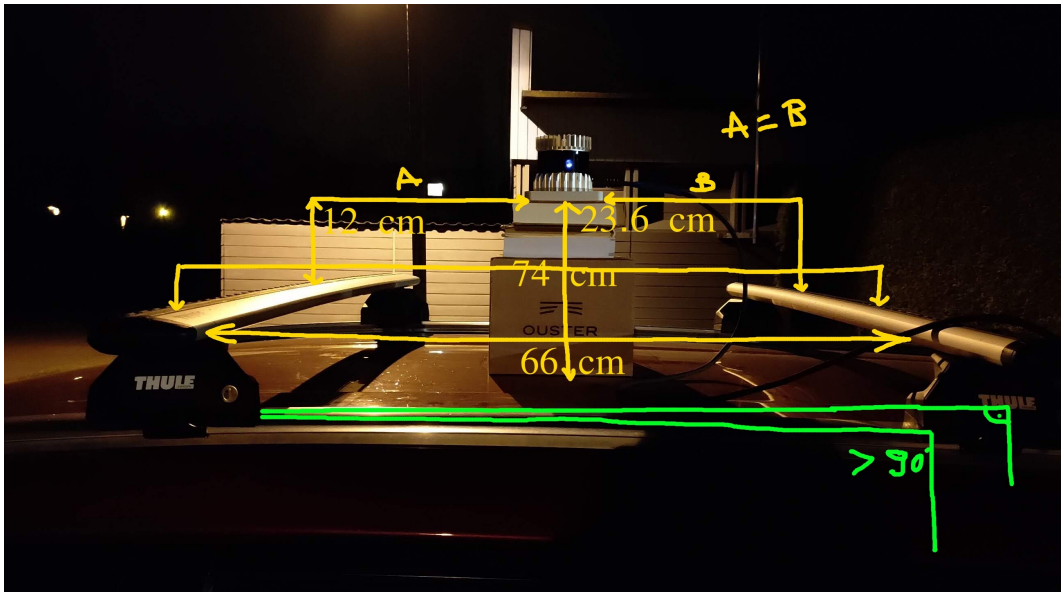


Figure 5.4: Hand sketch from determining the optimal lidar position. The sketch was used to measure important dimensions that are necessary for building the lidar mounting rack. Actual dimensions vary with car and are not important for the reader to follow.

Finally, we construct the mounting rack from steel bars. The lidar can be easily mounted and unmounted using butterfly screws. The lidar is mounted such that the RJ45 connector port is facing the rear end of the car. This is to minimize the wind pressure on the connection cable at high speeds. The cable runs into the car where it connects to the computer system inside the car. A complete lidar installation is shown in Figure 5.5. As can be seen from the figure, the car has some snow on it. To account for such weather we kept the gap between the lowest laser ray and the car surface a few centimetres rather than millimetres to avoid recording the snow on the surface of the car.

Figure 5.5: A completed mounting rack assembled on top of the test car driving in a historical district in Haapsalu, Estonia. The installation takes into account that the surface of the car might have some snow.

5.2 Software integration

The system runs on Ubuntu 20.04 operating system. To process the point clouds, a standard Robot Operating System (ROS)[2] is used to run the system. The system is depicted in Figure 5.6. A Ouster lidar SDK ROS package[3] is installed which processes the incoming point cloud data stream. As points arrive in a stream this package captures them and outputs the whole 360° point cloud for the next step. A specially programmed wrapper software then captures the point clouds. It then runs the object detection and semantic segmentation algorithms and outputs the results on the screen for human assessment. It also makes the results available for another programs. Simultaneously, the wrapper saves the point cloud for future research and validation. An industry standard rosbag file format is used to save the recorded point clouds.

[2]ROS is used to operate robotic systems but it is not a real operating system as such.
[3]Provided by Ouster Inc

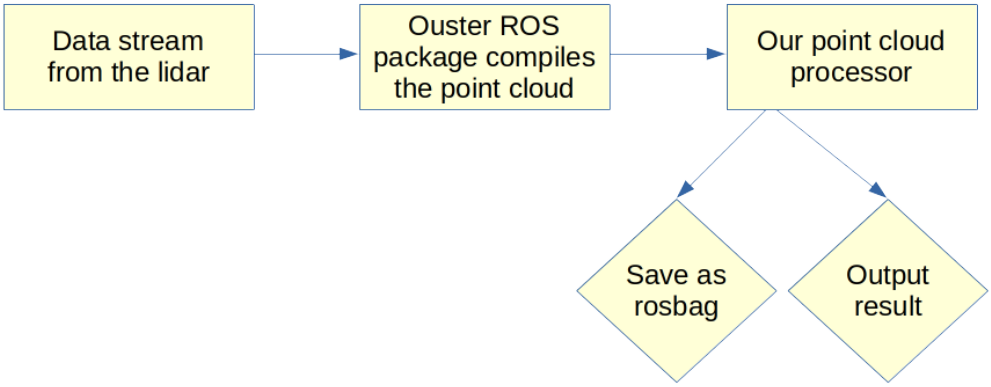


Figure 5.6: A software setup for lidar data collection and testing illustrating how an incoming data stream is processed and passed to object detection and semantic segmentation algorithms in point cloud processor node (upper right).

5.3 Summary

This chapter proposed and described the test car system that is used to collect the data for this PhD thesis. It described the hardware system setup and proposed a simple empirical method to find an optimal place for a lidar on top of the car without complex 3D calculations. It then gave an overview of the ROS-based software architecture that is used to collect the data. The actual test car used for this PhD thesis was set up accordingly and proved to work reliably.

Chapter 6

Potential Collision Test Based on Lidar Point Clouds

6.1 Introduction

6.1.1 Task-agnostic approach

The aim of this Chapter is to contribute to the development of a collision avoidance system that is able to adapt to different scenarios, i.e., not limited to any specific task or environment. For some cases, the vehicle needs to follow a given path, for others take the shortest path, and yet for others follow landmarks in the environment. Due to this requirement it is not reasonable in this thesis to integrate path planning and collision avoidance tasks into one specific method as some works [17] do for other purposes. Path planning is very much related to the specific task while collision avoidance is a more general goal. Rather, collision avoidance and path planning should follow the high cohesion and low coupling principle where components are separate from each other, but work together for a greater navigation goal.

This chapter presents the construction of a collision avoidance approach from this perspective. First, the chapter develops a testing method to assess the viability of object detection (and possibly semantic segmentation) models for collision avoidance. Second, the chapter reviews and tests the performance of object detection model that was described in Chapter 3. The collision avoidance system is tested using the test car setup that was introduced in Section 5.1.1. More specifically, we implement VoxelNet [183] for object detection and train it using 3D anchor box selection with our adaptive multi-component loss function from Section 3.4. For semantic segmentation we suggest using our 3D-SEQNET model detailed in Section 4.6.4. Finally, this Chapter analyses the test results and develops an advanced

system design to overcome one of the most crucial problems of object detection, namely false negative predictions (i.e., when a system does not detect an actual object, e.g., a car) caused by unfavourable random sampling.

Collision avoidance is defined for this thesis as a goal of an autonomous vehicle to avoid both the collisions that it might cause and collisions that might be caused by other traffic participants. There is a significant amount of work on collision avoidance, from simple systems to avoid rear-end collisions on passenger vehicles [139] to robots that learn to avoid collisions themselves while moving (i.e., use reinforcement learning instead of pre-trained models) [70]. Some works are very specific, e.g., propose an algorithm to compute a collision probability [19]. Other are more general works that develop navigation models and algorithms for autonomous vehicles [17, 167]. Finally, commercial patents are also available. Ford has patented [95] an "off-road autonomous driving system" which is essentially a flowchart of rules. For example, if there is a ditch ahead, the vehicle might drive through, and also, perhaps, it might not. Obviously, such patents are not used for this research.

6.1.2 Preliminaries and setup

The goals of this chapter will be achieved using a fully functional object detection deep learning model but without a modern navigation algorithm. Instead, the system uses a dummy navigation algorithm. As there is no fully autonomous test car available for the research in this thesis, it is not possible to plug in a real navigation algorithm. Instead, the test car will be driven by a human pilot, the author of this thesis. In addition to Ouster OS-1 lidar system installation described in Section 5.1.1, an additional camera is installed inside the test car. The camera is Logitech Brio 4K and its output is not used as part of automatic collision test, but rather to provide additional qualitative verification and guidance. The photos taken with this camera are used throughout this chapter to illustrate different test cases. The test car is shown in Figure 6.1.

As described in Section 5.2, sensor output is processed using a system based on ROS Noetic software where data collection, navigation and collision avoidance software are added. These parts are working as ROS nodes, a term used for a software component that listens, reads, processes and outputs data for other nodes. The system can run both in real time (when an onboard computer with a GPU is used) or just used for data collection (when onboard computer uses modest hardware capable of running ROS).

Figure 6.1: Test car system complete with the OS1 lidar and Logitech Brio 4K camera. Locations of the lidar and the camera are indicated by the orange arrows.

6.1.3 Navigation goal

Object detection and semantic segmentation deep learning models developed in this thesis are only part of an autonomous vehicle navigation system. Their output can be used by different navigation algorithms which are responsible for avoiding collisions with any significant objects on the road, e.g., other vehicles, pedestrians and other large objects. Furthermore, a complete autonomous navigation system will have redundancy in the system. Each sensor and subsystem will have several instances that will either work independently or in an orchestrated way.

6.2 Unsupervised validation of object detection results

6.2.1 Testing framework and criteria

What to test? As outlined in Section 6.1.2 the aim is to test against the real object detection and/or semantic segmentation model outputs that enable to foresee the probable causes of collisions.

How to test? We build a dummy navigation algorithm that enables to run the system like it was a fully autonomous operational navigation system. The dummy navigation algorithm runs object detection and/or semantic segmentation algorithms but will not use their output for navigation. The output will be collected for analyses and review. The main functional goal of the dummy navigation algorithm is to enable this test system to run in the loop as it would run if a completely functional autonomous navigation system was used.

6.2.2 Test dataset

Two sets of test data were collected and are used to develop, validate and illustrate this collision test. First, the lidar data and corresponding camera images are collected by driving the test car around the cities of Haapsalu and Tallinn in Estonia. The streets are selected to be roughly similar, but not identical to the streets where KITTI [49] and SemanticKITTI [15] datasets were collected (Karlsruhe, Germany). The streets include small roads with lots of parked cars, normal city streets with two-way traffic and some large streets with barrier between the driving directions. Figure 6.2 illustrates two example scenes. Collected data is later used to run a collision test many times with different parameters. As the navigation is not affected by the experiment, the initial conditions are identical for all configurations. Second,

(a) (b)

Figure 6.2: Example scenes of the collected dataset.

another set of data was collected in Estonia, near Vellavere and Vapramäe during the winter. This data was collected such that it was as different from KITTI as possible to allow testing models on various conditions. Therefore, the dataset contains rural winter scenes and country roads, some of the scenes have heavy snowfall. Both datasets were collected using Ouster OS-1 lidar for point clouds and Logitech Brio 4K RGB camera for collecting reference camera images. The list of collected

data is given in Table 6.1.

Data location	Environment	Data length	Data format	Sensor
Haapsalu	Winter, city	1 hrs 30 min	ROS bag	Ouster OS-1
Tallinn	Summer, city	1 hrs	ROS bag	Ouster OS-1
Vellavere	Winter, rural	1 hrs	ROS bag	Ouster OS-1
Vapramäe	Winter, rural	1 hrs	ROS bag	Ouster OS-1

Table 6.1: List and description of collected data that is used in this chapter.

6.2.3 Model training

Object detection and semantic segmentation deep learning models are trained from scratch using KITTI [49] and SemanticKITTI [15] datasets, respectively. Object detection algorithm uses Voxelnet [183] architecture. We train it for 150 epochs using stochastic gradient decent optimiser with 0.01 learning rate and using our adaptive multi-component loss function from Section 3.4. We then further refine it by training for another 20 epochs using 0.001 learning rate. Due to the limited GPU memory size we train the object detection model using batch size 2, but update gradients of the model after every 16 processed batches, i.e., 32 point clouds. This emulates a batch size 32 and provides more stable convergence of the model. 3D-SEQNET semantic segmentation from Section 4.6.4 is trained for 40 epochs using stochastic gradient decent optimiser with learning rate 0.2 and weight decay 0.0001. The training uses cross-entropy loss function.

6.2.4 Validation architecture for object detection results

The aim is to validate object detection algorithm against the collision avoidance goal outlined in Section 6.1.1. Two versions of collision validation architectures are tested. Both versions first define a navigation space where the ego-car might drive into. The method then checks whether recorded lidar points in this space are also detected by the object detection model. We call this method a collision testing system. If the model detects all objects in the navigation space, the test is successful. If there are undetected points in the space, the test fails. Failure cases are later manually evaluated to understand the cause.

