Dynamic object detection with camera and laser scanner data for autonomous outdoor mobile robotics

Turkka Helin

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 18.12.2019

Supervisor

Prof. Ville Kyrki

Advisor

M.Sc. (Tech.) Samuli Peurasari
**Author** Turkka Helin  
**Title** Dynamic object detection with camera and laser scanner data for autonomous outdoor mobile robotics  
**Degree programme** Automation and Electrical Engineering  
**Major** Control, Robotics and Autonomous Systems  
**Code of major** ELEC3025  
**Supervisor** Prof. Ville Kyrki  
**Advisor** M.Sc. (Tech.) Samuli Peurasraari  
**Date** 18.12.2019  
**Number of pages** 64 + 8  
**Language** English  

**Abstract**

Autonomous mobile robots require an accurate perception of the environment in order to operate safely and reliably in dynamic environments. Safe operation requires predictive estimation of the state of the environment in which dynamic object detection is paramount. Recent advances in neural networks have improved the accuracy of the state-of-the-art general object detectors significantly. However, they are mostly utilized in autonomous driving research.

This work studies dynamic object detection in autonomous mobile robotic applications. In this work, a multi-modal dynamic object detector was implemented for an outdoor industrial logistic robot. The detector fuses state-of-the-art camera detector and segmentation of point cloud data from a light detection and ranging sensor. The detector was evaluated against an open dataset on performance, accuracy and robustness criteria and compared to a state-of-the-art multi-modal object detector. The detector produced satisfactory results for the application requirements although the results were not on par with the state-of-the-art.

**Keywords** camera, LiDAR, multi-modal, convolutional neural networks, object detection
Tekijä  Turkka Helin
Työn nimi  Liikkuvien kohteiden tunnistus kamera- ja laserkeilainaineistosta autonomisia ulkoilmamobiilirobootteja varten
Koulutusohjelma  Automaatio ja sähkötekniikka
Päätäne  Säättöteknika, robotiikka ja autonomiset järjestelmät  Päätäneen koodi  ELEC3025
Työn valvoja  Prof. Ville Kyrki
Työn ohjaaja  DI Samuli Peurasaari
Päivämäärä  18.12.2019  Sivumäärä  64 + 8  Kieli  Englanti

Tiivistelmä

Tässä työssä tutkitaan dynaamisten kohteiden tunnistusta autonomisissa mobiilirobotiikkasovelluksissa. Työssä kohdeltiin monituloinen tunnistusmenetelmä ulkoilmalogistiikkarobottia varten, joka yhdistää viimeisintä tekniiikkaa edustavan kamerapohjaisen tunnistimen ja laserkeilaimen pistepilviaineiston segmentaation. Tunnistinta arvioitiin sen suoritusajan, tarkkuuden ja robustisuuden mukaan avoimella aineistolla ja sitä verrattiin viimeisintä tekniikkaa edustavaan monituloiseen kohteiden tunnistimeen. Tunnistin saavutti riittävät tulokset sovellusta varten, mutta ei saavuttanut viimeisintä tekniikkaa edustavan tunnistimen tasoa.

Avainsanat  kamera, laserkeilain, konvoluutioneuroverkot, kohteiden tunnistus
Preface

This work was done at GIM Ltd., the most groundbreaking and inspiring company that I have ever worked in. I want to thank all my colleagues for all the help and advice I have received during this work and my time at the company. Special thanks to my instructor M.Sc. Samuli Peurasaari for his guidance and mentoring. I want to also thank my supervisor Prof. Ville Kyrki for his ideas and feedback of my work.

I want to also thank all my family and friends who have supported me during my studies. I want to thank all my friends in FTMK’15 & 16, TJ’16 & ’17, SIKH’15, ITMK’14, HTMK’013 and all the other volunteers in SIK & AYY I’ve had the pleasure to work with.

Lastly, I want to thank my beloved Recta, whose never-ending cheerfulness has supported and motivated me throughout my work and studies.

Otaniemi, 18.12.2019

Turkka Helin
# Contents

Abstract

Abstract (in Finnish)

Preface

Contents

Abbreviations and Acronyms

1 Introduction

2 Background
   2.1 Camera based detection
   2.2 LiDAR based detection
      2.2.1 Foreground segmentation
      2.2.2 Machine learning methods
   2.3 Multi-modal detection
      2.3.1 Sensor Fusion
      2.3.2 Multi-modal object detection methods

3 Methods
   3.1 System description
   3.2 LiDAR foreground segmentation
      3.2.1 Ground removal
      3.2.2 Object clustering
   3.3 Camera detector
   3.4 Sensor Fusion

4 Results
   4.1 Experiments
   4.2 Dataset
   4.3 Parameters
   4.4 Detection results
   4.5 Evaluation of the results
   4.6 Computational performance evaluation

5 Conclusions

References

A Detection results
Abbreviations and Acronyms

Abbreviations

BEV  Bird's-eye View
BFS  Breadth-first Search
CNN  Convolutional Neural Network
CPM  Camera-plane Map
CPU  Central Processing Unit
DATMO Detection and Tracking of Moving Objects
DCNN Deep Convolutional Neural Network
DBSCAN Density-based Spatial Clustering of Applications with Noise
DNN  Deep Neural Network
DoF  Degrees of Freedom
EKF  Extended Kalman Filter
FOV  Field of View
FPS  Frames Per Second
GIF Network Gated Information Fusion Network
GNN  Global Nearest Neighbour
GNSS Global Navigation Satellite System
GP-INSAC Gaussian Process Incremental Sample Consensus
GPU  Graphics Processing Unit
HHA  Horizontal disparity, Height above ground, and Angle
HOG  Histogram of Oriented Gradients
INSAC Incremental Sample Consensus
IOU  Intersection-over-union
LiDAR Light Detection and Ranging
MV3D Multi-View 3D Network
NN  Neural Network
OFT  Orthographic Feature Transform
PCA  Principal Component Analysis
RAM  Random Access Memory
RANSAC Random Sample Consensus
RCNN Regions with Convolutional Neural Network features
RGBD Red-Green-Blue-Depth
RMSE Root-Mean-Square Error
RNN Recurrent Neural Network
RoI Region of Interest
RPN Region Proposal Network
SVM Support Vector Machine
YOLO You Only Look Once
1 Introduction

Autonomous mobile robots require robust and precise perception of their surroundings to be able to navigate autonomously in dynamic environments. Robots need information of static obstacles such as walls, as well as information about dynamic objects such as pedestrians, to avoid collisions. Obstacle detection is a crucial part of any autonomous mobile robot, especially when operating in a real-world environment with humans and other vehicles where a robot in the worst case can be even harmful. Obstacle detection poses the question, if the robot is about to collide into anything if the current trajectory is continued. However, reactive obstacle detection can cause the vehicle movement to be abrupt if used only by itself. Hence, detection and tracking of moving objects (DATMO) is required.

DATMO systems consist of two parts: dynamic object detection and target tracking. Object detection is used to identify distinct objects in the vicinity of the robot. This is used as an input for target tracking to estimate future positions of the detected objects. Target tracking and state estimation is required to make the navigation smooth and robust as the robot is able to control its trajectory predictively instead of reactively. This is especially important when traversing at higher speeds, since there has to be enough time and distance for braking. Thus, dynamic object detection is a critical part of successful autonomous navigation, since accurate detections enable target tracking to estimate smooth trajectories, which in turn helps with the navigation and path planning of the robot.

Recent advances in neural networks (NN) have made them the focus of computer vision research. Generic object detectors have had significant improvements in less than a decade due to advancements in convolutional neural networks (CNN). Generic object detectors have been an especially interesting field of research for autonomous driving since accurate and reliable perception of the environment is paramount in day-to-day traffic. Currently there are multiple open datasets for autonomous driving research containing data from different traffic scenarios. Most of the research and datasets are focused on image-based data, but there are more and more light detection and ranging (LiDAR) and radar data included. Sensor fusion of multiple sensing modalities has been of special interest in the most recent research, as combined modalities provide a more comprehensive perception of the environment. However, as the research has focused mostly on autonomous driving, other domains of mobile robotics are lagging behind, mostly due to the lack of large enough datasets.

In this work a camera and LiDAR based dynamic object detector was implemented for an autonomous mobile robot for industrial logistics applications. This work was done as a case study, which evaluates how dynamic objects can be detected in a reliable way while being efficient for robotic applications. Furthermore, the detector must be able to run on the embedded hardware of the robot. This work studies how accurately dynamic objects can be detected given the restrictions imposed by the hardware of the robot in the given domain, as this is critical in order to estimate the safety and reliability of the robot.

The robot is operating in warehouse and stockyard environments, which imposes restrictions on the methods used in the detector such as that the detector must
be robust to different environmental conditions such as weather and illumination. Furthermore, the objects required to be detected range from pedestrians to forklifts and trucks. Moreover, the system is required to fulfill soft real-time requirements, since the objects moving in the environment can move with fast pace and the robot must be able to react fast to changing surroundings.

The robot uses camera and LiDAR sensors, because of the modalities the sensors complement each other. The camera gives accurate 2D color and shape information, which can be used to identify objects reliably. However, it does not give distance information, which is crucial as the robot is required to have an accurate 3D spatial perception of the environment in order to operate safely and reliably. On the other hand, a 3D LiDAR gives accurate distance and shape information, but it is sparser and it does not contain color information. Hence, the sensor fusion of these two sensors produces data that can be used for estimation of the 3D spatial location and size as well as accurate classification of the object. Furthermore, cameras and LiDARs have been used in state-of-the-art object detectors and literature successfully in multiple domains.

The work is evaluated against an open road dataset with ground truth labels given as the robot is not yet in operation during this work and data from the domain of the robot was not available. Hence, this work studies how accurate dynamic object detector can be made given the restrictions imposed by the hardware and the domain of the robot. The detections are evaluated on the detection accuracy and the geometric accuracy of the detections. Detection accuracy is the measure of how reliably a detection is found and the geometric accuracy is the measure of how well the 3D spatial location and size are estimated. Furthermore, the work is evaluated by the computational performance of the detector.