First version of the collision testing system is illustrated in Figure 6.3. The system runs in a ROS environment as described in Section 5.2. The pre-processing step (a) in the upper left corner in Figure 6.3 represents software components that were outlined in Figure 5.6 and are responsible for outputting a complete lidar point

cloud. The system uses point clouds in two ways. First, it inputs them into a dummy

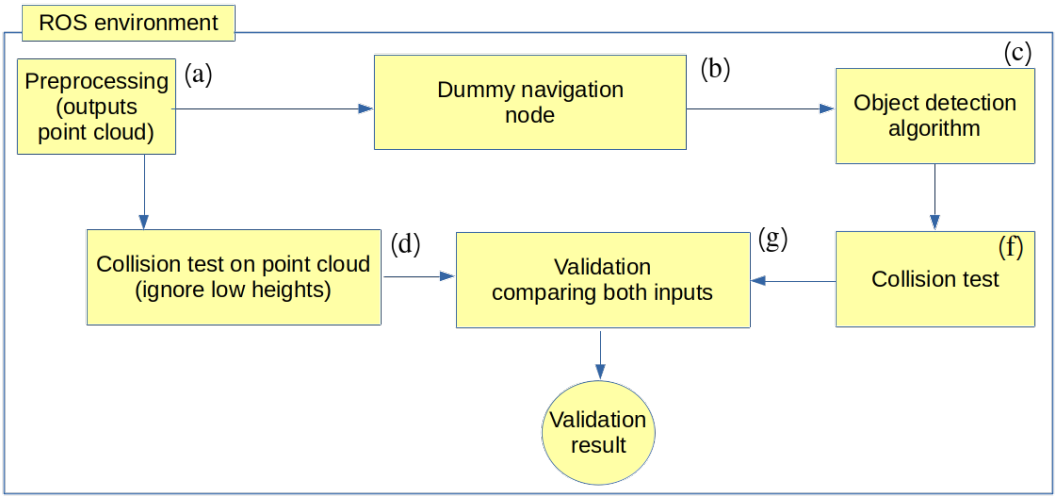


Figure 6.3: High level software architecture for the first version of collision avoidance test system that tests for collisions by comparing object detection result to pointwise collision test result. This test is applied for a single scene and repeated for each new recording.

navigation node (b) which operates object detection and/or semantic segmentation models and uses their output to perform a collision test. The test (depicted on the right side in Figure 6.3 as (f)) uses output object coordinates to localise any objects that might cause collisions. It is solely based on deep learning model output. Second, the same point cloud is sent into deterministic collision test (on the left side in Figure 6.3 as (d)) which checks if there is a risk of collision by performing a simple collision test based solely on the existence of points in a defined field of view. The middle validation component (g) takes outputs from both modules and compares if objects were found around the same locations of navigation space. The test is applied for a single scene and repeated for each new scene that is recorded. Therefore, the limitation of the test is that it does not have memory of previous scenes, so it does not compute if the object is moving. This information can be obtained from a object detection or semantic segmentation algorithm to make a navigation decision. This test is meant to be used for a single scan at a time.

However, the architecture of this system appeared to be of limited use after testing because the validation component had to compare two sets of points, taking each point from the data in one logical branch and looking it up from the data of another branch. Therefore, an updated version was developed which does not need such full point-to-point comparison, but instead removes detected and classified points from the point cloud (step (d)) and performs collision test based on remaining

points in the navigation area (step (e)). An updated collision testing architecture is illustrated in Figure 6.4. The main difference is that the new version uses sequential processing while the first version uses parallel processing. However, the new version is computationally more efficient as it does not require a point-to-point comparison.

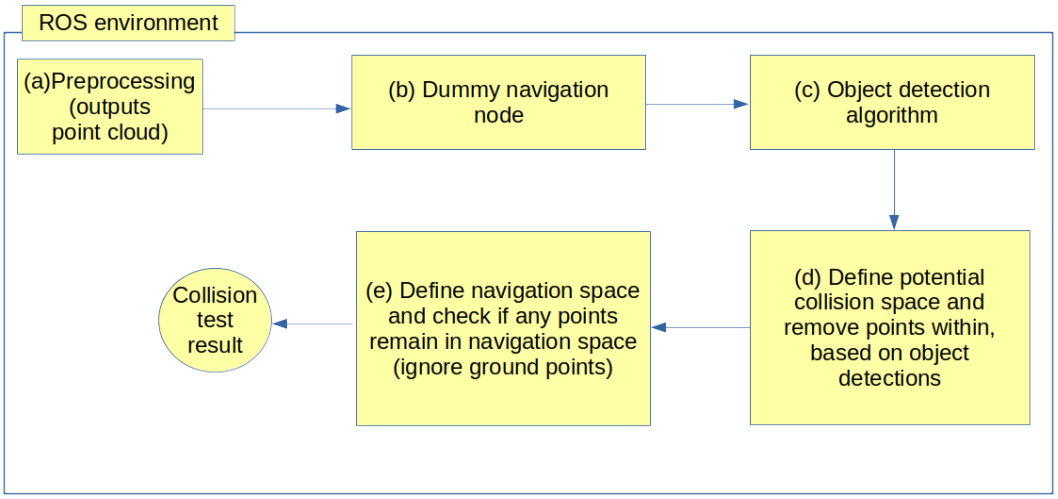


Figure 6.4: High level software architecture for the advanced version of collision avoidance test system that tests for collisions by removing detected points.

Technical details of the collision test presented in Figure 6.4 are given in the following sections.

6.2.5 Collision test

The different setups are defined to select the navigation area. Test selects areas with 5 meter increment from 3×5, 3×10 up to 3×40 meters in front of the car. All points in the navigation area are analysed for collision test. The test discards points below 0.05 meters to avoid counting ground/road surface points.

Unsupervised validation of a potential collision area

The module (d) as depicted on the lower right corner in Figure 6.4 receives object bounding box coordinates from a deep learning model output. It then enlarges the bounding box by a safety margin of 0.5 meters on all sides except for the rear (unrecorded) side of the detected object. The rear side is considered to reach to infinity along forward z-axis. It is considered to reach the infinity as there is little or no data behind the detected object. Thus, it is justified to treat this as an unsafe area for collision avoidance purposes. Even if the lidar sensor has managed

to record some points behind a detected object, it will not be considered as failure if the algorithm does not detect the occluded object. The concept is illustrated in Figure 6.5. The figure contains two detected objects A and B. Each object is

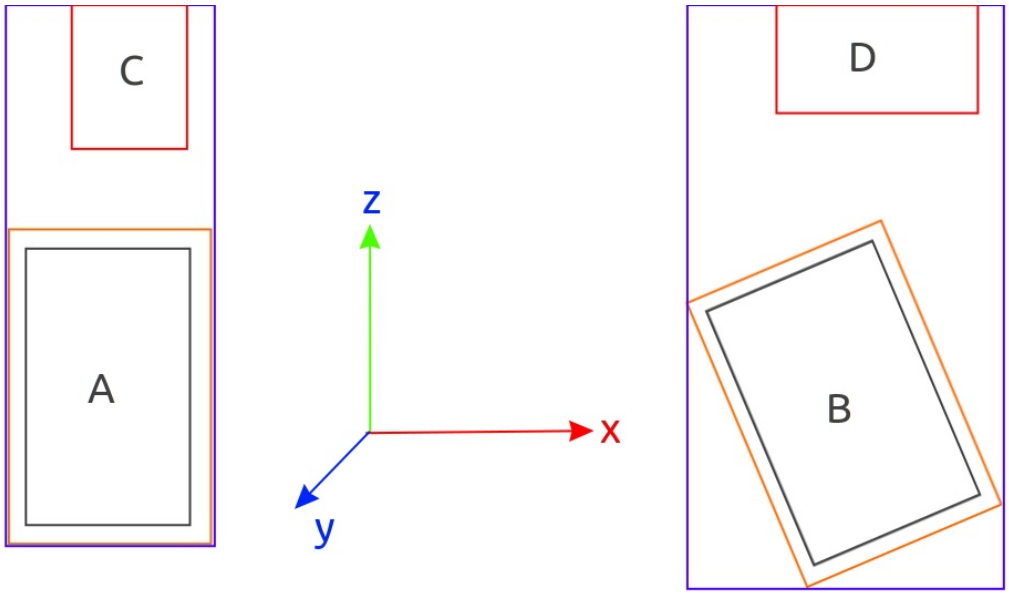


Figure 6.5: Potential collision area for two detected objects A and B (inside the blue rectangles). The area depends on the rotation of the detected object's bounding box (in grey with safety margin in orange). Detecting other objects C and D (red rectangles) that fall into the predicted collision area is not essential for validating results for the purposes practical collision avoidance. The figure uses KITTI [49] camera coordinate system.

represented by a grey bounding box. The orange box is an expanded bounding box with a safety margin. The blue box represents a potential collision area for the observing ego-car. The left object is parallel both with the forward z-axis and with the horizontal x-axis. The object on the right side is 26 degrees rotated. As can be observed, the predicted collision area is much larger for a rotated object. Such collision avoidance method does not require detecting all possible objects inside the blue rectangle while it is still practically usable. The red rectangles C and D inside the blue rectangles are occluded objects. Whether they are detected or not, is not important for the purpose of validation as they are already inside the potential collision area.

Conversion of object detection output

Object detection model outputs predicted bounding boxes parameterized as (z_{cam}, x_{cam}, y_{cam}, l, w, h, yaw) where the first three parameters are z, x and y coordinates in a KITTI [49] camera coordinate frame, l, w and h are length, width and height of the box and yaw is the rotation angle around the vertical y-axis.

Before calculating a potential collision area, the bounding box must be expanded into three dimensions. The first step is to transform bounding box parameters into a set of corner points P_{local} such that the bounding box is constructed in a local coordinate frame around box's center (0, 0, 0) using l, w and h and then rotated using standard rotation matrix:

$$P_{local} = \left(\begin{pmatrix} \frac{-w}{2} & \frac{-l}{2} & \frac{-h}{2} \\ \frac{-w}{2} & \frac{l}{2} & \frac{-h}{2} \\ \frac{w}{2} & \frac{l}{2} & \frac{h}{2} \\ \frac{w}{2} & \frac{-l}{2} & \frac{h}{2} \end{pmatrix} \begin{pmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{pmatrix}^T \right)^T . \tag{6.1}$$

After the transformation, we add original camera coordinates back to each point in P_{local} to obtain the final corner points set P_{corner} in camera coordinate frame such that

$$P_{corner} = P_{local} + \begin{pmatrix} z_{cam} & x_{cam} & y_{cam} \end{pmatrix}. \tag{6.2}$$

Similarly to [49] we assume that roll and pitch angles of the bounding box are close to 0 and hence not considered. This assumption enables to simplify the representation and use just four corner points to present all information that are required to reconstruct the whole bounding box. However, the reconstruction of the whole bounding box is only required for visualisation.

6.2.6 Formulation of potential collision space

The width of the potential collision area is defined in Section 6.2.5 as the maximum width of the bounding box along x-axis even if the bounding box is not aligned to x-axis. To achieve this we reconstruct all 8 corners of the bounding box. We then temporarily discard z-coordinates and project all points using x (horizontal) and y (vertical) coordinates to the plane as if they were in 2D. Finding maximum and minimum coordinates of both x and y axis of P_{corner} will produce facet F of the potential collision area. If s is the required safety margin, X and Y are vectors with

x and y coordinates of P_{corner} then the facet is defined via 4 corner points as

$$F = \begin{pmatrix} \max(X) + s & \max(Y) + s \\ \max(X) + s & \min(Y) - s \\ \min(X) - s & \min(Y) - s \\ \min(X) - s & \max(Y) + s \end{pmatrix}. \tag{6.3}$$

The formulation in Equation 6.3 is only applicable when using a cuboid bounding box as above. It is possible to support any possibly rotated 3D shape that is represented as a set of points P_{xyz} and then use its original facet instead of a rectangle. For this purpose we implement facet definition method by using the Quickhull algorithm [11] for finding convex hull. We use Quickhull implementation from the Scipy software library. P_{xyz} is input to convex hull algorithm and a set of hull vertices V is obtained. Facet F_{hull} can then be defined as $F_{hull} = V$ to keep the facet of arbitrary 3D shape. For a cuboid bounding box $F = F_{hull}$.

Finally, we construct the 3D shape of the potential collision space by adding back z-axis coordinates. We note Z is a vector containing z coordinates of P_{corner} and F has a shape of $n \times 2$ where n is the number of points. We construct two vectors $P_{zmin} = \min(Z)$ and $P_{zmax} = \max(Z)$ both with a shape $n \times 1$ containing the same element, minimum and maximum z coordinate respectively n times. Construction of the potential 3D collision space C is then finalised by concatenating F with z coordinates as

$$C = \begin{pmatrix} F, P_{zmin} \\ F, P_{zmax} \end{pmatrix}, \tag{6.4}$$

such that the resulting matrix C has the shape $2n \times 3$ containing all the points necessary to define the 3D shape of a potential collision space C.

Visualisation of a potential collision space

This section illustrates how the method is applied to a real point cloud. We select a scene that is tricky to validate. The scene is illustrated in Figure 6.6. The ego-vehicle (a test car used to record the data) is driving out of a parking space. The only other vehicle is parked further down the road, but not on the projected driving path of the car. As can be observed, is is not possible to tell if the observed car is indeed the only car in the vicinity as there might be other cars that are occluded by it. Object detection algorithm takes the corresponding point cloud as input and outputs one prediction for the scene, correctly recognising the car. The predicted bounding box

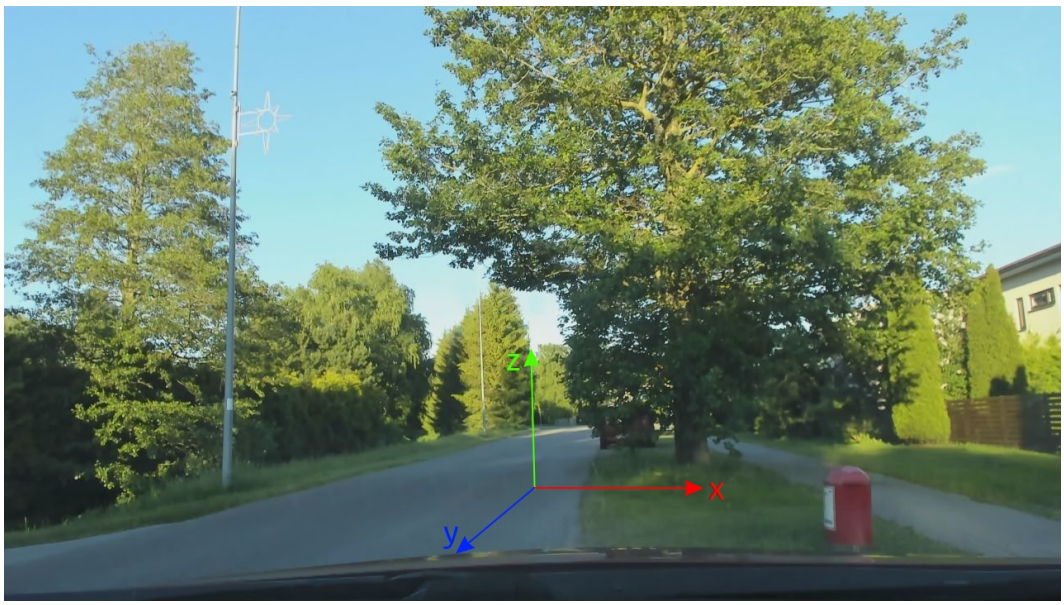


Figure 6.6: Scene from driving dataset with one parked car. Collected in Tallinn, Estonia in May 2022.

is visualised in Figure 6.7 as a coloured point cloud. Green cuboid represents a predicted bounding box. Red points fall into the corresponding potential collision space. For the purposes of validation, all red points are considered as detected. Please note that the view point of the point cloud has been changed for a better comprehension. The result represent a good collision avoidance result. The area of a potential collision is well localised. However, the limitation of this method is that if the object detection model fails to correctly detect the object, then the collision avoidance methods also fails as this method alone does not provide any redundancy.

6.2.7 Detecting collision points

The ultimate goal of the method in this section is to detect if there are any potential points that are not detected as part of object by the deep learning model. To achieve this, the method removes all points P_C within a detected potential collision space C from the whole point cloud P

$$P_{keep} = P \setminus P_C. \tag{6.5}$$

Further, a point set P_{nav} is defined as all points that fall ahead of the vehicle, in a driving direction, into a space of potential navigation. To achieve this, a selection

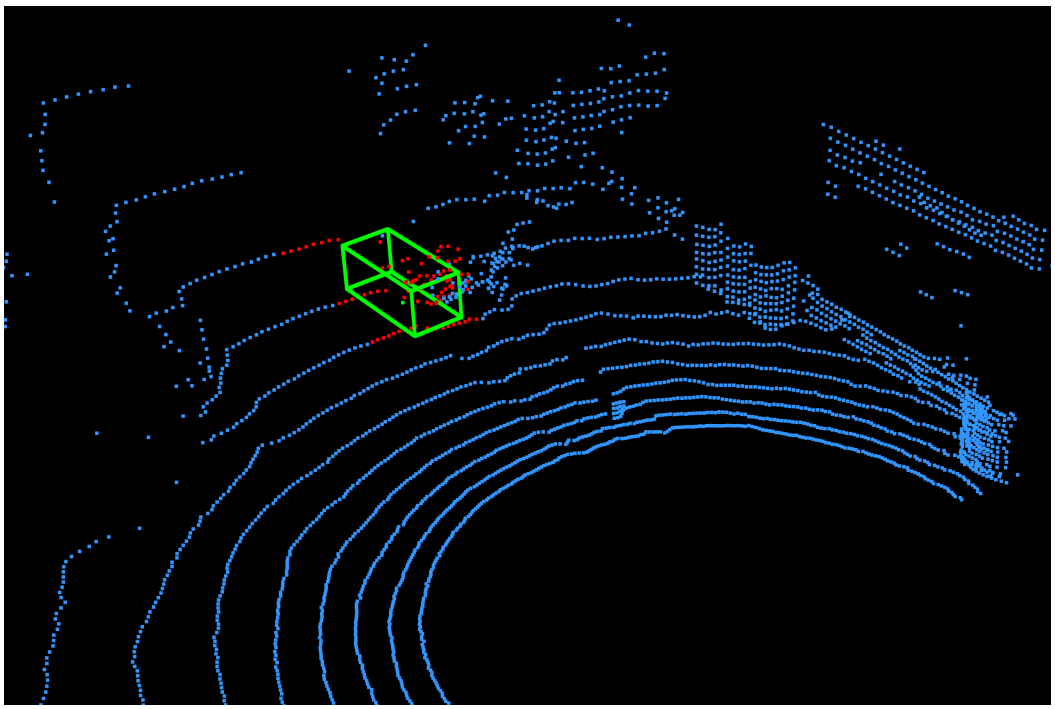
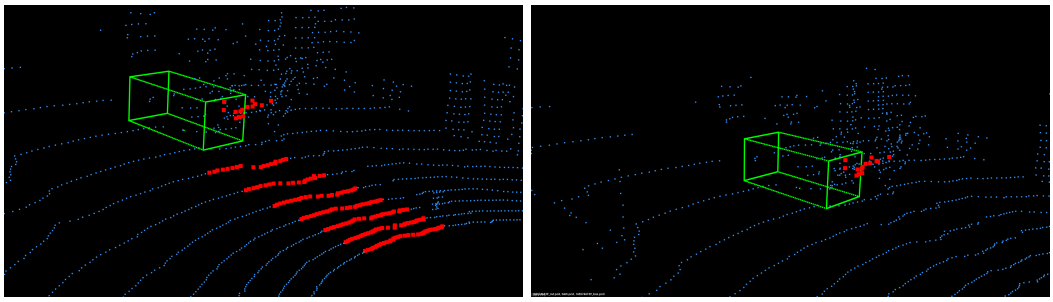


Figure 6.7: Lidar point cloud of a scene from driving dataset with one parked car. Collected in Tallinn, Estonia in May 2022. Viewpoint is changed. Green cuboid is a predicted bounding box and red points fall into a possible collision space.

criteria is applied for each point in P_{keep} based on its x, y and z coordinates as follows. Given that cw is car width with safety margin applied, ch is a maximum height coordinate of the car with safety margin applied, gh is ground height and $maxdist$ is the maximum distance along z axis where we check for collisions, we select points to P_{nav} based on criterion sel:

$$sel = x \geq \frac{-cw}{2} \wedge x \leq \frac{cw}{2} \wedge y \geq ch \wedge y \leq gh \wedge z \geq 0 \wedge z \leq maxdist. \tag{6.6}$$

The remaining points in the navigation area are then obtained by $P_{nav} = P_{keep}[sel]$. Figure 6.8 (b) illustrates the result. Red points represent the undetected object. As can be seen from a corresponding image in Figure 6.7 these points represent a overhanging tree branch. Figure 6.8 (a) and (b) also show the effect of setting ground height gh. There is no ground height restriction in Figure 6.8 (a). As a result all the ground points (in red) are registered as undetected object points, which is not desirable as they do not affect the collision probability. After applying ground height restriction in (b), the road surface is no longer interfering with collision avoidance.



(a) (b)

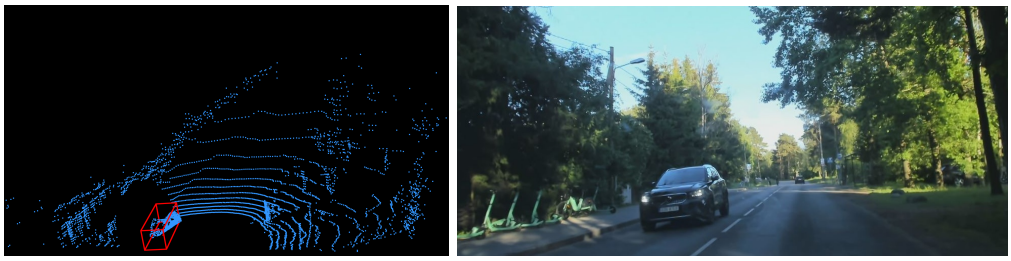
Figure 6.8: Red points fall within navigation area that is checked against potential collisions and that are undetected by the deep learning model. (a) displays all such points. (b) displays such points when ground height gh is set 0.05m above the surface height to remove ground points.

6.2.8 Limitations

Partly recorded close objects

The collision avoidance depends on an output of an object detection model. If the model output is precise then the collision avoidance also succeeds and vice versa. When reviewing the scenes that are processed using the proposed unsupervised validation system, we notice that for some scenes the object detection model fails to output a correct estimate of object bounding box. Therefore, collision avoidance system labels many points belonging to such object as undetected.

The cause of the incorrect detection is that very close objects are often only partially visible in the recorded point cloud as the lidar has a blind spot around it. Such objects are detected, but often misaligned due to their shape, but fully visible for camera as illustrated in Figure 6.9.



(a) (b)

Figure 6.9: An example of detected but misaligned object detection and a corresponding camera image.

However, such effect appears for a limited time when the object is under an unfavourable angle towards the recording sensor. Often the preceding and the following scenes have better, more precise detection. Section 6.3 below addresses this problem.

6.3 Pooling over the sequence of point clouds

Using a failed prediction might lead to collisions. As demonstrated, prediction based on one point cloud can easily fail given only slight changes in the input parameters. Hence, when using object detection models, it is risky to base decisions on only a single prediction. This section proposes a method to overcome it. The method is usable even on modest hardware but requires hardware that supports multi-threading. There are two assumptions that must hold to use this method. First, a lidar must be able to output more scans than can be processed in a single thread. We observe that most laser scanners are able to produce point clouds at a much higher rate than algorithms are able to process them. For example, the Ouster OS1 used in this thesis can output up to 20 scans per second [116] while the processing rate of our object detection model on a mid-range GPU is about 5 scans per second. Second, the slowest part of the system should be a single software component rather than input/output or data bus speed limitation. Most probably the slowest component is a model inference process, i.e., running the input data through the model to output the bounding box estimate. Given a model is already optimised, it is normally very expensive to speed up the inference time. It is much cheaper to run several inferences concurrently. The reason is that boosting the inference time for a model requires significantly more expensive and powerful hardware, which in turn requires more power, which in turn requires larger batteries. The cost of the whole system soars. Having several instances running in parallel is cheaper as both the cost of hardware and power requirements are lower.