The content of the thesis is structured as follows: In Section 2, the related work in camera and LiDAR object detection as well as multi-sensor object detection is discussed. Section 3 describes the robot platform that this work is designed to, the methods used in this work, the experiments and the data used for testing the system. In Section 4, the results of this work are described and the evaluation of the results is given. In Section 5, conclusions and the future work are discussed.
2 Background

Object detection is the act of identifying and localizing distinct objects from measurement data [15]. This is usually presented by extracting boundary information of the object such as a bounding box. Depending on the sensor modalities, the bounding boxes can be in 2D or 3D, although 2D bounding boxes are mostly used for camera-only object detection. Furthermore, 3D bounding boxes are favored in robotics, since they are more useful in 3D navigation, as they contain the 3D spatial location information. The boundary information can be presented with much tighter fitting as done in semantic segmentation such as [43], which means classification of each pixel in the image to a object class [15]. Object detection on the other hand, refers to localizing the object with a bounding box. A robust bounding box is often enough, since it contains the position and size estimate of the object. Datasets such as nuScenes [5] and KITTI [19] contain annotated 3D bounding box information for autonomous driving research.

Object detection and classification are closely related, since the class, size and shape of an object are tightly connected. Even though dynamic object detection for tracking purposes is the focus of many detection systems, it is often beneficial for a robotic application to detect and classify even the static objects. Static objects such as parked cars or standing pedestrians can be dynamic at another point in time. Although most robotic applications map the static environment for navigation purposes, correct identification of possible dynamic objects is paramount for tracking purposes. However, there is a difference in detecting static and dynamic objects, since it is impossible to detect movement from a single camera or LiDAR measurement, as they do not contain velocity information. This is why dynamic object detection and tracking focus on detecting the movement over multiple time instances in order to estimate their future location [6]. On the other hand, correct classification of objects can determine if the objects are static or dynamic from one single measurement, since stationary objects that can move are classified as dynamic objects. For example pedestrians and vehicles can always be classified as dynamic objects, even though they were stationary.

Feng et al. [15] propose three criteria for object detection systems for autonomous driving. First, the methods need to be accurate i.e. giving accurate perception of the surroundings. Second, the methods need to be robust i.e. they are required to work in different conditions such as different weather conditions and environments. Lastly, the methods need to work in real-time, especially when traversing at high speeds. The criteria are also applicable to other autonomous mobile robotics, thus, they are also used in this work to evaluate different methods.

Recent development in deep learning methods has significantly increased the accuracy of the state-of-the-art object detection and classification [15, 36]. In the KITTI benchmark 3D object detection evaluation leader-board [19] has object detection accuracies over 70% for moderate and hard objects and over 80% for easy objects. Furthermore, for 2D objects the accuracy is over 90%. Accuracy is measured with the PASCAL intersection-over-union (IoU) criterion [14] i.e. over 50% of the bounding box must be overlapping with the ground truth.
According to Feng et al. [15] state-of-the-art deep learning methods can be roughly categorized into two categories: two-stage or one-stage object detection. In the two-stage detection, first object bounding box proposals are made and second, the proposals are refined. On the other hand, in the one-stage detection, the method aims to create bounding boxes and classification directly from the feature maps with a single convolutional neural network (CNN) model. The benefit of two-stage object detectors is that they are more accurate than one-stage detectors. However, one-stage detectors tend to be faster and require less training.

As stated by Goodfellow et al. [23] deep neural networks (DNN) are a highly effective tool for supervised learning, given a sufficiently large dataset. Deep learning methods are based on learning features of the objects directly from the data instead of humans engineering them [36]. This makes them ideal for object detection purposes, since in real life environments there are too many parameters for any human to tune. This can be seen from the KITTI leader-board [19], as there are only DNNs among the best ranking methods. Traditional object detection methods such as point cloud foreground segmentation or image edge detection can not produce as good results as deep learning methods [36]. However, there is one major caveat with deep learning methods: they require a large amount of training data and have a tendency of overfitting, which must be taken into account in the training data [23]. Due to this limitation, in this work traditional methods are also discussed.

**Deep learning** refers to multilayer representation learning i.e. methods have multiple levels of representations that are discovered automatically from raw data [36]. Especially in object detection, the methods used are usually a form of supervised learning, which means that they learn the representations from a large set of labeled data, which is fed to the neural network (NN) [15, 36]. The NN produces an output, which is then used in the backpropagation step to calculate gradients for the weights of the network using a loss function [36]. This basically means that the output is used to calculate the error between the label of the input data and output label and the error is used to calculate how much the weight of each neuron in the neural network is tuned.

**Deep neural networks** refer to an essential deep learning framework, which consist of multiple layers of neurons, which are simple perceptrons, which map input data to an output [23]. Network refers to the structure of the perceptrons, which are connected to each other in each layer and deep refers to the network having multiple layers. There are multiple different kinds of structures and types of DNNs, but the most commonly used in object recognition are CNNs, which are a type of feedforward network, which means that there are no feedback loops in the network [23].

**Traditional methods** refer to methods, which use features tuned by human engineers instead of learning features from raw data. Traditional methods are not limited by the domain or the amount of training data as much as DNNs, but they are prone to human error and can not have as many parameters as DNNs that can have hundreds of millions of weights to adjust [36]. Thus, the accuracy of traditional methods can not compete with DNNs [15, 36]. However, traditional methods are a favorable option, when there is not enough data to learn from, since DNNs require a large amount of data [23]. LeCun et al. [36] stated, that a typical DNN can be
trained with hundreds of millions of labeled data points.

In this section, the related work in object detection with camera and LiDAR is discussed. First, camera based detection methods are discussed. Second, LiDAR based detection methods are discussed. Finally, multi-sensor fusion and fusion based detection methods are discussed.

2.1 Camera based detection

Image based methods have been a staple in object detection, as cameras are the most commonly used sensor type, due to cameras being cheap and their accuracy in object classification is unparalleled compared to LiDARs and radars [30]. Thermal cameras have also been used in object detection, but since in this work thermal cameras are not used, the remaining work will focus on more traditional visual spectrum cameras.

According to Janai et al. [30] the traditional computer vision pipeline consists of preprocessing, region of interest (RoI) extraction, object classification and refinement. However, in this work traditional methods for image based object detection are not going to be discussed, since state-of-the-art deep learning methods outperform traditional methods by a significant margin [15, 18, 30, 38]. There are pretrained DNN models available such as YOLOv3 [52] and there are lots of open image data available for training own models, such as ImageNet [56]. Hence, the focus will be on deep learning methods.

Object detection from images refers to spatially locating an object from the image and representing the objects location with a bounding box [18]. This is a common approach used in mobile robotics and computer vision, since it is often enough to know a rough spatial location of the object.

According to Liu et al. [38] object detection methods can be divided into two types: detection of broad categories and detection of specific instances of a category. The former means detecting if an image is an image of a car or a cat and the latter means detecting if an image is an image of a specific car, such as a specific model or type of a car. Furthermore, authors denote generic object detection to include the spatial location and the size of the detected objects in the image. As the location of the object is of interest in mobile robotics, in this work object detection will refer to generic object detection.

Semantic segmentation refers to pixelwise classification of images i.e. classifying each pixel in an image to an object class [18]. On the other hand, instance segmentation gives separate labels to separate instances of the same class [18]. As semantic segmentation only classifies every pixel into broad categories, it is not as useful for autonomous mobile robotics as instance segmentation, since if there are more than one instance of the same object class in the image, the actual tracking of the objects is difficult without the localization. On the other hand, instance segmentation provides also information of position, shape and count, making it much more viable for mobile robotics [30].

Deep convolutional neural networks (DCNN) have completely revolutionized modern image based object detection [15, 30, 38]. Krizhevsky et al. [33] introduced their DCNN, AlexNet, in 2012, which broke all the records in image
classification accuracy against other competitors that used traditional methods in Large Scale Visual Recognition Challenge (ILSVRC) [56]. Ever since then, DCNNs have been the focus of computer vision research [30, 38]. DCNNs are a specific type of feedforward DNNs i.e. they do not have feedback such as recurrent neural networks (RNN) [23]. As stated by LeCun et al. [36] DCNNs are based on four key ideas: local connections, shared weights, pooling and the use of many layers, hence, the architectures of typical DCNNs are similar. DCNNs contain multiple layers of levels of fully-connected, convolutional and pooling layers, which are used for feature learning.

A **fully-connected layer** or a linear layer is a type of NN layer, which can be regarded as the basic block of a NN. A fully-connected layer has all of its neurons connected to each of the neurons of the previous layer, hence the name fully-connected. The term linear layer comes from the linear matrix multiplication the layer performs:

\[ f(x) = w^T x + b, \]

where the \( w \) is the weight matrix, the \( x \) is the feature vector and \( b \) the bias vector [23].

A **convolutional layer** is a type of a NN layer, which runs a sliding kernel window on top of the feature matrix and calculates a discrete convolution operation,

\[ s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a), \]

on the features [23]. An illustration of convolution process can be seen in Figure 1. In image case for example, a 2x2 pixel kernel window may be run iteratively through the whole image and performing the convolution operation between the pixel values and the kernel weights. This is usually done multiple times to all channels in the image. The convolution operation performed by the kernel causes the convolutional layers to have local connectivity, which means that each neuron of the layer is connected only to a few neurons of the previous layer. As stated by Goodfellow et al. [23] the local connectivity of convolutional layers enable the network to be more scalable i.e. it enables there to be more layers and more feature maps i.e. channels in the network, since they are not as memory intensive as fully-connected layers. Furthermore, this in turn enables the system to learn more local features, making them ideal for tasks such as image recognition, where there is a lot of local correlation. Local connectivity and full connectivity are illustrated in Figure 2.

The convolutional layers consist of channels, also called feature maps, which are connected locally to the feature maps of the previous layer [36]. This essentially means that the channels of a convolutional layer are each learning a single feature of the object and the locality means that the units of the layer contain only local features from an area size of the kernel of the layer. Furthermore, a single convolutional layer usually contains a huge number of channels ranging anywhere from tens to thousands of channels. The reason for such an architecture is that the data from array data, for example images, is highly correlated locally, i.e. pixels next to each other are often correlated [36]. Furthermore, each feature map of the layer shares parameters, since
the feature patterns can often be repetitive in images, thus, there are different units sharing the same weights detecting the same pattern in different parts of the image [36].