A normal singe-threaded process that does not use our method takes a single scan at time step t, outputs the predictions and then processes scan at time step $t + 1$. As the lidar scan rate is higher than processing time then t and $t + 1$ most probably are not sequential scans. In such case intermediate scans are discarded. The proposed method, on the contrary, makes use of those intermediate scans. Given a lidar scan rate s (Hz) and processing rate p (Hz) and assuming that $s >> p$ the final prediction output time t_{out} of a single-threaded system is

$$t_{out} = \frac{1}{s} + \frac{1}{p}. \tag{6.7}$$

The proposed method to reduce prediction errors caused by a single bad prediction is illustrated in Figure 6.10 where in addition to a single scan at time t, it also uses previously discarded scans. There can be one or more such scans, up

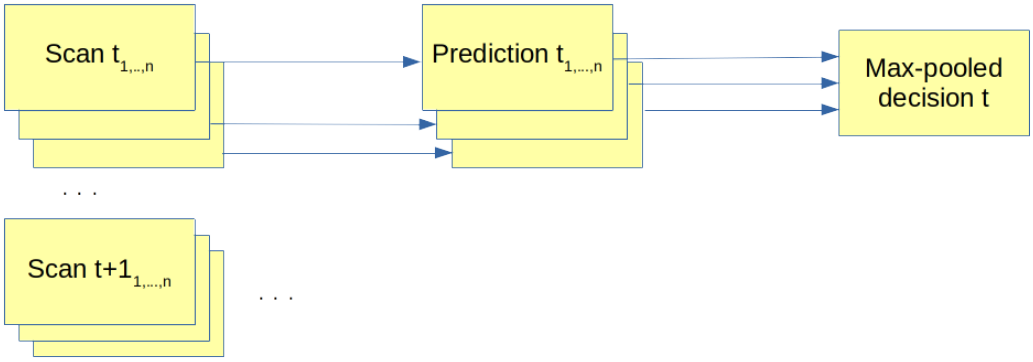


Figure 6.10: Max-pooled decision mechanism to minimise the possibility of erroneous prediction.

to n. We number time steps such that the first time step t becomes t_1, the first intermediate time step is t_2, and the last used intermediate time step is t_n. The system takes all n scans as they arrive and uses parallel processing to process them. That way several point clouds are processed concurrently. All n predictions are then input to the core component of the system, max-pooled decision block. The block is depicted on the upper right corner in Figure 6.10. It waits for the whole batch of n predictions to arrive and then outputs the pooled maximum collision perspective (worst case prediction). The total running time of max-pooled prediction is

$$t_{out} = \frac{n}{s} + \frac{1}{p}. \tag{6.8}$$

The optimal value of n depends on a specific hardware configuration, model performance and system requirements. It also depends on the speed of the vehicle and the frame rate of the lidar. The slower the vehicle and faster the frame rate, the larger n can theoretically be. If $s >> p$ then the overall increase in processing time is minimal as the max-pooling operation is a computationally simple operation. Ideally, n can also be a variable that depends on the speed of the vehicle. This work uses $n = 3$.

6.3.1 Max-pooled decision

The specific implementation of max-pooled decision depends on the navigation algorithm that processes the object detection output. The nearest object detection is

often used to find safely traversable area. Therefore, we select this to demonstrate that max-pooled decision method is computationally effective and easy to implement for most tasks. The method is derived from maximum pooling, which is a widely used concept in deep learning. For any tensor T with dimensions $x \times y \times z$, max-pooling along z would output a tensor T' with dimensions $x \times y \times 1$ where any element

$$T'_{ij} = \max(T_{ij0}, ..., T_{ijz}). \tag{6.9}$$

The only difference between max-pooling in deep learning and in this method is the definition of maximum element. The term "maximum" is defined as the value that would most likely produce a collision. For object detection, each bounding box estimation is represented as a set P containing points p that are described by 3D coordinates $x \times y \times z$. Each point represents a corner of a predicted object's bounding box. For the nearest object along forward axis X ($x = 0$ is the lidar origin) the max-pooled prediction is effectively min-pooling over x-coordinates x^p of all p in all P. If m is the number of points p in P then we first find closest point in each P:

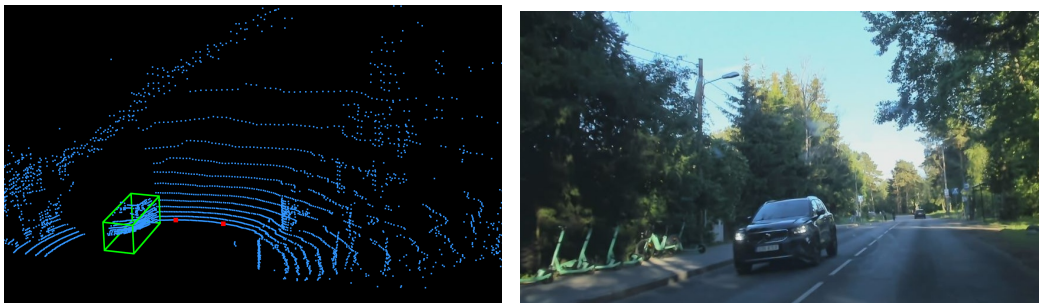
$$p_{max} = p \in P \text{ where } x^p \geq 0 \text{ and } x^p = \min_{i=0,...,m}(x_i^p). \tag{6.10}$$

We only consider points where $x^p \geq 0$ as this test only looks for forward collisions. Then we select P with minimum p_{max} value to be the max-pooled bounding box that is closest to the sensor. However, it could easily be adapted for backward collisions by looking $x_p \leq 0$ Finally, all predictions from all n point clouds can be simply concatenated before the pooling operation.

6.3.2 Experiments

The proposed method for detecting collision points and validating object detection results has a mathematically solid formulation. However, the main purpose of the method in this research project is to evaluate the model quality. Therefore, the main assessment is qualitative. The method is implemented as Python software for deployment in ROS environment. To assess the quality of models we play the recorded lidar point clouds and camera images in a ROS simulation mode. This enables us to pause and review each outputted result when needed.

Figure 6.11 illustrates that a max-pooled decision works and none of the points of the car are no longer outside of the object bounding box and hence, outside



(a) (b)

Figure 6.11: An example of a max-pooling-corrected object detection and a corresponding camera image. Also two false collision points appear on the road surface. Corresponding camera image in (b).

detected collision area. However, we notice that some road surface points are labelled as potential collision points (in red). This is most probably caused by the car suspension that changes the relative height of the lidar when recording. We raise the ground height value from 5 centimeters to 7 centimeters which fixes the problem.

The method works well to detect areas that are not covered by the object detection model. Figure 6.12 (a) illustrates a case where the object detection algorithm misplaces a bounding box such that the rear end of the car (marked with a purple circle) in front of the ego-vehicle (marked with a yellow arrow) is left out of the bounding box. Our method detects the problem accurately. We also note that

(a) (b)

Figure 6.12: An example of a detection of a potential collision. Potential collision is marked with red point in the purple circle. Ego-vehicle is marked with a yellow arrow. Another undetected vehicle is inside the yellow circle in (a), but outside of the potential collision area. Corresponding camera image in (b).

there is another undetected car in the point cloud (marked with a yellow circle). Our method does not pay attention to it as it is outside of the potential collision area (as defined in Section 6.2.6). Finally, we also detect a situation where the

method works as intended, but does not detect a potentially dangerous situation in Figure 6.13. While the model recognises that two bounding boxes are misaligned

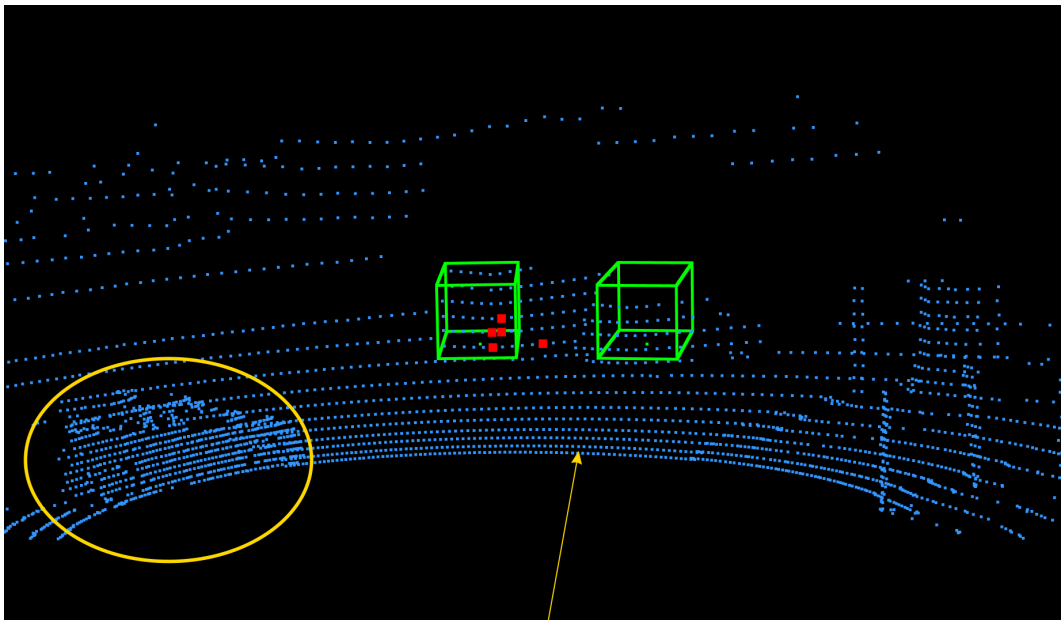


Figure 6.13: An example of an undetected potential collision situation. The model fails to alert on the vehicle from the side (inside the yellow circle).

and marks points that fall out of the safety margin (in red), it fails to alert on the approaching vehicle from the left side (inside the yellow circle). This is because the vehicle heads towards the yellow arrow itself, making a turn. This indicates a need to have a more complex definition for potential collision area when the vehicle is maneuvering.

The need for collision test algorithm rose from the need to evaluate unlabeled point clouds to test the object detection algorithm. Figure 6.14 shown an example. The collision test find points (in red on the right) which are out of any known

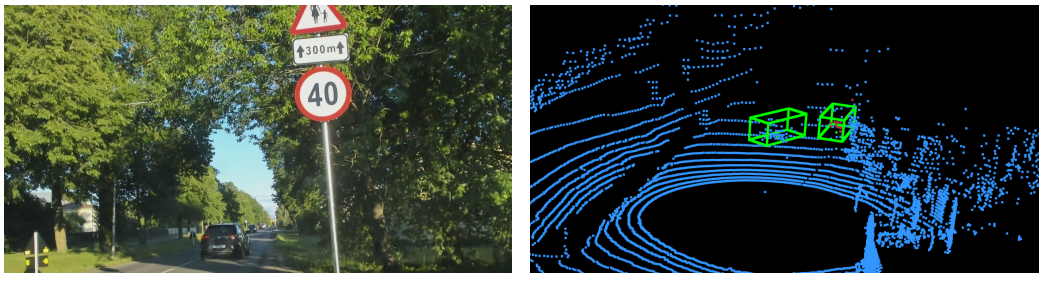
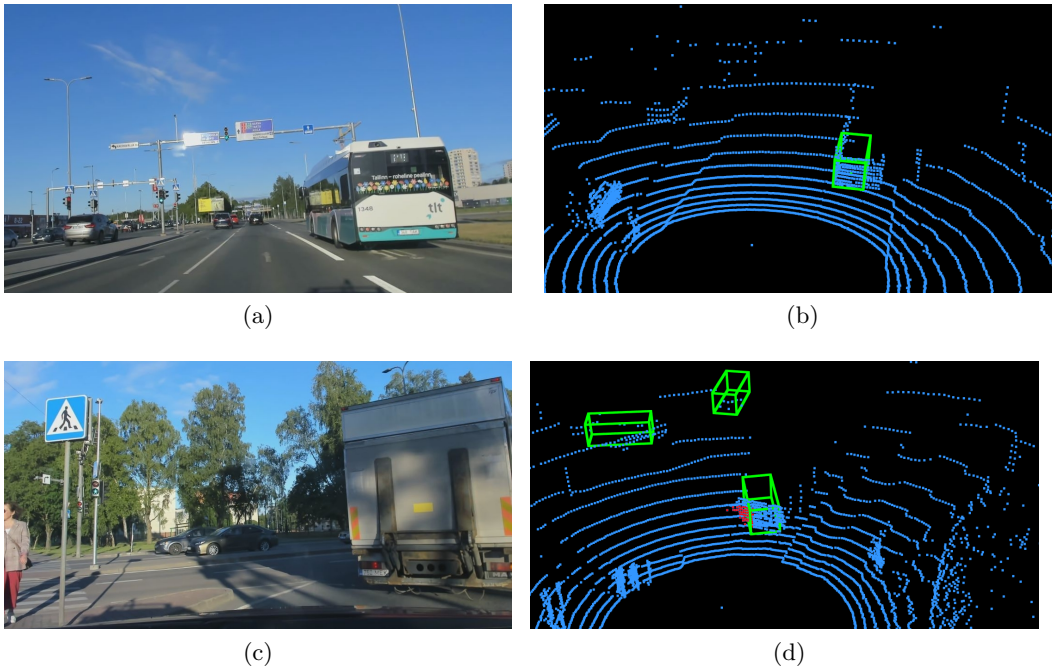


Figure 6.14: Collision test is designed to evaluate object detection results. Collision test finds points that are ahead of the vehicle but outside a known box.

bounding box and are on the path of the vehicle. These are overhanging branches. Of course, the bounding box behind the branch is an erroneous detection, but as such error will not lead to a collision, the test does not detect that (as it is not designed to do so). However, the test located point where collision might occur. To make sure that it does, we look at two further examples in Figure 6.15. The scene



(a) (b)

(c) (d)

Figure 6.15: Examples of successful collision tests.

in Figure 6.15 (a) and (b) does not indicate any collision area and indeed, there is no risk of imminent collision ahead. In scene (c) and (d) object detection algorithm has placed the bounding box under a wrong angle, so part of the truck is out of the bounding box. Collision test detects the error correctly.

6.3.3 Rural winter test

We use data collected in Vellavere and Vapramäe in Estonia (details in Table 6.1) to test both the collision test itself but also the object detection method on rural winter conditions. We discover that there are some situations where the collision test fails, namely when the car is going to descend or climb uphill, and lidar plane is not parallel to the ground as in Figure 6.16 (a), (b), (c) and (d). At that point the road elimination algorithm does not work as there is an angle between the lidar plane and the road plane while on a flat surface they are parallel to each other.

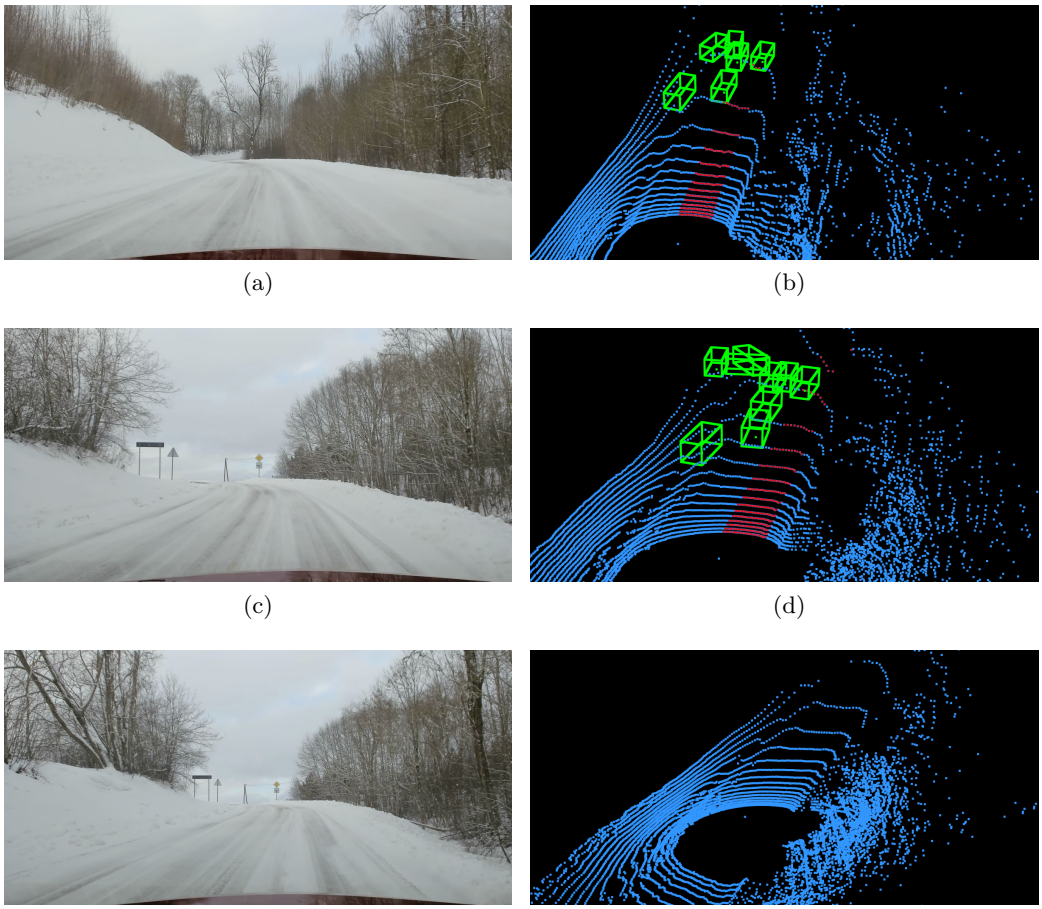
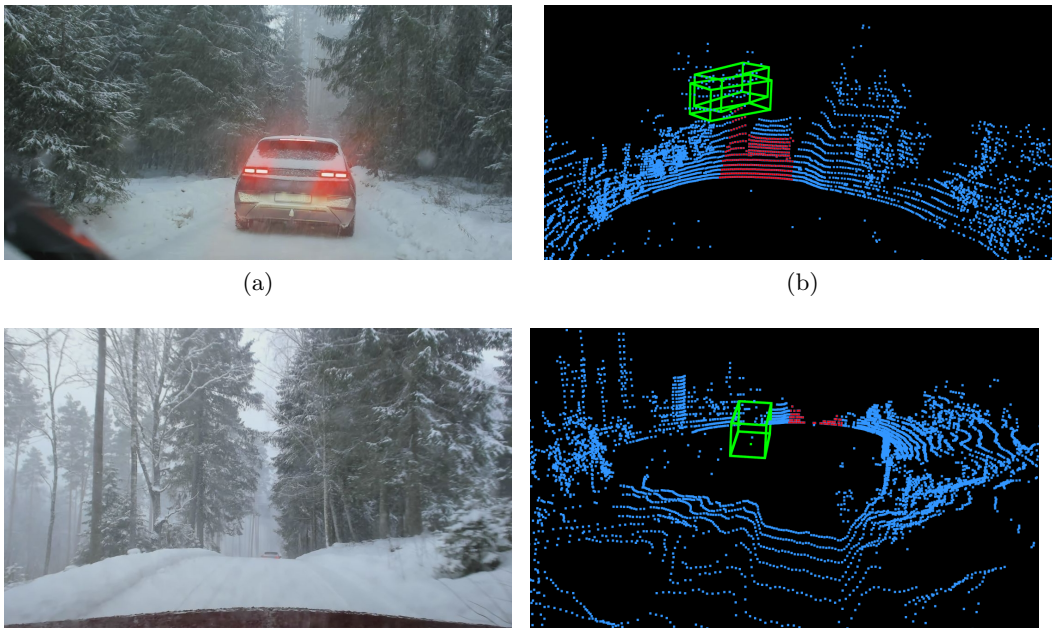


Figure 6.16: Collision test reveals shortcomings in object detection model in (a), (b), (c) and (d). Each row contains one scene. No false detections in (e) and (f). Red colour marks potential collision points.

Setting ground removal threshold to a higher value will probably fix the problem for many scenes, e.g., for small slopes. However, the proper way to do it would be to use semantic segmentation algorithm instead and remove all ground points as per detection. However, when the car has reached such position on the hill that the road ahead is again parallel with a lidar plane then the effect disappears as shown in Figure 6.16 (e) and (f).

Another anomaly in object detection algorithm is detected thanks to the collision test in deep spruce forest as in Figure 6.17 (a) and (b). The object detection algorithm misses the car in front and detects part of the spruce tree as a car. Collision test, however, detects the error and marks potential collision area. Figure 6.17 (c) and (d) present another extreme situation where the recording car is ascending



(a) (b)

(c) (d)

Figure 6.17: Examples of problems found in winter forest object detection.

but the other car in front is already descending and only the roof is barely visible. The object detection algorithm is not able to detect that car. A false detection appears where the snow wall is. However, looking closely at the point cloud we notice that there are hardly any points of the car recorded, so there was no data to detect the car. Collision test alerts about the top of hill here. It is obvious that our collision test is an useful tool for manual qualitative validation of object detection models and has pointed out many errors which we overlook on manual validation. However, this test in its current form cannot be used in a real traffic situation to detect collisions.

6.3.4 Considerations

Similar performance could also be achieved without the max-pooling component, but then it would just increase the throughput rate but would not eliminate possible erroneous predictions. The benefit of max-pooled prediction is that with a negligible increase in processing time there will be significantly less missed cases. Of course, the opposite is also true. While this system is designed to compensate for erroneously missed objects, false negatives, it will possibly also introduce more false positives as shown in a winter test in Section 6.3.3. Still, as can be observed from the prediction

results, false negatives often appear very close to the sensor while false positives are mostly distant. Also, false positives are much less harmful than false negatives as they signal an object where there is none, causing perhaps a delay in navigation. False negatives conceal an object where there is one, possibly causing a collision. Therefore, it is reasonable to trade false negatives for false positives. Due to their average distance from the lidar, false positives might not even affect the navigation algorithm at all (if the vehicle approaches the location of a false positive, the model starts outputting more precise estimates as the point cloud becomes more dense).

The processing time of the system increases by $\frac{n-1}{s}$ seconds when using max-pooled decision. This might not be suitable for every case, especially when fitting the system on existing hardware platform. However, the system requirements are rather modest and can be easily incorporated into the navigation system at design time.

6.3.5 Rationale behind the method

The method uses lidar scans with a very small recording time difference. A critical reader might rightfully question that if the point clouds are captured so close in time, would the model make the same mistake in subsequent scans as it did in the first one? This is a justified question as human perception tends to grasp whole objects in the point cloud. A deep learning model, on the other hand, processes a set of points with unique coordinates. A lidar point-sampling process is stochastic. Even if the lidar is standing still and the scene is static, then the point coordinates of two sequential scans are different. Given that there is also some motion or ego-motion, the point coordinates of sequential scans vary significantly. This proves to be essential in the process and guarantees the max-pooling method to work well.

The effect is similar to a rolling average filter or Kalman filter which are similarly used in smoothing out noise (see Section 2.5.4), but max-pooling does this without the overhead of matrix computations that are necessary for Kalman filters. However, the author acknowledges that experiments here demonstrate that the system is not robust in every case and under all kinds of conditions, there are still many open issues for future research as our tests showed. However, the max-pooling system itself performs well under different conditions.

6.4 Summary

This Chapter proposes a method to validate object detection (and potentially also semantic segmentation) deep learning models.

The collision test method is based on defining a potential collision area, eliminating known objects and marking remaining points as collision points. The method enables to assess object detection and semantic segmentation models for practical collision avoidance purposes. We chose object detection approach to validate the idea. While object detection has a simpler output, it is harder to validate as we need to separately find points that belong to any given object. Semantic segmentation would make it easier and less complex as each point already has a class label attached by the model.

Max-pooled sequential processing was proposed to overcome the limitations that are caused by a sudden and temporary fluctuation in the bounding box estimation quality. This idea is also applicable and useful for real life collision avoidance purposes.

The most important benefit of this method is that the assessment of an object detection model does not require data annotations, i.e., labelling. Also, this chapter showed that the object detection algorithm trained with an adaptive multi-component loss on KITTI [49] dataset (which used Velodyne lidar to collect the data) is working well on real life dataset with a different Ouster lidar. In another words, the deep learning models in this thesis are practically useful. However, when tested in rural winter conditions which are very different from the summertime city-traffic training data, the amount of errors increases. The tool assists in finding problematic scenes which would be an extremely labour-intensive work otherwise as there are tens of thousands of lidar scans for each hour of recording.

The method was evaluated qualitatively. It proved to be a useful tool and helped to detect several different occasions where the object detection model failed. However, also several shortcomings in the collision test were found, e.g., when driving uphill and downhill. Future work should also assess the method on semantic segmentation models.

Chapter 7

Discussion and Conclusions

7.1 Discussion

Collision avoidance is one of the most important goals of any autonomous navigation system. While autonomous driving for passenger cars is perhaps the most known application area, there are numerous other fields that will benefit from it. Some examples include rescue-equipment driving to the accident scene, courier vans servicing remote areas, snow ploughs ready to work as soon as the snow falls, and autonomous military equipment protecting us from barbaric attacks. The topic of this thesis was initially proposed by Oxford Robotics/Dynium.ai, a company that works on autonomous vehicles and smart algorithms for agriculture. Collision avoidance is important for all of these purposes, but unlike in a structured city environment where most passenger cars are driven, many other fields require much more agile and adaptive approach. Tractors on an agricultural field or snow ploughs in the snow storm in a remote rural area cannot rely on a well-defined and structured environment. It becomes increasingly important to design algorithms and deep learning models that are able to adapt to different setups and to unknown data. Hence practical goals such as collision avoidance will prevail over scientific competitions like achieving the highest intersection over union (IoU) value on some benchmark dataset. This thesis took the approach of advancing collision avoidance through expanding scientific research into more practical directions without abandoning the metrics. The research direction grew out of practical experiments that were conducted in Dynium.ai labs in Oxford and Reading, UK, and later in Tallinn, Haapsalu, Vellavere and Vapramäe in Estonia.