**Activation function:** The output of a fully-connected or a convolutional layer, which can be thought of as a set linear activations [23], is usually passed down through an activation function, which is a non-linear function such as a rectified linear unit (ReLU),

\[ f(x) = \max(0, x) \]  \hspace{1cm} (3)

**Pooling layer:** After a convolutional layer or multiple convolutional layers, there is usually a pooling layer, such as max pooling, which is meant to merge semantically similar features [36]. Max pooling calculates the maximum of local units.
Figure 2: (Top) Local connectivity of neurons. Each neuron is connected only to a few neurons of the previous layer. (Bottom) Full connectivity of the neurons. Each neuron is connected to each neuron of the previous layer. [23]

A typical DCNN architecture consists for image processing of one or a few convolutional layers with non-linearities and a pooling layer, followed by more of similar blocks and finally fully-connected layers [36]. Architecture of the Darknet [49] network used by YOLOv3 [52] image detector can be seen in Figure 3.

As stated in Section 2, state-of-the-art image recognition networks can be divided into two categories: two-stage and one-stage object detectors, where two-stage detectors tend to produce better accuracy than one-stage detectors with the expense of computational time and training time [15]. Two-stage detectors include a preprocessing step, where they generate object or region proposals, which are a set of preliminary bounding boxes or regions that may contain objects [38]. There has been a lot of progress done in the field of object detection after AlexNet [33]. According to Liu et al. [38] most notable state-of-the-art two-stage detectors are RCNN [21, 22], SPPNet [25], Fast-RCNN [20], Faster RCNN [53] and RFCN [9].
Figure 3: The Darknet [49] architecture used by YOLOv3 [52].

**RCNN:** Girshick et al. [21, 22] introduced RCNN (Regions with CNN features) network in 2014, where they used AlexNet [33] to extract feature vectors from region proposals extracted by selective search. After the feature extraction, they used the features as an input for set of class-specific linear support vector machines (SVM) [63] to classify each region. The main drawbacks of this region based method are the computational time and the training time required, even though the detection quality was high [30, 38].

**SPPNet:** He et al. [25] introduced SPPNet, which used spatial pyramid pooling to compute the feature map with one run of the CNN instead of running it for each region separately, as was done in RCNN [21]. This reduced the computational time by an order of magnitude [25]. Furthermore, it solved a problem with variable sized input for CNNs, since the fully-connected layers in the end of the network only accept fixed size inputs, unlike convolutional layers, which accept variable sized inputs. They applied the spatial pyramid pooling after the last convolutional layer to fix the input size for the fully-connected layers.

**Fast RCNN:** Girshick et al. [20] introduced Fast RCNN in 2015, which improved their prior work and tried to solve the disadvantages of RCNN [21] and SPPNet [25]. They improved the training algorithm of the RCNN to allow single-stage training using multi-task loss, instead of training all the stages separately as was done in RCNN. Furthermore, they required no more storage for feature caching. Moreover, they were able to increase the detection quality by using region of interest (RoI) pooling layer between the last convolutional layer and the first fully-connected layer, which produced a fixed size feature vector to the fully-connected layers. Lastly, the fully-connected layers fed into two output layers: first produced a softmax probability estimate for a set of classes and the second outputted bounding box positions for the classes. Architecture of Fast RCNN is illustrated in Figure 4.

**Faster RCNN:** Ren et al. [53] further improved the Fast RCNN [20], by removing the external region proposals used as an input for the CNN, which were the bottleneck of the Fast RCNN. Instead, they used the feature vector created by the CNN as an input for a Region Proposal Network (RPN). Furthermore, the region proposals were classified using the Fast RCNN detector.
RFCN: Dai et al. [9] proposed a further improvement to Faster RCNN [53]: Region based Fully Convolutional Network (RFCN), which removed the fully-connected layers from the network, since the region-wise network that did the classification and bounding box regression had to be ran for each RoI separately. Furthermore, Dai et al. [9] added a set of position-sensitive score maps created with specialized convolutional layers with a position-sensitive RoI pooling layer. They were able to achieve faster computational time with comparable accuracy with RFCN with ResNet101 [26] than Faster RCNN. Architecture of RFCN is illustrated in Figure 5.

While two-stage object detectors have produced ground breaking results in detection quality and accuracy, their biggest drawback is the computational time [38]. This makes them unfavorable for mobile robotics, where one criterion for object detection is real-time application, as mentioned in Section 2. For their faster computational time, the one-stage detectors are being researched heavily [38]. The idea with one-stage detectors is to predict class probabilities and bounding box locations using one
feedforward CNN [15, 38]. According to Liu et al. [38], the most notable state-of-the-art single-stage object detectors are DetectorNet [62], OverFeat [58], YOLO [50], YOLOv2 and YOLO9000 [51], SSD [39] and CornerNet [35]. Furthermore, Redmon et al. have since then improved their prior work with YOLOv3 [52].

**DetectorNet:** Szegedy et al. [62] introduced their object detector DCNN, DetectorNet, in 2013, which utilized the AlexNet [33]. Their idea was to use bounding box masks to the object detection regression problem. They used five networks: one for prediction foreground pixels from background pixels and four networks to predict the objects top, bottom, left and right sides. Then they grouped the predicted output into bounding boxes. However, they required the detector to be trained per object and per mask type, resulting in larger computational time at training.

**OverFeat:** Sermanent et al. [58] introduced their single-stage object detector OverFeat in 2013, which utilized a network similar to AlexNet [33]. They performed the end-to-end object detection with a single forward pass through the network. They performed object classification using a sliding window approach on the multiscale images to find object candidates. After the last convolutional layer they used a offset max pooling to increase the number of predictions and then applied a bounding box regression. Finally, they merged the bounding boxes. They were able to achieve greater speed than the RCNN [21], with the expence of the accuracy of the detector.

**YOLO:** Redmon et al. [50] introduced their single-stage object detector You Only Look Once (YOLO), in 2016. The idea of YOLO is to consider the object class probabilities and bounding box locations as a single regression problem. They divide the image to $S x S$ grid and predict $B$ bounding box locations, $C$ class probabilities and prediction confidences for each cell of the grid. This is illustrated in Figure 6. They were able to run YOLO at 45 frames per second (fps) and Fast YOLO at 155 fps on PASCAL VOC (Visual Object Classes) 2007 [14] detection challenge.

**SSD:** Liu et al. [39] introduced their Single Shot Detector (SSD) in 2016, which achieved faster processing times than YOLO [50] while achieving similar accuracies as the Faster RCNN [53]. Their idea was to add multiple feature layers, basically a convolutional filter, at the end of their base convolutional network, used to produce a set of detection predictions.

**YOLOv2 and YOLO9000:** Redmon et al. [51] further improved their prior work with YOLOv2 and YOLO9000. YOLOv2 replaced the GoogLeNet [61] network with the Darknet-19 [49] network with batch normalization, removed the fully-connected layers after the convolutional layers and used anchor boxes to predict the bounding boxes. YOLOv2 achieved the state-of-the-art on PASCAL VOC 2012 [14] challenge on detection quality and was faster than any other method. YOLO9000 on the other hand, was trained to detect over 9000 object categories.

**YOLOv3:** Redmon et al. [52] even further improved their prior work with YOLOv3, which switched the Darknet-19 to bigger DCNN Darknet-53. Furthermore, they used logistic regression in their bounding box regression and multilabel class prediction. They were able to achieve higher accuracies than SSD [39] with three times faster computational time. Architecture of Darknet can be seen in Figure 3.
DCNNs have become the state-of-the-art in image based object detection by a significant margin and they have improved at a tremendous pace within the last decade. Two-stage detectors have reached higher accuracies than one-stage detectors. However, currently two-stage detectors cannot reach real-time performance, which is why one-stage detectors are the preferable choice for autonomous mobile robotics.

2.2 LiDAR based detection

3D LiDARs are commonly used sensors in autonomous vehicles and mobile robotics in current research [59]. 3D LiDARs produce a point cloud with 360° horizontal field of view (FOV) containing accurate range information of the environment, by emitting laser beams and measuring the reflections [15]. 3D LiDARs give highly accurate position estimates, since the sensor measurement error is small. For example the accuracy of Velodyne VLP-16 16-beam LiDAR is ±3 cm [64]. LiDARs are more robust to weather and lighting conditions than cameras, but since the resolution is much sparser, they are inferior in object classification problems [15].

Points of the LiDAR point cloud can be roughly categorized in two categories: background and foreground points. Background points contain mostly ground points and foreground points on the other hand contain dynamic objects. Static objects can be included in either depending on the use case. Even though in this work the focus is on detecting dynamic objects, static objects are required to be found for later use, since the application requires also static obstacle detection. Hence, it is better to find all objects in the vicinity of the robot and classify them as static or dynamic than try to find separately dynamic and static objects, since point cloud
processing is computationally heavy. Thus, in this work foreground points contain all non-ground points.

In this section, related work on LiDAR based detection methods are discussed. First, traditional foreground segmentation methods are discussed. Second, machine learning methods used for LiDAR based detection are discussed.

### 2.2.1 Foreground segmentation

**Foreground segmentation** i.e. the clustering of 3D foreground points into groups that represent individual objects [41] is a critical part in most of the object detection methods utilizing only a 3D LiDAR point cloud, since most of the points of a point cloud are considered as background points. The foreground segmentation methods can be roughly divided into two categories: those that filter out the background points first, e.g. a combination of ground plane estimation and clustering of foreground points, and those that process the whole point cloud, e.g. occupancy maps.

The methods can be evaluated on accuracy of the method and the computational time, since point cloud segmentation methods are robust to condition and environmental changes. The biggest problem with many methods is the computational complexity of the algorithm as it can grow the computational time significantly as the amount of data in point clouds is high. This can in turn make the methods not applicable for real-time performance. Here real-time requirement means that the point clouds have to be processed before the next point cloud is received from the sensor i.e. in less than 100 ms, since for example Velodyne VLP-16 LiDARs rotate at 5 – 20 Hz frequency [64], where 10 Hz is often used.

Nguyen et al. [44] surveyed 3D point cloud segmentation methods. They grouped the methods into five different categories: edge, region, attributes, model and graph-based methods. Edge-based methods detect the edges of objects to determine their shape. They have the benefit of being fast, but are sensitive to noise and changes in the density of the point cloud. Region-based methods on the other hand, combine nearby points using neighbourhood information. While they are not as susceptible to noise as edge based methods, they have the problem of over or under segmenting and require precise seed point selection or large amount of prior knowledge. Attribute-based methods cluster attributes such as density or surface texture of the point cloud data, making them accurate and robust. However, they can be computationally heavy, making them not applicable for real-time performance. Model-based methods, such as the Random Sample Consensus (RANSAC) [16] or the Hough transform [3], use basic geometric shapes, such as a plane, a circle or a sphere, to group points. Model-based methods are robust and fast, but they are dependent on the chosen model. Lastly, graph-based methods represent point clouds as a graph and their benefits are that they are accurate and efficient. However, they often use machine learning methods or other sensors to assist in segmentation and they are often not applicable for real-time performance. Authors concluded that robust real-time application has not been achieved and that in complex environments machine learning methods produce better results than traditional methods, since they are more robust to noise.
The methods surveyed in [44] were not applicable for real-time applications, therefore the remaining survey of the related work will focus on faster methods: those that filter the background points and cluster the foreground points, and grid based methods, since they have successfully been used in real-time mobile robotic applications. As the real-time performance of the segmentation algorithm is critical in mobile robotics, the complexity of the algorithm is a crucial component in making the algorithm applicable. As the size of the point cloud grows, the computational time of many methods increases exponentially.

Many of the methods try to filter out ground points first before clustering foreground points. An often used method for ground plane fitting is the RANSAC algorithm [16]. Asvadi et al. [1] used a piecewise ground surface estimation using the RANSAC algorithm to fit planes to slices of the point cloud data. The RANSAC is a computationally complex algorithm [11], which is why using the plane fitting only to slices of the point cloud data enables the algorithm to take into account the changes in ground shape and to reduce the computational time of the algorithm [1]. Their piecewise RANSAC algorithm can be seen in Figure 7. Asvadi et al. [1] concluded that their method is able to run at 0.3 fps, thus making it too slow for real-time computation. However, they calculated that 83.2% of their computational time was used to calculate the point cloud registration done with Iterative Closest Point (ICP) algorithm and only 7.1% of the computational time was used to ground surface estimation i.e. the plane fitting took around 200 ms to compute, thus making it still too slow for real-time processing.

![Figure 7: Piecewise RANSAC plane fitting by Asvadi et al. [1]. The point cloud points are divided into two slices, which are first gated to remove outliers and then a plane is fitted to the points.](image)

Douillard et al. [11] presented an alternative segmentation method to RANSAC called Gaussian Process Incremental Sample Consensus (GP-INSAC), which is an iterative probabilistic ground plane estimation. However, even though the method showed promise with the accuracy of the segmentation results, worked for sparse point cloud data and was computationally lighter than RANSAC algorithm, the computational time of the algorithm was 170 ms. Although, since most of the research have been done with 64-beam LiDAR, the computational time can be lower with 16 or 32-beam LiDAR, making the algorithm possibly capable of real-time performance.
However, the ground point filtering is only a part of the foreground segmentation, the computational time must be significantly lower, especially considering that the computational power of an on board computer in mobile robotics is not as high as a desktop computer.

Himmelsbach et al. [27] introduced a fast foreground segmentation algorithm that reached near real-time performance using a 64-beam LiDAR. They used simple line fitting to detect the ground plane. First the point cloud was evenly divided to sectors and each sector was divided to bins according to the distance from the vehicle. The distance each bin covered increased the further the bin was from the vehicle to account the dispersion of the LiDAR beams. After the points were divided to bins, a prototype point was calculated for each bin containing only the distance from the vehicle and the height of the point, which is illustrated in Figure 8. The height of the prototype points were decided according to the lowest point in the bin, since it is most likely to be on the ground plane. Authors mentioned that the average height of the points could be used as well. After prototype point calculation, a line was fitted iteratively to the prototype points. If a point did not anymore fit to the previous line, a new line was fitted after the previous one. This takes into account the curvature of the ground plane. Finally, all points in the point cloud were checked if points were close enough to the lines fitted to the sector. If a point was close enough, it was classified as an inlier i.e. ground point and if not it was classified as an outlier i.e. non-ground point. The ground removal part of the algorithm was efficient and accurate enough to be used in real-time. Their ground plane estimation can be seen in Figure 9.

Figure 8: Illustration of line fitting by Himmelsbach et al. [27]. A sector $S_0$ is divided to bins and all points in the bins $p_i$ are used to calculate a prototype point $p'_i$ for the bin. Then lines are iteratively fitted to the prototype points.

After the ground points have been removed from a point cloud, the remaining non-ground points i.e. foreground points need to be clustered in order to find the foreground objects. A common way of doing 3D point clustering is the Euclidean clustering i.e. clustering points according to the Euclidean distance between the points with an algorithm such as the one presented by Klasing et al. [32]. This
method finds all points that are closer than a predefined distance from each other in 3D space and classifies them to be in the same cluster. Furthermore, authors concluded that this method is more efficient and accurate than k-nearest neighbours (KNN) algorithm, which is also a commonly used clustering algorithm. However, the Euclidean clustering is not applicable to real-time processing if done to all foreground points as stated by Himmelsbach et al. [27]. Moreover, another popular clustering method k-means, a partial clustering algorithm, is often used especially in image segmentation. However, it is not applicable to point cloud segmentation, since it has restrictions in finding arbitrarily shaped clusters [32].

Since the computational complexity of 3D clustering methods is often too high for real-time applications, voxel based clustering methods, such as occupancy grids, can be used to reduce the amount of data. The idea behind voxel grids is to map the point cloud points to a discrete voxel grid according to their coordinates. Voxel based methods can be used to segment the whole point cloud or only the non-ground points. However, the applications usually differ between the two, since voxel grids can be used for clustering or detecting dynamic points by filtering out static occupancy values of voxels over multiple scans. Moreover, voxel based methods can be used in 2D or 3D, however, as stated by Klasing et al. [32], 3D occupancy grids can be computationally too heavy for real-time processing, due to the increase in complexity.

As stated by Klasing et al. [32], 2D occupancy grids are most often used for navigation. However, they are not commonly used for detection purposes, since the reduction of dimensionality makes it hard to determine the attributes of the objects. A middle-ground between 3D and 2D occupancy grids is 2.5D grids such as height maps. They retain the height information of the points in the voxel, while also reducing the dimensionality of the data and the computational complexity of the map. Zhang et al. [70] used a height map to detect vehicle-like objects in real-time. They first projected the points to a planar 2D polar grid, which retained the height information of the points in each cell. With the height information they filtered out ground cells by filtering out all cells under certain threshold. After ground cell
filtering, they clustered remaining cells according to the distance between the cells and created a 3D cube bounding box to the cells according to the shape and height of the cell cluster. Furthermore, they roughly classified the clusters to vehicle-like objects according to heuristics on length, width and height of the cluster.

Asvadi et al. [1] created a 3D Carthesian occupancy grid of the non-ground points. The grid was updated with each scan, after the point cloud registration, by updating the occupancy values of each voxel with a probabilistic function. Furthermore, probabilistic functions were used to filter out noise, by increasing the occupancy value of a voxel, if the voxel gets multiple hits over multiple scans and by reducing the occupancy value, if the voxel has not been hit. Lastly, they used the occupancy value to classify objects to static and dynamic by setting a threshold for the occupancy value. Since stationary objects are hit multiple times by the LiDAR in the same voxel, the occupancy values of the static objects are higher than those of dynamic objects. The occupancy grid is illustrated in Figure 10.

However, as 3D occupancy grids tend to be computationally heavy [32], the 3D occupancy grid calculation and dynamic and static object classification of Asvadi et al. [1] took almost 300 ms, which makes it not applicable for real-time applications. Furthermore, since the point cloud registration done with ICP took 83.2 % of the computational time and it is a critical part in using occupancy grids, it seems to make accumulating point clouds for occupancy map unfavorable solution for real-time applications. However, Miyasaka et al. [40] used similar method by using ICP algorithm with map matching to estimate the ego-motion of the vehicle and by creating a 3D occupancy grid in front of the vehicle to detect dynamic objects. They used the occupancy grid only in front of the vehicle to reduce the amount of data needed to be processed and the amount of memory needed for the processing. This helps to reduce the computational time of the detection process. Considering that Miyasaka et al. [40] were able to successfully use the occupancy grid in real-time driving environment and Asvadi et al. [1] had immensely slower processing time, it seems that the algorithm implementation plays a major role in achieving real-time processing capability.

Vo et al. [65] introduced an octree based segmentation method, where they
created a voxel grid utilizing an octree. The idea of an octree is to iteratively divide 3D cubes into eight equal sized cubes until a predefined criterion is reached, such as that each cube contains fewer than a certain number of point cloud points. This is illustrated in Figure 11. This reduces the number of empty voxels in the grid by not dividing them smaller than necessary, thus reducing the complexity of going through the grid. Vo et al. [65] showed a reduction of computational time by orders of magnitude compared to point-based segmentation method. However, they used significantly denser point clouds than are used in mobile robotics, hence their computational times were order of magnitude too high for real-time processing. Azim et al. [2] used a similar octree based method for creating an occupancy grid for detecting dynamic objects in real-time. They classified all deviations between two measurements as dynamic voxels and clustered the voxels to groups using Euclidean distance between the voxel centers as a criterion for the clustering. Furthermore, to increase the robustness of the algorithm, they filtered out all clusters that had too few points to be classified as a dynamic object. Moreover, they used shape of the bounding box i.e. the ratio between length, width and height to roughly classify the dynamic objects into predefined categories such as car, bicycles and motorbikes.

![Octree Diagram](image)

Figure 11: An illustration of Octree by Azim et al. [2]. Each voxel is divided to smaller voxels until each voxel contains fewer points than a predefined threshold.