Deep learning models for 3D navigation are highly competitive and rapidly developing field of research. However, most of the research is directed towards im-

proving the results of IoU or average precision metric. These metrics measure how precise are the bounding boxes that are output by a deep learning model, and how many objects are detected by the model. Different approaches are compared by looking at high level metrics over all the scenes in a large dataset. The highest granularity that such benchmarks offer is perhaps classifying objects into difficulty categories (e.g., easy, medium and hard as in KITTI [49]) or class-wise (e.g. separate metric value for cars and pedestrians) and providing metric value for each category. It is still an aggregate value over thousands of objects. However, this level of measurement is not sufficient for deploying such deep learning models in practice. While high level metrics are a good overall indicator of the expected performance, deployment in real life requires much finer details. There are important aspects that scientific benchmarks do not measure, but which are essential for collision avoidance and safe autonomous navigation. Some examples are robustness to noise, sensitivity to different weather conditions and ability to work with different sensors (e.g., with different types of lidar).

Last, but not least, most research papers on 3D object detection and semantic segmentation do not even mention how their models perform in collision tests or how they adapt to noisy data. State-of-the-art models (e.g., [156, 181]) rarely disclose in detail their failure cases nor do they disclose any safety issues that may arise from using the models. These are left for the users to discover as it is not the goal of such research. All models and algorithms have their limitations and issues, but models are only deployed safely where such issues are known and hence can be mitigated. This was perhaps one of the most important understandings that evolved over the course of conducting this research. The work in this thesis was aimed at reducing the knowledge gap in this area. Visualisation of the multi-component adaptive loss function output in Section 3.4 enables to observe the trade-offs that the model is making already during the training phase. Model interpretability in Section 4.9.2 enables to understand which parts of the input and of the model affect the results, both good and bad. The idea of interpreting a deep learning model was of course not new, but the way such algorithms were adapted and integrated into 3D navigation research, is novel and contributes to practical collision avoidance.

Collecting and labelling ground truth data and then training a deep learning model for each new scenario is very time consuming and expensive. Even if such resources are of disposal, the model is not guaranteed to work flawlessly or even well enough. Hence, methods that aid in assessing the practical performance of the model are of great importance both for training new models and for evaluating existing ones. The assessment of practical performance has not only practical, but also

a high scientific value. Understanding the errors that the model makes, eventually leads to much better models and also helps in reducing the gap between practical and scientific evaluation criteria. Unified criteria allows to transfer new scientific findings into practice much more confidently. Unsupervised validation of object detection results that was developed in Section 6.2 is a step towards that direction. Existence of such validation methods might eventually also lead to a more widespread transfer learning in 3D point cloud research. Transfer learning allows to take a model trained for one purpose, optionally partly re-train it with a fraction of the cost and time, and use it for another purpose. Affordable and reliable validation is crucial to enable such kind of use.

7.2 Conclusions

This thesis contributed in several ways into advancing deep learning models for point clouds processing, keeping in mind the application of safe navigation and collision avoidance.

Chapter 3 deals with advancing object detection from point clouds. Object detection algorithms are slow to train as their architecture is rather complicated. A novel multi-component loss proposed in this work helps to reduce training time while improves the accuracy of the model.

Chapter 4 proposes a novel sequential point cloud processing model 3D-SEQNET which allows to expand the point classification from 19 classes in the reference model into 27 classes. The extra 8 classes are derived by differentiating static and moving object. For example, a class "car" is split into "car" and "moving-car" where the former represents a static object while the latter represents a moving object. The main contribution is a latent feature fusion method used in the sequential model. The methods reuses latent features of a previous scan to speed up inference time.

Chapter 5 proposes a simple yet effective hardware setup and software architecture for point cloud collection. The test car used in this research was built according to the proposed system design.

Chapter 6 developed an unsupervised validation of object detection results. The method directly assesses a practical performance of a model in a well-defined way. This method is easily adaptable for very different 3D object detection models and purposes. As the analysis in Section 6.2.8 illustrates, unsupervised validation is far from perfect, but still has a very high practical and research value. First, using the method enables to test a model performance on a new data without

having to annotate the data first, thus saving hundreds or even thousands of working hours. Second, it quickly, and with a very low cost, enables the discovery of the weaknesses of a deep learning model for object detection. Analysing the results using this method provides more insight to the model performance than many high level metrics combined. Importantly, it enables the discovery of extreme failure cases of the model should they be present in the data.

7.3 Future work

Deep learning for point cloud processing is a rapidly evolving field of research. Every year, more and more companies also make use of lidar technology as it matures. This thesis demonstrates there are many approaches that are still in an early stage of development. Most current research focuses on achieving state-of-the-art results in academic benchmarks, while practical objectives of safe navigation and collision avoidance might not always directly correlate to the results of such benchmarks. Therefore, the methods developed in this thesis are just the beginning in this research direction. There is plenty of room for further advancements.

Object detection from point clouds could and should be extended to new areas of application. Trying to locate again and again the same sparsely recorded cars from Karlsruhe [49] will likely not advance the research much further. The adaptive multi-component loss proposed in this thesis can be further tested on new data, on new models, and on new types of objects (tractors, farm animals etc).

Sequential semantic segmentation is crucial for capturing the movement. However, our method 3D-SEQNET only fuses the feature spaces of the last two point clouds. There is enormous potential in extending this to a larger number.

Our collision assessment method in Chapter 6 has a solid mathematical foundation, but the output is currently only used for qualitative visual assessment. It can be extended to output numerical evaluation. Also, the current implementation does not use semantic segmentation output (although the system is designed to support it). It should be extended and tested with semantic segmentation output too.

Finally, the most important and well-performing methods arise from empirical testing in real life, from the hands-on approach. There should be much more of it in the 3D point cloud research community!

Bibliography

[1] Agarwal, S., Vora, A., Pandey, G., Williams, W., Kourous, H., McBride, J.: Ford Multi-AV Seasonal Dataset. arXiv:2003.07969 [cs] (Mar 2020), arXiv: 2003.07969

[2] Agrawal, P., Iqbal, A., Russell, B., Hazrati, M.K., Kashyap, V., Akhbari, F.: PCE-SLAM: A real-time simultaneous localization and mapping using LiDAR data. In: 2017 IEEE Intelligent Vehicles Symposium (IV). pp. 1752–1757 (Jun 2017)

[3] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2623–2631. KDD '19, Association for Computing Machinery, New York, NY, USA (Jul 2019)

[4] Aqel, M.O.A., Marhaban, M.H., Saripan, M.I., Ismail, N.B.: Review of visual odometry: types, approaches, challenges, and applications. SpringerPlus **5**(1), 1897 (Dec 2016)

[5] Armeni, I., Sax, S., Zamir, A.R., Savarese, S.: Joint 2D-3D-Semantic Data for Indoor Scene Understanding. arXiv:1702.01105 [cs] (Apr 2017), arXiv: 1702.01105

[6] Asvadi, A., Garrote, L., Premebida, C., Peixoto, P., Nunes, U.J.: DepthCN: Vehicle detection using 3D-LIDAR and ConvNet. In: Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on. pp. 1–6. IEEE (2017)

[7] Asvadi, A., Premebida, C., Peixoto, P., Nunes, U.: 3D Lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes. Robotics and Autonomous Systems **83**, 299–311 (Sep 2016)

[8] Aygun, M., Osep, A., Weber, M., Maximov, M., Stachniss, C., Behley, J., Leal-Taixe, L.: 4D Panoptic LiDAR Segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5527–5537 (2021)

[9] Balestriero, R., Bottou, L., LeCun, Y.: The Effects of Regularization and Data Augmentation are Class Dependent. arXiv:2204.03632 [cs, stat] (Apr 2022), arXiv: 2204.03632

[10] Ball, D., Upcroft, B., Wyeth, G., Corke, P., English, A., Ross, P., Patten, T., Fitch, R., Sukkarieh, S., Bate, A.: Vision-based Obstacle Detection and Navigation for an Agricultural Robot. Journal of Field Robotics **33**(8), 1107–1130 (Dec 2016)

[11] Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software **22**(4), 469–483 (1996)

[12] Bechar, A., Vigneault, C.: Agricultural robots for field operations. Part 2: Operations and systems. Biosystems Engineering **153**, 110–128 (Jan 2017)

[13] Behl, A., Jafari, O.H., Mustikovela, S.K., Alhaija, H.A., Rother, C., Geiger, A.: Bounding Boxes, Segmentations and Object Coordinates: How Important is Recognition for 3D Scene Flow Estimation in Autonomous Driving Scenarios? In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2574–2583 (2017)

[14] Behl, A., Paschalidou, D., Donné, S., Geiger, A.: PointFlowNet: Learning Representations for 3D Scene Flow Estimation from Point Clouds. arXiv:1806.02170 [cs] (Jun 2018), arXiv: 1806.02170

[15] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., Gall, J.: A Dataset for Semantic Segmentation of Point Cloud Sequences. arXiv:1904.01416 [cs] (Apr 2019), arXiv: 1904.01416

[16] Bergerman, M., Maeta, S.M., Zhang, J., Freitas, G.M., Hamner, B., Singh, S., Kantor, G.: Robot Farmers: Autonomous Orchard Vehicles Help Tree Fruit Production. IEEE Robotics Automation Magazine **22**(1), 54–63 (Mar 2015)

[17] Berntorp, K.: Path planning and integrated collision avoidance for autonomous vehicles. In: 2017 American Control Conference (ACC). pp. 4023–4028 (May 2017)

[18] Bijelic, M., Gruber, T., Mannan, F., Kraus, F., Ritter, W., Dietmayer, K., Heide, F.: Seeing through fog without seeing fog: Deep multimodal sensor fusion in unseen adverse weather. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)

[19] Blake, A., Bordallo, A., Hawasly, M., Penkov, S., Ramamoorthy, S., Silva, A.: Efficient Computation of Collision Probabilities for Safe Motion Planning. arXiv preprint arXiv:1804.05384 (2018)

[20] Bresson, G., Alsayed, Z., Yu, L., Glaser, S.: Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving. IEEE Transactions on Intelligent Vehicles **2**(3), 194–220 (2017)

[21] Broggi, A., Cerri, P., Ghidoni, S., Grisleri, P., Jung, H.G.: A New Approach to Urban Pedestrian Detection for Automatic Braking. IEEE Transactions on Intelligent Transportation Systems **10**(4), 594–605 (Dec 2009), conference Name: IEEE Transactions on Intelligent Transportation Systems

[22] Burt, A., Disney, M., Calders, K.: Extracting individual trees from lidar point clouds using treeseg. Methods in Ecology and Evolution **10**(3), 438–445 (Mar 2019)

[23] Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuScenes: A multimodal dataset for autonomous driving. arXiv:1903.11027 [cs, stat] (May 2020), arXiv: 1903.11027

[24] Cao, T., Xiang, Z.Y., Liu, J.L.: Perception in Disparity: An Efficient Navigation Framework for Autonomous Vehicles With Stereo Cameras. IEEE Transactions on Intelligent Transportation Systems **16**(5), 2935–2948 (Oct 2015)

[25] Caraffi, C., Cattani, S., Grisleri, P.: Off-Road Path and Obstacle Detection Using Decision Networks and Stereo Vision. IEEE Transactions on Intelligent Transportation Systems **8**(4), 607–618 (Dec 2007)

[26] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. arXiv:1512.03012 [cs] (Dec 2015), arXiv: 1512.03012

[27] Cheng, R., Razani, R., Taghavi, E., Li, E., Liu, B.: (AF)2-S3Net: Attentive Feature Fusion with Adaptive Feature Selection for Sparse Semantic Segmentation Network. arXiv:2102.04530 [cs] (2021), arXiv: 2102.04530

[28] Choy, C., Gwak, J., Savarese, S.: 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. arXiv:1904.08755 [cs] (Apr 2019), arXiv: 1904.08755

[29] De Gregorio, D., Cavallari, T., Di Stefano, L.: SkiMap++: Real-Time Mapping and Object Recognition for Robotics. In: Computer Vision Workshop (ICCVW), 2017 IEEE International Conference on. pp. 660–668. IEEE (2017)

[30] Deng, H., Birdal, T., Ilic, S.: Ppfnet: Global context aware local features for robust 3d point matching. Computer Vision and Pattern Recognition (CVPR). IEEE **1** (2018)

[31] Dhamdhere, K., Sundararajan, M., Yan, Q.: How Important Is a Neuron? arXiv (May 2018), arXiv:1805.12233 [cs, stat]

[32] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs] (Oct 2020), arXiv: 2010.11929

[33] Drusch, M., Del Bello, U., Carlier, S., Colin, O., Fernandez, V., Gascon, F., Hoersch, B., Isola, C., Laberinti, P., Martimort, P., Meygret, A., Spoto, F., Sy, O., Marchese, F., Bargellini, P.: Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services. Remote Sensing of Environment **120**, 25–36 (May 2012)

[34] Dubé, R., Gawel, A., Sommer, H., Nieto, J., Siegwart, R., Cadena, C.: An online multi-robot SLAM system for 3D LiDARs. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1004–1011 (Sep 2017)