Himmelsbach et al. [27] solved the problem of the increased computational time of 3D voxel grids and the reduction of dimensionality of 2D or 2.5D grids by using first immediate reduction of dimensionality of projecting non-ground points to 2D grid, while retaining the original point data inside the 2D cells for later use. Then they clustered the 2D cells using a connected components algorithm, which is often used in computer vision, as detailed in their prior work [28]. Connected components finds all non-empty cells that are connected to each other and classifies them as a cluster. After they had found all the connected cells, they went through the cells in each cluster to check whether the cells contained significant height differences. This is done to segment for example overhanging trees from cars below them. If a cell cluster contained height differences above a predefined threshold, then they
performed a 3D Euclidean clustering to all points in that cell cluster. Otherwise, they declared all the points in the cell cluster as one 3D cluster. This was done to reduce the complexity of the algorithm, while retaining the full dimensionality of the data. However, they were only able to run the algorithm in real-time using a 64-beam LiDAR without the 3D clustering part.

2.2.2 Machine learning methods

One of the biggest drawbacks of the traditional foreground segmentation methods is that they are not capable of accurate classification of objects. Instead, they are dependent on heuristics on shapes and sizes of the objects. Machine learning methods can be a solution for this problem, since they learn the features of the objects directly from the data itself and are not as dependent on human engineering than traditional methods. Furthermore, as discussed in Section 2, machine learning methods have the capability of optimizing orders of magnitude more parameters than human engineers. However, LiDAR based generic object detection is still a difficult problem that is widely researched [68] and does not reach the same levels of accuracy as image based detection systems, as discussed in Section 2.

There are a few different approaches to object detection with 3D LiDAR data. First, the object detection can be separated into three categories, as was done in Section 2.1: object detection i.e. spatially localizing and classifying an object, semantic segmentation i.e. pointwise classification of the point cloud and instance segmentation i.e. pointwise classification giving each instance of the same class a different label. Most of the methods focus on the former two. Second, the object detection can be done by a more traditional machine learning method such as SVM [63] or with deep learning methods such as DCNNs. The state-of-the-art methods are currently all DCNNs such as PointNet [47].

Support vector machines (SVM) have been commonly used in point cloud cluster classification especially in mobile robotics, since they require orders of magnitude less training data than DNNs. SVMs require thousands of training samples [68], where as DNNs can require hundreds of millions of training samples [36]. However, the number of features used with SVM is also significantly lower than DNNs. A major drawback of SVMs is that the features used are designed by human engineers, thus, the features might not be optimal for representing the objects. As stated by Zhou et al. [71] these hand picked features may not be able to adapt to more complex scenes and objects.

Wei et al. [68] used a SVM to detect traffic cones in road environments. They first clustered the point cloud using a modified density-based spatial clustering of applications with noise (DBSCAN) [13] clustering algorithm. Then they extracted features from the clusters for the SVM classification step. Finally, they ran a binary classification of the cluster features with a linear SVM to classify the points to traffic cones or not traffic cones. Wang et al. [67] used a similar approach for pedestrian detection. They mapped all point cloud points to a 2D occupancy grid and used connected components clustering for the grid cells. Pedestrian candidates were extracted from the clusters with height, width and length conditions. Lastly, they
classified each pedestrian candidate cluster using SVM using hand-picked features. Furthermore, they complemented the detections with a target tracker to improve the detection accuracy by 10 – 15%. They were able to reach over 90% true positive detection rate. Wang et al. [67] used a 64-beam LiDAR, whereas Wei et al. [68] used a 8-beam LiDAR, although they complemented the LiDAR with a RGB camera. However, this supports the idea of using SVM as a classifier for point cloud data, since they appear to be robust to point cloud density.

Even though point cloud based object detectors have not yet reached the accuracy of the image based object detectors, the current research has heavily focused on deep learning methods and DNNs are the current state-of-the-art even in the point cloud based object detection [34, 37, 47, 48, 59, 71]. As Shi et al. [59] stated, the 3D object detection from point clouds still has many challenges, which are caused by the irregularity of the point cloud data and the 6 degrees of freedom (DoF) of 3D objects. Even though many of the state-of-the-art 3D object detectors utilize camera and LiDAR data, as discussed in Section 2.3.2, there have been many advances in point cloud -only object detectors, which is discussed in this section.

**PointNet:** Qi et al. [47] introduced the PointNet architecture, which supports end-to-end learning from unordered set of points. The PointNet architecture is composed of two CNNs: a classification network and a segmentation network, which share features with each other. The classification network takes an array of points as an input, applies input and feature transformations, combines point features using a max pooling layer and outputs classification scores for each class. The segmentation network extends the classification network, as it combines global and local features and outputs scores for each point. The segmentation network performs semantic segmentation i.e. it gives a score for each class for each point. Authors made the observation that even though the point set is unordered there exists connectivity between points that are spatially close to each other, which must be taken into account in the network architecture. They were able to achieve state-of-the-art in ModelNet40 [69] classification challenge with their approach processing ≈ 1 million points per second while being robust to point cloud sparsity. Their accuracy dropped 2.4% - 3.8% with 50% of the points missing.

**PointNet++:** Qi et al. [48] further improved their prior work by introducing the PointNet++ network. The newer network is designed to tackle two issues: how to partition the point set and how to extract spatially local features through a local feature learner. One major idea of the CNNs is the local connectivity, which is not properly addressed with PointNet [47], since the point set is unordered. PointNet++ [48] addresses this issue by applying PointNet [47] recursively to a partitioned point cloud, which was partitioned according to the distance from the sensor. They were able to further improve their prior work and achieve state-of-the-art results in the ModelNet40 [69] classification challenge.

**PointCNN:** Li et al. [37] introduced the PointCNN network, which tries to solve the issue of the unordered point set losing the spatial local connectivity. They proposed using a convolution operation on a x-transformed point cloud, since the normal convolution operation does not produce as good results with the sparse and unordered point cloud data. They were able to achieve higher accuracy than the

**VoxelNet:** Zhou et al. [71] introduced the VoxelNet network, which divides the point cloud to 3D voxel grid representation, which is then fed to feature learning network. The feature learning network groups the points to the grid voxels and samples points from them, to reduce the amount of data. Then the points are run through their pointwise network CNN, fully-connected layer and max pooling layer to produce voxelwise features. The voxelwise features are then fed to convolutional middle layers, which combine the voxelwise features. Finally, they feed the output of the convolutional middle layer to a modified Region Proposal Network (RPN) [53], which outputs region proposals for each object. The architecture of the VoxelNet is illustrated in Figure 12. The VoxelNet outperformed state-of-the-art by a significant margin in the KITTI [19] 3D object detection benchmark.

![Figure 12: An illustration of the architecture of VoxelNet by Zhou et al. [71].](image)

**PointPillars:** Lang et al. [34] introduced their PointPillars network in 2019, which performed 3D point cloud object detection using only 2D convolutions, since 3D convolutions are computationally expensive. They first projected the point cloud to a 2D heightmap on $xy$-plane, where the $z$ values were stored in the height of the pillar. Then they fed the pillar information to the their pillar feature network, which utilized PointNet [47] to extract features from each pillar, which were then fed to their 2D CNN. Lastly, they used the output of their 2D CNN to predict 3D object bounding boxes with the Single Shot Detector (SSD) [39]. The architecture of the PointPillars is illustrated in Figure 13. They showed that their approach was able to outperform state-of-the-art in the KITTI [19] 3D object detection challenge both in 3D detection benchmark and the bird’s-eye view (BEV) that included also the hybrid object detector that used LiDAR and camera data. Qualitative analysis of the detection results is illustrated in Figure 14. Furthermore, their network was able to run with 62 $Hz$ frequency, which is 13 times faster than the 4.4 $Hz$ of VoxelNet [71].
Figure 13: An illustration of the architecture of the PointPillars by Lang et al. [34].

Figure 14: Detection results of PointPillars [34] projected as 3D bounding boxes on top of BEV point clouds and RGB images.

**PointRCNN**: Shi et al. [59] introduced their PointRCNN network, which is a two-stage point cloud-only network. The stage one network generates 3D bounding box proposals in a bottom-up manner i.e. it takes the whole raw point cloud as an input and first utilizes the PointNet++ [48] to extract pointwise features of the raw point cloud. Then they simultaneously generate bin-based 3D bounding box proposals and segment the foreground points from the cloud. The stage two network refines the bounding box proposals and outputs 3D bounding boxes and confidence scores for each object. They were able to outperform the state-of-the-art in KITTI [19] 3D object detection challenge including the hybrid object detectors by a significant margin. However, they did not report the performance time of the network, which tends to be longer for the two-stage object detectors. Thus, an assumption can be made that they did not outperform the state-of-the-art single-stage object detectors in computational time.

LiDAR-based object detection has become more and more relevant field of study as LiDARs are highly used sensor in autonomous driving research. However, the LiDAR-based generic object detectors are not yet on par with image-based detectors in terms of the accuracy. Nonetheless, the LiDAR-based object detection is currently intensively researched topic. Even though deep learning methods have become the state-of-the-art even in LiDAR-based detection, traditional methods are still utilized significantly more in point cloud processing compared to image-based detection.
2.3 Multi-modal detection

In this section, multi-modal detection methods, i.e. methods that combine multiple sensor or detection modalities for object detection purposes are discussed. In this work, the modalities that are used are camera and LiDAR, thus, the focus of the discussion is on these modalities. First, the sensor fusion methods for fusing the data as well as what data to fuse and when to fuse are discussed. Second, multi-modal object detection methods are discussed.

2.3.1 Sensor Fusion

Sensor fusion is one of the biggest unsolved problems in multi-modal object detection and there is no consensus on what is the best way to fuse camera and LiDAR data especially for multi-modal object detection [15]. Feng et al. [15] discuss in their survey sensor fusion for deep multi-modal object detection. They addressed three topics, which are discussed in this section. What to fuse: what perception modalities should be fused. How to fuse: how the modalities should be fused. When to fuse: at what point of the detection pipeline the modalities should be fused.

What to fuse: Since the sensing modalities of camera and LiDAR data are different by design, the fusion of the modalities leads to trade-offs in some ways in almost every case. LiDAR data gives accurate but sparse spatial 3D position information and the camera gives dense and accurate 2D texture, color and illumination information. Thus, the sensor fusion approaches differ heavily depending on when the fusion is done. Roughly the approaches can be categorized in two ways: fusion of raw or preprocessed sensor data and fusion of detected features or object proposals [15].

According to Feng et al. [15] many methods that opt to use multi-modal DNN object detectors use the LiDAR data in 2D with one of two methods: a camera-plane map (CPM) or a bird’s-eye view (BEV) map. A CPM is produced by projecting 3D LiDAR point cloud into the coordinate system of the camera, given that the calibration matrix of the camera is known. A CPM can be fused with camera directly, since the size of both is the same. However, since the point cloud is much sparser than the camera image, many methods up-sample the point cloud data. A CPM can then be used in multiple ways such as creating a RGBD image, which incorporates a depth channel alongside the usual RGB color channels of an image, as was done in [17]. Furthermore, Schlosser et al. [57] and Banerjee et al. [4] both used the horizontal disparity, height above ground, and angle (HHA) image [24] to represent the depth information with the color image.

A BEV map projects the LiDAR point cloud to 2D map on the xy-plane, which makes localization and size estimation easier than a CPM, since it preserves the positions of the objects [34]. According to Lang et al. [34] BEV maps are currently commonly used point cloud presentation for deep multi-modal object detection, even though the authors argued that this might not be optimal, since the methods often use hand picked features in the grid representation of the BEV. Furthermore, Roddick et al. [55] even proposed a Orthographic Feature Transform (OFT) to create a BEV map from a single RGB monocular image for 3D bounding box calculation. Chen et
al. [7], proposed the use of both a CPM and a BEV map in their Multi-View 3D networks (MV3D).

Some multi-modal DNNs prefer to fuse the data at intermediate layers of the network as was done for example by Kim et al. [31], which fuses the feature maps of different layers as an input for the detection network. Furthermore, Schlosser et al. [57] tested the fusion of RGB image and HHA image in different stages of the network combining both images to one 6 channel network as well as combining the feature maps of intermediate network layers in different stages of the network. On the other hand, Chen et al. [7] also fused outputs of three different networks with their Region-based Fusion Network.

As opposed to multi-modal DNNs solutions, which use mostly a DNN to fuse either the raw data or the intermediate feature maps, traditional methods tend to fuse the outputs of single-sensor detectors. A common approach is to detect objects from LiDAR data using clustering and a possible SVM classifier [63] and to detect objects from images, represented by bounding boxes, by a DCNN or by using histogram of oriented gradients (HOG) [10], and fuse the individual detections together as was done for example in [8, 12, 66].

**How to fuse**: In case of raw sensor data, how to fuse depends heavily on the detection architecture. For example the fusion of a CPM and image data is trivial, since they are already in the same frame and of the same size as described in *what to fuse*. A CPM or BEV map can be used to create a HHA image or RGBD image, which can then be fed to a DNN [4, 24, 57]. On the other hand, for the fusion of intermediate layers of a DNN Feng et al. [15] summarized typical feature map operations used in intermediate fusion in DNNs: element-wise *addition*, element-wise *average mean*, feature map *concatenation*, *ensembling* feature maps from different sensing modalities, and *mixture of experts*, which means using networks to learn the weight of different sensing modalities, i.e. how much information each sensing modality contributes.

The fusion of outputs of single-sensor detectors is not as trivial as fusion of raw sensor data, since the outputs of single-sensor detectors are already highly processed. Cho et al. [8] chose to do the fusion of detection in the tracking part of their detection pipeline by utilizing an Extended Kalman Filter (EKF) by feeding the tracker with detections from individual single-sensor detectors. On the other hand, El et al. [12] used LiDAR clusters to extract RoIs in the image plane, HOG descriptor [10] to extract features of the RoIs from the images and a SVM classifier [63] to classify the RoIs.

**When to fuse**: According to Feng et al. [15] the sensor fusion can be done in three distinct stages, which have their pros and cons: early, intermediate or late. Early fusion is the fusion of raw or preprocessed sensor data, such as fusion of a CPM and a camera image. The benefit of early fusion is that the detector learns the joint features of all sensing modalities at an early stage without information loss caused by the processing of single modalities. However, this might cause information loss during the fusion of modalities, due to the representations of the sensor data such as the CPM or BEV map. Furthermore, the models are inflexible to changes in sensing modalities or extensions as the networks need to be retrained completely.
Late fusion is the fusion of outputs of different networks or detectors [15], such as was done in [7], which combined outputs of three different networks with one fusion network. According to Feng et al. [15] late fusion in DNNs are highly flexible and modular, as there is no need for retraining of every network in case of changes of sensing modalities. However, they come with the expense of high computational cost and the information loss of the intermediate features of the network. Furthermore, traditional methods that use for example HOG descriptors [10] or SVMs [63] are considered as late fusion, since they fuse the outputs of multiple detectors.

According to Feng et al. [15] middle or intermediate fusion is the combination of early and late fusion. It fuses the feature maps of intermediate layers of the DNNs as was done in the [31] and [57]. This has the benefit of learning combined modalities and rich feature representations at different depths of the network [15]. Different fusion methods are illustrated in Figure 15.

![Figure 15: An illustration of different fusion methods in deep multi-modal object detection by Feng et al. [15]](image)

### 2.3.2 Multi-modal object detection methods

As discussed in Section 2, the deep learning methods used for object detection have surpassed traditional methods by a significant margin. However, there are still use cases for traditional or other machine learning methods such as when not enough labeled training data is available. Most traditional methods that combine camera and LiDAR data use some sort of foreground segmentation or ground removal and clustering combined with a possible SVM classifier [63] for processing the LiDAR data, and a HOG descriptor [10] or a DCNN for the camera data. Even some of the approaches, which utilize traditional methods for the LiDAR data, might use a camera DCNN detector, such as YOLO [50], as was done in [68]. This is possible because the existing amount of labeled image data is orders of magnitude bigger than the LiDAR data [15]. Hence, it is often possible to use a deep learning approach in a
camera detector, even though deep learning is not possible for LiDAR data. This kind of sensor fusion is classified as late fusion.

Cho et al. [8] used camera, LiDAR and radar in their approach, but since radar is not of interest in this work, only the LiDAR and camera approach is described here. They extracted their objects from the point cloud using traditional foreground segmentation and presented the objects as 2D bounding boxes in the $xy$-plane. Furthermore, they extracted 2D bounding boxes and classes for the objects from the images using HOG descriptors [10] and a SVM classifier [63]. Lastly, they combined the data in their tracker, which had three observation models, one for each sensor, which were used as an input for an EKF. Likewise, Vu et al. [66] fused single-sensor detectors in the tracking phase of the pipeline by optimizing a batch of trajectories created by the detections. Like Cho et al. [8], they used a HOG descriptor for the images and an occupancy grid to filter out dynamic objects from the LiDAR data.

Premebida et al. [46] used similar approach to [8], utilizing HOG descriptors [10] and SVM classifiers [63] for their pedestrian detection. However, they opted to do an early fusion by creating a depth image from upsampled LiDAR data projected on the image plane. They ran the RGB and the depth images through the HOG descriptors and trained their SVM models using features extracted from the images. On the other hand, as described in Section 2.3.1, El et al. [12] used late fusion while using similar methods. They used DBSCAN [13] to cluster the LiDAR points, which were then used to create candidate RoIs in the image plane. Furthermore, they used a HOG descriptor to extract features from the candidate RoIs in the images, which were then fed to a SVM classifier to classify the RoIs as a pedestrian or non-pedestrian. Moreover, Wei et al. [68] used LiDAR foreground segmentation and SVM classifier for the LiDAR data and YOLO [50] object detector for image data and fusing the outputs with a NN.

Gao et al. [17] created a CPM of a point cloud from a LiDAR, which was then utilized to create a high-resolution depth image by upsampling the point cloud. The depth image was then used as a pair with the RGB image of the camera and fed as an input to a modified AlexNet [33], which used a four channel input image instead of the original three channel input. They were able to increase the average accuracy of the classifier by 15% compared to a RGB-model at KITTI [19] object detection challenge. Similarly to Gao et al. [17], Schlosser et al. [57] used depth features from upsampled LiDAR data. However, they used HHA images created from the depth features [24] with RGB images instead of using only one channel to describe the depth. They examined at which point of the pipeline the fusion of the two images is preferable to do. They found that early fusion i.e. fusing the input before the DNN and early intermediate fusion i.e. fusing features after first few convolutional layers produced the best results compared to the number of parameters needed. However, they noticed that late fusion of the outputs of two identical convolutional networks for each image fused by a fully-connected layer produced significantly better results at the expense of doubling the number of parameters needed in the network. Similar results were found by Banerjee et al. [4], who noticed that RGB and HHA image fusion performed better, if the fusion was done as in intermediate fusion i.e. in the DNN instead of early fusion of raw sensor data.
Chen et al. [7] presented their Multi-View 3D (MV3D) object detection network, which is composed of two parts: 3D Proposal Network and Region-based Fusion Network. The 3D Proposal Network combines three separate networks for LiDAR BEV map, LiDAR Front View image and RGB image from camera. The networks extract features from each separate representations, which are then fused in the Region-based Fusion Network, the output of which is fed to a multiclass classifier and a 3D bounding box regressor. Their approach outperformed all the previous LiDAR and image-based methods in KITTI [19] 3D object detection challenge. MV3D network can be seen in Figure 16.

![Multi-View 3D object detection network](image)

Figure 16: Multi-View 3D object detection network introduced by Chen et al. [7].

Kim et al. [31] suggested using a CPM and RGB image in separate identical DCNNs, similar to SSD [39], and fusing them through their Gated Information Fusion (GIF) Network, which combines the intermediate features of the different DCNNs from different layers using a fusion network. They achieved better or similar results than the state-of-the-art detector in KITTI [19] object detection challenge. GIF Network can be seen in Figure 17.

As stated by Feng et al. [15] the state-of-the-art in 3D object detection is achieved by deep multi-modal object detectors i.e. DNNs, which utilize both camera and LiDAR data, and the results are significantly better than those that use only a single sensor. Although some LiDAR-only detectors such as PointPillars [34] and PointRCNN [59] claim to have higher accuracy than state-of-the-art multi-modal approaches. The differences in accuracies of the state-of-the-art detectors are small. There is at least theoretical basis on multi-modal approaches producing better results, as all sensors have their pros and cons, thus, fusing the sensing modalities could improve the performance. Hence, it makes multi-modal approaches preferable to single-sensor approaches, even though there still are a lot of open questions in the sensor fusion of camera and LiDAR data for multi-modal detectors and the subject is currently under heavy research.
Figure 17: Gated Information Fusion Network introduced by Kim et al. [31].
3 Methods

This chapter describes the methods used in this thesis work. First, the robot system and its specifications and restrictions are described. Second, the LiDAR based segmentation is described. Third, the camera based detector is introduced. Finally, fusion of the two sensor modalities is described.