[35] Dubé, R., Cramariuc, A., Dugas, D., Nieto, J., Siegwart, R., Cadena, C.: SegMap: 3D Segment Mapping using Data-Driven Descriptors. arXiv:1804.09557 (Apr 2018)

[36] Eriksen, B.H., Wilthil, E.F., Flåten, A.L., Brekke, E.F., Breivik, M.: Radar-based maritime collision avoidance using dynamic window. In: 2018 IEEE Aerospace Conference. pp. 1–9 (Mar 2018)

[37] Everingham, M., Eslami, S.A., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International journal of computer vision **111**(1), 98–136 (2015)

[38] Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J., Zisserman, A.: The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision **88**(2), 303–338 (Jun 2010)

[39] Fan, H., Yang, Y.: PointRNN: Point Recurrent Neural Network for Moving Point Cloud Processing. arXiv:1910.08287 [cs] (Nov 2019), arXiv: 1910.08287

[40] Fan, H., Yang, Y., Kankanhalli, M.: Point 4D Transformer Networks for Spatio-Temporal Modeling in Point Cloud Videos. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14204–14213 (2021)

[41] Fidler, S., Dickinson, S., Urtasun, R.: 3D Object Detection and Viewpoint Estimation with a Deformable 3D Cuboid Model. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 611–619. Curran Associates, Inc. (2012)

[42] Florence, P.R., Carter, J., Ware, J., Tedrake, R.: Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. arXiv preprint arXiv:1802.09076 (2018)

[43] Fong, W.K., Mohan, R., Hurtado, J.V., Zhou, L., Caesar, H., Beijbom, O., Valada, A.: Panoptic nuScenes: A Large-Scale Benchmark for LiDAR Panoptic Segmentation and Tracking. arXiv:2109.03805 [cs] (Dec 2021), arXiv: 2109.03805

[44] Fridman, L.: MIT Self-Driving Cars: Sacha Arnoud, Director of Engineering, Waymo (2018), https://www.youtube.com/watch?v=LSX3qdy0dFg

[45] Fu, X., Huang, J., Zeng, D., Huang, Y., Ding, X., Paisley, J.: Removing rain from single images via a deep detail network. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)

[46] Funke, J., Brown, M., Erlien, S.M., Gerdes, J.C.: Collision Avoidance and Stabilization for Autonomous Vehicles in Emergency Scenarios. IEEE Transactions on Control Systems Technology **25**(4), 1204–1216 (Jul 2017)

[47] Gal, Y., Ghahramani, Z.: Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In: International Conference on Machine Learning. pp. 1050–1059 (Jun 2016)

[48] Gal, Y., Smith, L.: Idealised Bayesian Neural Networks Cannot Have Adversarial Examples: Theoretical and Empirical Study. arXiv:1806.00667 [cs, stat] (Jun 2018), arXiv: 1806.00667

[49] Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research **32**(11), 1231–1237 (Sep 2013)

[50] Geyer, J., Kassahun, Y., Mahmudi, M., Ricou, X., Durgesh, R., Chung, A.S., Hauswald, L., Pham, V.H., Mühlegg, M., Dorn, S., Fernandez, T., Jänicke, M., Mirashi, S., Savani, C., Sturm, M., Vorobiov, O., Oelker, M., Garreis, S., Schuberth, P.: A2D2: Audi Autonomous Driving Dataset. arXiv.2004.06320 [cs] (2020)

[51] Ghahremani, M., Tiddeman, B., Liu, Y., Behera, A.: Orderly Disorder in Point Cloud Domain. arXiv:2008.09634 [cs] (Aug 2020), arXiv: 2008.09634

[52] Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1440–1448 (2015)

[53] Glennie, C.L., Kusari, A., Facchin, A.: Calibration and stability analysis of the VLP-16 laser scanner. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences (2016)

[54] Godard, C., Mac Aodha, O., Brostow, G.J.: Unsupervised monocular depth estimation with left-right consistency. In: CVPR. vol. 2, p. 7 (2017)

[55] Graham, B.: Spatially sparse convolutional neural networks. arXiv (Sep 2014), arXiv:1409.6070 [cs]

[56] Graham, B., Engelcke, M., van der Maaten, L.: 3D Semantic Segmentation With Submanifold Sparse Convolutional Networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9224–9232 (2018)

[57] HE, J., Erfani, S., Ma, X., Bailey, J., Chi, Y., Hua, X.S.: alpha-IoU: A Family of Power Intersection over Union Losses for Bounding Box Regression. In: Advances in Neural Information Processing Systems. vol. 34, pp. 20230–20242. Curran Associates, Inc. (2021)

[58] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)

[59] Hesai, AI, S.: Pandaset dataset (2021), pandaset.org

[60] Hiremath, S., van Evert, F.K., Braak, C.t., Stein, A., van der Heijden, G.: Image-based particle filtering for navigation in a semi-structured agricultural environment. Biosystems Engineering **121**, 85–95 (May 2014)

[61] Ho, K.L., Newman, P.: Loop closure detection in SLAM by combining visual and spatial appearance. Robotics and Autonomous Systems **54**(9), 740–749 (2006)

[62] Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (Nov 1997)

[63] Hong, W., Yuan, J., Das Bhattacharjee, S.: Fried Binary Embedding for High-Dimensional Visual Features. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2749–2757 (2017)

[64] Horn, M., Engel, N., Belagiannis, V., Buchholz, M., Dietmayer, K.: DeepCLR: Correspondence-Less Architecture for Deep End-to-End Point Cloud Registration. arXiv:2007.11255 [cs] (Jul 2020), arXiv: 2007.11255

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

[66] Huang, R., Zhang, W., Kundu, A., Pantofaru, C., Ross, D.A., Funkhouser, T., Fathi, A.: An LSTM Approach to Temporal 3D Object Detection in LiDAR Point Clouds. arXiv:2007.12392 [cs, eess] (Jul 2020), arXiv: 2007.12392

[67] Ito, E., Okatani, T.: Self-calibration-based approach to critical motion sequences of rolling-shutter structure from motion. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. vol. 2 (2017)

[68] Jiang, T.X., Huang, T.Z., Zhao, X.L., Deng, L.J., Wang, Y.: A novel tensor-based video rain streaks removal approach via utilizing discriminatively intrinsic priors. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'17) (2017)

[69] Kahn, G., Villaflor, A., Ding, B., Abbeel, P., Levine, S.: Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. arXiv:1709.10489 [cs] (Sep 2017), arXiv: 1709.10489

[70] Kahn, G., Villaflor, A., Pong, V., Abbeel, P., Levine, S.: Uncertainty-Aware Reinforcement Learning for Collision Avoidance. arXiv:1702.01182 [cs] (Feb 2017), arXiv: 1702.01182

[71] Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., Happold, M., Herman, H., Mandelbaum, R., Pilarski, T., Rander, P., Thayer, S., Vallidis, N., Warner, R.: Toward Reliable Off Road Autonomous Vehicles Operating in Challenging Environments. The International Journal of Robotics Research **25**(5-6), 449–483 (May 2006)

[72] Kendall, A., Gal, Y.: What uncertainties do we need in bayesian deep learning for computer vision? In: Advances in neural information processing systems. pp. 5574–5584 (2017)

[73] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.M., Lam, V.D., Bewley, A., Shah, A.: Learning to Drive in a Day. arXiv:1807.00412 [cs, stat] (Jul 2018), arXiv: 1807.00412

[74] Keselman, L., Iselin Woodfill, J., Grunnet-Jepsen, A., Bhowmik, A.: Intel RealSense Stereoscopic Depth Cameras. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 1–10 (2017)

[75] Kornblith, S., Chen, T., Lee, H., Norouzi, M.: Why Do Better Loss Functions Lead to Less Transferable Features? In: Advances in Neural Information Processing Systems. vol. 34, pp. 28648–28662. Curran Associates, Inc. (2021)

[76] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012)

[77] Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.L.: Joint 3D Proposal Generation and Object Detection from View Aggregation. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1–8 (Oct 2018), iSSN: 2153-0866

[78] Kurup, A., Bos, J.: Winter adverse dataset (wads) (2021), `digitalcommons.mtu.edu/wads/`

[79] Lai, X., Liu, J., Jiang, L., Wang, L., Zhao, H., Liu, S., Qi, X., Jia, J.: Stratified transformer for 3d point cloud segmentation. In: Proceedings of

the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8500–8509 (2022)

[80] Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: PointPillars: Fast Encoders for Object Detection from Point Clouds. arXiv:1812.05784 (Dec 2018)

[81] LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10), 1995 (1995)

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

[83] Lee, U., Reynolds, T., Barzgaran, B., Badyn, M.H., Chrisope, J., Adler, A., Kaycee, K., Mesbahi, M.: Development of Attitude Determination and Control Subsystem for 3U CubeSat with Electric Propulsion. In: AIAA SPACE and Astronautics Forum and Exposition. AIAA SPACE Forum, American Institute of Aeronautics and Astronautics (Sep 2017)

[84] Lenac, K., Kitanov, A., Cupec, R., Petrović, I.: Fast planar surface 3D SLAM using LIDAR. Robotics and Autonomous Systems **92**, 197–220 (Jun 2017)

[85] Levin, S.: Video released of Uber self-driving crash that killed woman in Arizona. The Guardian (Mar 2018)

[86] Levinson, J., Thrun, S.: Automatic Online Calibration of Cameras and Lasers. In: Robotics: Science and Systems. vol. 2 (2013)

[87] Li, T., Fang, J., Zhong, Y., Wang, D., Xue, J.: Online High-Accurate Calibration of RGB+3D-LiDAR for Autonomous Driving. In: Image and Graphics. pp. 254–263. Lecture Notes in Computer Science, Springer, Cham (Sep 2017)

[88] Liao, Y., Xie, J., Geiger, A.: KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D. In: IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI) (2022)

[89] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: A Research Platform for Distributed Model Selection and Training (Jul 2018). https://doi.org/10.48550/arXiv.1807.05118, arXiv:1807.05118 [cs, stat]

[90] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollar, P.: Focal Loss for Dense Object Detection. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2980–2988 (2017)

[91] Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft COCO: Common Objects in Context. arXiv:1405.0312 [cs] (Feb 2015), arXiv: 1405.0312

[92] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single Shot MultiBox Detector. In: Computer Vision – ECCV 2016. pp. 21–37. Lecture Notes in Computer Science, Springer, Cham (Oct 2016)

[93] Liu, X., Qi, C.R., Guibas, L.J.: FlowNet3D: Learning Scene Flow in 3D Point Clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 529–537 (2019)

[94] Liu, X., Yan, M., Bohg, J.: MeteorNet: Deep Learning on Dynamic 3D Point Cloud Sequences. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9246–9255 (2019)

[95] Lu, J., Hrovat, D., Tseng, H.: Off-road autonomous driving (May 2016)

[96] Lu, W., Wan, G., Zhou, Y., Fu, X., Yuan, P., Song, S.: DeepICP: An End-to-End Deep Neural Network for 3D Point Cloud Registration. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 12–21 (Oct 2019), arXiv: 1905.04153

[97] Luettel, T., Himmelsbach, M., Wuensche, H.J.: Autonomous Ground Vehicles - Concepts and a Path to the Future. Proceedings of the IEEE **100**(Special Centennial Issue), 1831–1839 (May 2012)

[98] Luo, W., Li, Y., Urtasun, R., Zemel, R.: Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. arXiv:1701.04128 [cs] (Jan 2017), arXiv: 1701.04128

[99] Maddern, W., Pascoe, G., Linegar, C., Newman, P.: 1 year, 1000 km: The Oxford RobotCar dataset. The International Journal of Robotics Research **36**(1), 3–15 (Jan 2017)

[100] Mancini, M., Costante, G., Valigi, P., Ciarfuglia, T.A., Delmerico, J., Scaramuzza, D.: Toward Domain Independence for Learning-Based Monocular Depth Estimation. IEEE Robotics and Automation Letters **2**(3), 1778–1785 (Jul 2017)

[101] Maqueda, A.I., Loquercio, A., Gallego, G., Garcia, N., Scaramuzza, D.: Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars. arXiv:1804.01310 [cs] (Apr 2018), arXiv: 1804.01310

[102] Maturana, D., Scherer, S.: 3d convolutional neural networks for landing zone detection from lidar. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on. pp. 3471–3478. IEEE (2015)

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

[104] McCrae, S., Zakhor, A.: 3d Object Detection For Autonomous Driving Using Temporal Lidar Data. In: 2020 IEEE International Conference on Image Processing (ICIP). pp. 2661–2665 (Oct 2020), iSSN: 2381-8549

[105] Mendes, E., Koch, P., Lacroix, S.: ICP-based pose-graph SLAM. In: 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). pp. 195–200 (Oct 2016)