3.1 System description

The implementation of the detector done during this work was designed for an industrial logistics robot, which autonomously moves variable sized objects in an industrial warehouse and stockyard environments. However, during this work it will not be tested with the robot, as the robot is not yet in operation. Even though the robot is not available for testing, the detector system must be applicable to the domain and the hardware of the robot. Hence, it sets restrictions on the methods used.

The perception system of the robot consists of a Robosense RS-LiDAR-16 [54] 16-beam LiDAR, which has 360° horizontal FOV and 30° vertical FOV with 2,0° vertical resolution and 0,1° - 0,4° horizontal resolution and rotation speed of 5 - 20 Hz. Furthermore, there are two front-facing Stemmer Imaging DALSA GENIE NANO-C2590 IRC [60] RGB-cameras in the sensor head, which have 2592x2048 resolution and 22 fps frame rate. Moreover, the system contains odometry and Global Navigation Satellite System (GNSS) sensors, which are used for positioning, mapping and LiDAR rectification. However, they are not in the scope of this work, hence, they will not be discussed in detail. The system has a NVIDIA Jetson TX2 embedded computer.

The detector is a component of the detection and tracking of moving objects (DATMO) pipeline running on the robot. Hence, the system outputs the detections to a target tracker, which will not be part of this work. However, the whole pipeline must be taken into account in the implementation of the detector, as the detector is not stand-alone research on general object detection, but a real-life use case of a DATMO pipeline on an actual robot.

The object detection system must fulfill soft real-time requirements, which imposes restrictions on the methods used. The real-time requirements of this system are set by the sensor, which has the lowest frequency, which in this case is the LiDAR. The system is required to process every LiDAR measurement as missing measurements leads to loss of data. However, infrequent missing of deadlines does not lead to system failure but performance degradation. Moreover, the LiDAR spins at 10 Hz, hence the maximum processing time of a scan is 100 ms. Furthermore, there is always a trade-off between computational time and accuracy, and thus the methods have to be considered according to the hardware limitations of the platform in order to fulfill the real-time requirements.

Since the domain of the robot is in industrial logistics, no open datasets could be found with both camera and LiDAR data that would fit the domain, since most of the open datasets are aimed for autonomous driving development containing only
road traffic data. However, since the robot is not operational, the development and experiments are done with nuScenes [5] open road dataset, which is described in more detail in Section 4.2. Although, the methods are developed and tested with road data, the actual domain of the robot and the lack of data have to be taken into account in the design. These set some restrictions to the methods used, since most of the state-of-the-art methods are deep learning methods that require a lot of training data. Furthermore, enough data can not be gathered during this thesis work to use deep learning methods.

3.2 LiDAR foreground segmentation

In this section, the LiDAR foreground segmentation method is described. Since, deep learning methods are not applicable for this work, the foreground segmentation is done by more traditional methods. The foreground segmentation consist of background removal and object clustering. In the first subsection, the background removal method is described and in the second subsection the object clustering methods is described. In this work, the LiDAR rectification is given i.e. the ego-motion of the robot during a scan is taken into account and corrected in the LiDAR point clouds.

The LiDAR points are classified to two categories: background and foreground points. Depending on the goal of the segmentation, background points can consist of only ground points or ground points and static objects. Since in this work the robot is required to detect all obstacles in its path, the static obstacles should be included in the detections. Hence, the background should contain only ground points.

The methods used were chosen because they showed promising results in the literature with computational time and accuracy. Most methods in the literature are developed and evaluated with a dense point cloud produced by a 32- or a 64-beam LiDAR. However, the robot is equipped with a 16-beam LiDAR, thus, all methods are not applicable, since the sparsity of the point cloud makes it difficult to accurately segment objects especially far away. In the clustering part, the method had to be chosen carefully, since the computational complexity of clustering algorithms can be exponential, which can quickly increase the computational time [27].

The map-based methods were left out, since prerecorded maps are not always available and are susceptible to changes in the environment. Runtime generated maps such as occupancy grids require accumulation of points, which leads to slower detection rate, since multiple scans are required to detect objects. Local history of scans can be kept in memory, which reduces the need for accumulation of scans. Since occupancy grids are by design mapping the environment and filtering out noise and dynamic objects, they can be used to detect dynamic objects by filtering all static obstacles from the scan by matching them to the occupancy grid. However, it can be difficult to separate noise from actual true positive detections. Furthermore, the computational complexity of occupancy grids grows exponentially as the environment grows, which can lead to significant increases in computational time, especially with 3D occupancy grids [32].

Since 3D maps can be computationally heavy and 2D maps reduce the information too much [32], 2.5D representation such as a height map could be used. 2.5D maps
have been used successfully in real-time obstacle detection [70]. However, they produce the same problem as 2D grids: they reduce the information too much causing under-segmentation. For example, a car parked under a tree would register as one object as the map does not separate the overhanging leaves on top of the car from the car, since only the height information of cell is retained [27].

Since prerecorded maps and locally mapping the whole point cloud were ruled out, clustering the point cloud becomes a sensible solution. However, the computational complexity of 3D clustering algorithms is high, thus it is not reasonable to cluster the whole point cloud. Separating background and foreground points before clustering reduces the amount of points of interest significantly. However, the ground plane fitting algorithms such as RANSAC, INSAC and GP-INSAC can be computationally highly complex, causing them not to be applicable for real-time solutions [11].

The method used in this work is a modified version of the method presented by Himmelsbach et al. [27, 28]. First the ground is removed with a modified version of the algorithm presented in [27]. Second, the non-ground points are stored in a 2D grid and the non-empty 2D grid cells are clustered with a modified version of the algorithm presented in [28]. Finally, since the reduction of dimensionality by clustering 2D grid cells does not take into account the possibility of two different objects overhanging on top of each other, the 2D clusters are checked for large enough height differences. If the height difference is bigger than a predefined threshold, the points in the 2D cluster cells are run through a 3D clustering algorithm. First, the ground removal algorithm is described. Second, the object clustering is described.

3.2.1 Ground removal

The ground removal algorithm introduced by Himmelsbach et al. [27] showed promising results with the computational time even for a 64-beam LiDAR and it could separate ground and non-ground objects effectively. The algorithm used in this work was modified to work with the much sparser data of the 16-beam LiDAR used in the robot platform. The notation used to describe the algorithm is adapted from Himmelsbach et al. [27] to match the original algorithm.

The method first segments all unordered point cloud points of one scan at a time $t$, $P_t = \{p_1, ..., p_N\}$, where $p_i = (x_i, y_i, z_i)^T$, to a 2D circle on the $xy$-plane. The circle is then divided to sectors of size angle $\Delta \alpha$ and radius $r = \infty$. Hence, the circle consist of $M = \frac{2\pi}{\Delta \alpha}$ sectors $S_i$. The segment index $s = [0, M]$ of each point is calculated

$$s = \text{segment}(p_i) = \left\lfloor \frac{\text{atan2}(y_i, x_i)}{\Delta \alpha} \right\rfloor,$$

where $\text{atan2}(y_i, x_i) \in [0, 2\pi]$ is the angle between positive $x$-axis and the point. Thus, set of all points in one segment $S_s$ is denoted by

$$P_s = \{p_i \in P_t \mid \text{segment}(p_i) = s\}.$$

Then the points are sorted in ascending order according to the range from the sensor to $B$ bins, $b^*_j$, where $j = [0, B]$ inside the sector. Each bin covers range $[r_j^\text{min}, r_j^\text{max}]$, where $r_j^\text{min}$
growing as the distance from the sensor grows to account the dispersion of the LiDAR points. Thus, each point \( p_i \in P_s \) is mapped to a bin \( b_j^i \), if
\[
\sqrt{x_i^2 + y_i^2} < r_j^{\text{max}}.
\]
Furthermore, points mapped to bin \( b_j^i \) can now be denoted as a new 2D set
\[
P_{b_j} = \{ p_i' = (\sqrt{x_i^2 + y_i^2}, z_i) \mid p_i \in P_{b_j} \}.
\]
Then a prototype point \( p_{b_j} \) is calculated for each bin in each sector containing all points in that bin. The prototype point contains only a range from the sensor and the height of the point. The height of the prototype point is chosen to be the height of the point that has the lowest \( z \)-value, where \( z \)-axis is orthogonal normal of the ground plane, thus,
\[
p_{b_j} = (r_{b_j}, z_{b_j}^{\text{min}})^T.
\]
According to Himmelsbach et al. [27], the height could also be calculated by taking the average of all the points in the bin, however the lowest point works fine since that point is most likely to be closest to the ground. Sorting of the point cloud points to segments and bins, and the prototype point calculation are described in Algorithm 1.

**Algorithm 1 Point Cloud Sorting**

1. procedure POINTCLOUDSORTING\( (P_t) \)
2. for \( p_i \) in \( P_t \) do
3. \hspace{1em} \( s = \text{calculateSegmentIdx}(p_i) \)
4. \hspace{1em} \( P_s = P_s \cup p_i \)
5. for \( s = 0; s < M; s + + \) s do
6. \hspace{1em} for \( p_i \) in \( P_s \) do
7. \hspace{2em} \( j = \text{calculateBinIdx}(p_i) \)
8. \hspace{2em} \( P_{b_j} = P_{b_j} \cup p_i \)
9. \hspace{1em} \( i = 0; i < B; i + + \) i do
10. \hspace{2em} \( p_{b_j} = \text{calculatePrototypePoint}(P_{b_j}) \)

After all prototype points are calculated, a line in the form of
\[
y = mx + b,
\]
where \( \{m, b\} \in \mathbb{R} \), is fitted iteratively to the prototype points in each sector using root-mean-square error (RMSE). The algorithm iteratively goes through all prototype points in each sector in ascending order of distance from the robot starting from the closest one. It iteratively tries to fit a line to each point and all points that are closer to the robot. If the point fits to the line, it is considered as a ground point. On the other hand, if the point does not fit to the line, it is considered an outlier i.e. a non-ground point. The algorithm goes through all points and fits a line to all points that fitted on a line according to given thresholds. After that the algorithm
returns to the point that was the last one that fitted to the line and starts iterating further. This is done until all points are either fitted to a line or are classified as outliers. The iterative fitting allows the line to be fitted to slopes, such as hills, and the returning of the calculation back to the last fitted point after going through all points allows the algorithm to continue fitting a line after it has hit an outlier.