[106] Milella, A., Reina, G.: 3D reconstruction and classification of natural environments by an autonomous vehicle using multi-baseline stereo. Intelligent Service Robotics **7**(2), 79–92 (2014)

[107] Moeys, D.P., Corradi, F., Kerr, E., Vance, P., Das, G., Neil, D., Kerr, D., Delbrück, T.: Steering a predator robot using a mixed frame/event-driven convolutional neural network. In: 2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP). pp. 1–8 (Jun 2016)

[108] Mousazadeh, H.: A technical review on navigation systems of agricultural autonomous off-road vehicles. Journal of Terramechanics **50**(3), 211–232 (Jun 2013)

[109] Olah, C., Mordvintsev, A., Schubert, L.: Feature Visualization. Distill **2**(11), e7 (Nov 2017)

[110] Oleynikova, H., Millane, A., Taylor, Z., Galceran, E., Nieto, J., Siegwart, R.: Signed distance fields: A natural representation for both mapping and planning. In: RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics. University of Michigan (2016)

[111] Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., Nieto, J.: Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. pp. 1366–1373. IEEE (2017)

[112] Oleynikova, H., Taylor, Z., Siegwart, R., Nieto, J.: Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning. arXiv preprint arXiv:1803.04345 (2018)

[113] Ollis, M., Huang, W.H., Happold, M., Stancil, B.A.: Image-based path planning for outdoor mobile robots. In: 2008 IEEE International Conference on Robotics and Automation. pp. 2723–2728 (May 2008)

[114] Ort, T., Paull, L., Rus, D.: Autonomous Vehicle Navigation in Rural Environments Without Detailed Prior Maps. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 2040–2047 (May 2018)

[115] Ouster, I.: OS1 Hardware User Manual (Dec 2020)

[116] Ouster, I.: OS1 Gen 1 datasheet (Dec 2021)

[117] Ouster, I.: Ouster OS2 datasheet (2022)

[118] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)

[119] Pereira, M., Silva, D., Santos, V., Dias, P.: Self calibration of multiple LIDARs and cameras on autonomous vehicles. Robotics and Autonomous Systems **83**, 326–337 (Sep 2016)

[120] Pomerleau, F., Colas, F., Siegwart, R.: A Review of Point Cloud Registration Algorithms for Mobile Robotics. Foundations and Trends in Robotics **4**(1), 1–104 (May 2015)

[121] Pomerleau, F., Colas, F., Siegwart, R., Magnenat, S.: Comparing ICP variants on real-world data sets. Autonomous Robots **34**(3), 133–148 (2013)

[122] Premebida, C., Ludwig, O., Nunes, U.: LIDAR and vision-based pedestrian detection system. Journal of Field Robotics **26**(9), 696–711 (2009), _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20312

[123] Provodin, A., Torabi, L., Flepp, B., LeCun, Y., Sergio, M., Jackel, L.D., Muller, U., Zbontar, J.: Fast Incremental Learning for Off-Road Robot Navigation. arXiv preprint arXiv:1606.08057 (2016)

[124] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE **1**(2), 4 (2017)

[125] Qi, C.R., Su, H., Niessner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and Multi-View CNNs for Object Classification on 3D Data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5648–5656 (2016)

[126] Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)

[127] Qu, L., Tian, J., He, S., Tang, Y., Lau, R.W.: Deshadownet: A multi-context embedding deep network for shadow removal. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4067–4075 (2017)

[128] Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767 (2018)

[129] Reina, G., Galati, R., Milella, A.: All-terrain estimation for mobile robots in precision agriculture. In: 2018 IEEE International Conference on Industrial Technology (ICIT). pp. 63–68 (Feb 2018)

[130] Reina, G., Milella, A., Worst, R.: LIDAR and stereo combination for traversability assessment of off-road robotic vehicles. Robotica **34**(12), 2823–2841 (Dec 2016)

[131] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 91–99. Curran Associates, Inc. (2015)

[132] Ren, Z., Sudderth, E.B.: Three-Dimensional Object Detection and Layout Prediction Using Clouds of Oriented Gradients. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1525–1533 (Jun 2016)

[133] Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. pp. 234–241. Lecture Notes in Computer Science, Springer International Publishing (2015)

[134] Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: 2011 IEEE International Conference on Robotics and Automation. pp. 1–4 (May 2011)

[135] S., T.: Why Uber's self-driving car killed a pedestrian - The Economist explains. The Economist (May 2018)

[136] Scheiner, N., Kraus, F., Wei, F., Phan, B., Mannan, F., Appenrodt, N., Ritter, W., Dickmann, J., Dietmayer, K., Sick, B., Heide, F.: Seeing Around Street Corners: Non-Line-of-Sight Detection and Tracking In-the-Wild Using Doppler Radar. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2068–2077 (2020)

[137] Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4104–4113 (2016)

[138] Scona, R., Jaimez, M., Petillot, Y.R., Fallon, M., Cremers, D.: StaticFusion: background reconstruction for dense RGB-Dd slam in dynamic environments. In: 2018 IEEE International Conference on Robotics and Automation. Institute of Electrical and Electronics Engineers (2018)

[139] Seiler, P., Song, B., Hedrick, J.K.: Development of a Collision Avoidance System. SAE Transactions **107**, 1334–1340 (1998), publisher: SAE International

[140] Shalal, N., Low, T., McCarthy, C., Hancock, N.: Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion – Part B: Mapping and localisation. Computers and Electronics in Agriculture **119**, 267–278 (Nov 2015)

[141] Shan, T., Englot, B.: LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4758–4765. IEEE (2018)

[142] Shi, H., Lin, G., Wang, H., Hung, T.Y., Wang, Z.: SpSequenceNet: Semantic Segmentation Network on 4D Point Clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4574–4583 (2020)

[143] Shi, S., Wang, X., Li, H.: PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. arXiv (Dec 2018)

[144] Shibuya, M., Sumikura, S., Sakurada, K.: Privacy Preserving Visual SLAM. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) Computer Vision – ECCV 2020. pp. 102–118. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020)

[145] Silberman, N., Hoiem, D., Kohli, P., Fergus, R.: Indoor segmentation and support inference from RGBD images. In: Proceedings of the 12th European conference on Computer Vision. ECCV 12, vol. 5, pp. 746–760. Springer Verlag, Berlin, Heidelberg (Oct 2012)

[146] Simon, M., Milz, S., Amende, K., Groß, H.M.: Complex-YOLO: Real-time 3D Object Detection on Point Clouds (2018)

[147] Smolyanskiy, N., Kamenev, A., Birchfield, S.: On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach. arXiv preprint arXiv:1803.09719 (2018)

[148] Song, S., Lichtenberg, S.P., Xiao, J.: SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 567–576 (2015)

[149] Song, X., Wang, P., Zhou, D., Zhu, R., Guan, C., Dai, Y., Su, H., Li, H., Yang, R.: ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5452–5462 (2019)

[150] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi,

A., Zhao, S., Cheng, S., Zhang, Y., Shlens, J., Chen, Z., Anguelov, D.: Scalability in Perception for Autonomous Driving: Waymo Open Dataset. arXiv:arXiv.1912.04838 [cs] (2019)

[151] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic Attribution for Deep Networks. In: Proceedings of the 34th International Conference on Machine Learning. pp. 3319–3328. PMLR (Jul 2017), iSSN: 2640-3498

[152] Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning. pp. 1139–1147. PMLR (May 2013), iSSN: 1938-7228

[153] Szarvas, M., Sakai, U., Ogata, J.: Real-time pedestrian detection using LIDAR and convolutional neural networks. In: Intelligent Vehicles Symposium, 2006 IEEE. pp. 213–218. IEEE (2006)

[154] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)

[155] Tan, W., Qin, N., Ma, L., Li, Y., Du, J., Cai, G., Yang, K., Li, J.: Toronto-3d: A large-scale mobile lidar dataset for semantic segmentation of urban roadways. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (2020)

[156] Tang, H., Liu, Z., Zhao, S., Lin, Y., Lin, J., Wang, H., Han, S.: Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In: Computer Vision – ECCV 2020. pp. 685–702. Springer International Publishing (2020)

[157] Taniai, T., Sinha, S.N., Sato, Y.: Fast Multi-Frame Stereo Scene Flow With Motion Segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2017)

[158] Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient Transformers: A Survey (2022), http://arxiv.org/abs/2009.06732, arXiv:2009.06732 [cs]

[159] Tian, D., Zou, F., Xu, F., Di, P.: 3D Laser Odometry for a Mobile Robot Platform. In: 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER). pp. 423–426 (Jul 2017)

[160] Ugenti, A., Vulpi, F., Milella, A., Reina, G.: Learning and prediction of vehicle-terrain interaction from 3D vision. In: Multimodal Sensing and Artificial Intelligence: Technologies and Applications II. vol. 11785, pp. 167–173. SPIE (Jun 2021)

[161] Ullman, S.: The Interpretation of Structure from Motion. Proceedings of the Royal Society of London. Series B, Biological Sciences **203**(1153), 405–426 (1979)

[162] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)

[163] Vougioukas, S.G., He, L., Arikapudi, R.: Orchard worker localisation relative to a vehicle using radio ranging and trilateration. Biosystems Engineering **147**, 1–16 (Jul 2016)

[164] Wang, C.Y., Yeh, I.H., Liao, H.Y.M.: You Only Learn One Representation: Unified Network for Multiple Tasks. arXiv:2105.04206 [cs] (May 2021), arXiv: 2105.04206

[165] Wang, J., Chen, K., Yang, S., Loy, C.C., Lin, D.: Region Proposal by Guided Anchoring. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2965–2974 (2019)

[166] Wang, L., Huang, Y., Hou, Y., Zhang, S., Shan, J.: Graph Attention Convolution for Point Cloud Semantic Segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10296–10305 (2019)

[167] Wang, Z., Li, G., Jiang, H., Chen, Q., Zhang, H.: Collision-free Navigation of Autonomous Vehicles using Convex Quadratic Programming-based Model Predictive Control. IEEE/ASME Transactions on Mechatronics pp. 1–1 (2018)

[168] Wang, Z., Jia, K.: Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection. arXiv (Mar 2019)

[169] Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., Pan, B., Kumar, R., Hartnett, A., Kaesemodel Pontes, J., Ramanan, D., Carr, P., Hays, J.: Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting. Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks **1** (2021)

[170] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D ShapeNets: A Deep Representation for Volumetric Shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1912–1920 (2015)

[171] Wulff, J., Sevilla-Lara, L., Black, M.J.: Optical flow in mostly rigid scenes. arXiv preprint arXiv:1705.01352 (2017)

[172] Yan, Y., Mao, Y., Li, B.: SECOND: Sparsely Embedded Convolutional Detection. Sensors **18**(10), 3337 (Oct 2018)

[173] Yang, B., Luo, W., Urtasun, R.: PIXOR: Real-Time 3D Object Detection From Point Clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7652–7660 (2018)

[174] Yang, W., Tan, R.T., Feng, J., Liu, J., Guo, Z., Yan, S.: Deep joint rain detection and removal from a single image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1357–1366 (2017)

[175] Yin, J., Shen, J., Guan, C., Zhou, D., Yang, R.: LiDAR-Based Online 3D Video Object Detection With Graph-Based Message Passing and Spatiotemporal Transformer Attention. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11495–11504 (2020)

[176] Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014. pp. 818–833. Lecture Notes in Computer Science, Springer International Publishing, Cham (2014)

[177] Zeng, Y., Hu, Y., Liu, S., Ye, J., Han, Y., Li, X., Sun, N.: RT3D: Real-Time 3-D Vehicle Detection in LiDAR Point Cloud for Autonomous Driving. IEEE Robotics and Automation Letters **3**(4), 3434–3440 (Oct 2018)

[178] Zhang, J., Singh, S.: Visual-lidar odometry and mapping: low-drift, robust, and fast. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 2174–2181 (May 2015)

[179] Zhang, J., Singh, S.: LOAM: Lidar Odometry and Mapping in Real-time. In: Robotics: Science and Systems X. vol. 10 (Jul 2014)

[180] Zhang, X., Liniger, A., Borrelli, F.: Optimization-Based Collision Avoidance. arXiv:1711.03449 [cs, math] (Nov 2017), arXiv: 1711.03449

[181] Zhang, Y., Hare, J., Prügel-Bennett, A.: Deep Set Prediction Networks. arXiv:1906.06565 [cs, stat] (Jun 2019), arXiv: 1906.06565

[182] Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P.H.S., Zhang, L.: Rethinking Semantic Segmentation From a Sequence-to-Sequence Perspective With Transformers. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6881–6890 (2021)

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

[184] Zhu, A., Atanasov, N., Daniilidis, K.: Event-based visual inertial odometry. In: Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog. vol. 3 (2017)