Furthermore, a line is considered to be a part of the ground plane only if it fulfills certain requirements. The slope of the line must be under a threshold \( T_m \) to avoid vertical structures being classified as a part of the ground plane. If the slope of the line is small, the absolute \( y \)-axis intercept \( b \), must not exceed a threshold \( T_b \). Moreover, the RMSE of the line fit must not exceed a threshold \( T_{rmse} \). Lastly, the distance between the first point of the line and the last point of the previous line must not exceed a threshold \( T_{d-prev} \) to smoothen the transition between lines.

The original algorithm in [27] started to fit a new line after one point does not fit the line that is currently being fitted. This enables the line to fit to slopes, but can cause the line not to be smooth. Continuing the iteration after a point does not fit a line and checking the rest of the points still enables the line to be fitted to slopes, but creates more robustness for outliers. This is especially important with the sparser point cloud created by a 16-beam laser, since as the distance of the points from the robot grows, the distance between one point to another grows significantly. It can cause even clear outliers to be fitted as a ground point, since the slope between two points is much smaller, if the distance between points is bigger, than if the points are close to each other. Line fitting is described in Algorithm 2.

After the lines have been calculated to each segment, the algorithm filters out all points that are closer to the fitted line than a predefined threshold \( T_{ground} \). The algorithm first finds the line that has the closest ending points, according to the Euclidean distance, to the iterated point to know which is the line that should be used for filtering, since one segment can contain multiple lines as the result of the iterative approach. After the correct line has been found, the shortest Euclidean distance between the line and the point is calculated and used for the filtering using

\[
d = \frac{|mr_p - z_p + b|}{\sqrt{m^2 + 1}},
\]

where \( A = m, \ x_p = r_p, \ B = -1, \ y_p = z_p, \ C = b \). Equation 10 is formed by combining

\[
ay + bx + c = 0 \iff nz + mr + b = 0,
\]

and

\[
d = \frac{|Ax_p + By_p + C|}{\sqrt{A^2 + B^2}}.
\]

The ground plane removal is described in Algorithm 3. Further, the ground removal is illustrated in Figures 18, 19, 20 and 21, which represent an unfiltered and filtered point cloud from a bird’s eye-view and from 3rd person view respectively.
Algorithm 2 Line Fitting For Segment $S_s$

1: procedure LINEFITTING
2:     $L_s = \emptyset$, $c = 0$, $P_l = \emptyset$, lastFittedIdx = 0
3:     for $j = 0; j < B; ++j$ do
4:         if $j = B - 1$ then
5:             if $|P_l| > 1$ then
6:                 line($m_c, b_c$) = fitLine($P_l$)
7:                 if line.m_c $\leq T_m$ $\land$ (line.m_c $> T_{\text{msmall}}$ $\lor$ line.b_c $\leq T_b$) $\land$
8:                     fitError(line($m_c, b_c$), ($P_l \cup p'_l$) $\leq T_{\text{rms}}$ then
9:                     $L_s = L_s \cup \text{line($m_c, b_c$)}$
10:                     $c = c + 1$
11:                     $P_l = \emptyset$
12:                     $j = \text{lastFittedIdx}$
13:             if $P'_j \neq \emptyset$ then
14:                 if $|P_l| > 1$ then
15:                     line($m_c, b_c$) = fitLine($P_l \cup p'_l$)
16:                     if line.m_c $\leq T_m$ $\land$ (line.m_c $> T_{\text{msmall}}$ $\lor$ line.b_c $\leq T_b$) $\land$
17:                         fitError(line($m_c, b_c$), ($P_l \cup p'_l$) $\leq T_{\text{rms}}$ then
18:                         $P_l = P_l \cup p'_l$
19:                         lastFittedIdx = $j$
20:                     else
21:                         if $c = 0 \lor P_l \neq \emptyset \lor \text{distancePointLine}(p'_l, \text{line($m_{c-1}, b_{c-1}$)} \leq$
22:                             $T_{\text{dpre}}$ then
23:                             $P_l = P_l \cup p'_l$
24:                             lastFittedIdx = $j$

Algorithm 3 Ground Plane Filter

1: procedure GROUNDPLANEFILTER
2:     for $s = 0; s < M; ++s$ do
3:         for $p_i \in P_s$ do
4:             line($m, b$) = findClosestLine($p_i$
5:             if distancePointLine($p_i$, line($m, b$)) $\geq T_{\text{dground}}$ then
6:                 $p_i \notin P_s$
Figure 18: An unfiltered point cloud from a bird’s-eye view.

Figure 19: A point cloud with ground removed from a bird’s-eye view.
Figure 20: An unfiltered point cloud from 3rd person view.

Figure 21: A point cloud with ground removed from 3rd person view.
3.2.2 Object clustering

After the ground points have been removed, the remaining non-ground points i.e. foreground points need to be clustered. Object clustering was adapted from Himmelsbach et al. [27] with heavy modifications. The original algorithm showed promising results in computational time, when the full 3D clustering was left out. However, with the sparser data of a 16-beam LiDAR, the method suffered heavily from the dispersion of the LiDAR data. The sparsity of the point cloud produced by a 16-beam LiDAR makes it extremely hard to tune the clustering algorithm correctly to take into account the dispersion of the beams. Thus, some modifications were required.

The algorithm used in this work first maps the 3D non-ground points to a 2D polar grid instead of Cartesian grid, to take into account the dispersion of points. The horizontal angular resolution of the VLP-16 is between $0.1^\circ$ and $0.4^\circ$ [64]. This means that the $xy$-distance between two points at 100 m range is 17 cm to 70 cm, making it hard to find a Cartesian grid size that would be applicable to close objects and distant objects. This is emphasized further by the fact that point clouds are irregular and often have missing points due to the reflectivity of the environment not being high enough. Furthermore, points from same object can be registered as belonging to different objects, since there can be even multiple grid cells between the two points. However, if a polar representation is used, where the 2D grid coordinates are represented as only the distance from the sensor $r$ and angle $\gamma$, the horizontal cell size grows according to the distance, since the $\Delta \gamma$ is constant. Furthermore, if the vertical cell sizes grow according to the distance from the vehicle, as was done in Section 3.2.1, the representation is equally applicable to short and longer distances.

The non-ground points $p_i$ are mapped to 2D polar grid cells $c_i \in C$ according to Equation 4 and Equation 6. Even though there is immediate reduction of dimensionality, the 3D information is not lost during the process, as the original point cloud points are stored into the cells. The non-empty 2D polar grid cells $C$ are then clustered using a connected component algorithm, which utilizes breadth-first search (BFS). BFS goes iteratively through all the non-empty cells, marks them as visited, adds them to a cluster, checks if the cells next to the cell, which is being processed, are non-empty and not visited. If they are, they are added to a queue $Q$. After that, the next cell in the $Q$ is removed from the $Q$ and the same process is done to that cell until there are no more cells in the $Q$. All cells found during this traversal are added to the same cluster. After the first cluster, the algorithm iteratively goes through rest of the non-empty cells in the grid until all cells are marked as visited. The 2D clustering algorithm is described in Algorithm 4.
Algorithm 4 2D cell clustering

1: procedure 2DCLUSTERING(grid)
2:   \( C_{ne} = \emptyset \)
3:   \( C_{ne} = \text{findNonEmptyCells(grid)} \)
4:   clusters = \( \emptyset \)
5:   \( Q = \emptyset \)
6: while \( C_{ne} \neq \emptyset \) do
7:   \( \text{cluster} = \emptyset \)
8:   \( Q = Q . \text{push}(C_{ne}.\text{pop}) \)
9: while \( Q \neq \emptyset \) do
10: \( c_q = Q . \text{pop}() \)
11: \( \text{cluster} = \text{cluster} \cup c_q \)
12: \( Q = Q \cup \text{findConnectedCells}(C_{ne}, c_q, Q) \)
13: clusters = clusters \cup \text{cluster}
14: return clusters

After the 2D cells have been clustered there is an optional filtering step, shown in Algorithm 5, which is used to remove the last ground points that were misclassified during the ground removal. The filtering step goes iteratively through all cells in the clusters and removes them, if the maximum height difference between any two points, \( p_{i}^{cs} \) and \( p_{j}^{cs} \), in the cell is less than \( T_{z_{diff}} \)

\[
|p_{max}^{cs} - p_{min}^{cs}| < T_{z_{diff}} \mid p_{cs}^{i} \in P_{cs},
\]

and the height of all points in the cell \( p_{cs}^{i} \) is less than \( T_{z_{min}} \)

\[
p_{max}^{cs} < T_{z_{min}} \mid p_{cs}^{i} \in P_{cs}.
\]

Algorithm 5 2D cell filtering

1: procedure 2DFILTERING(clusters)
2: for cluster \( \in \) clusters do
3:   for \( c_{i} \in C_{\text{cluster}} \) do
4:     if \( z_{max}^{i} < T_{z_{min}} \) and \( |z_{max}^{i} - z_{min}^{i}| < T_{z_{diff}} \) then
5:       \( c_{i} \notin C_{\text{cluster}} \)

After the filtering has been done, the 2D cell clusters are gone through to find the height difference, as was done in [27]. However, there was a modification done in calculation of the height threshold \( T_{z} \), to again take into account the dispersion of the LiDAR beams. Instead of using a constant value for all cells, the threshold was calculated as a function of distance from the sensor. The threshold utilized the vertical resolution of the sensor \( \theta \). The height difference for each cell was calculated

\[
\cos(\beta) = \frac{p_{i}^{cs} \cdot p_{j}^{cs}}{|p_{i}^{cs}||p_{j}^{cs}|} \leq \cos(T_{z} \Theta),
\]
where \( \mathbf{p}_i^{c_i} \) and \( \mathbf{p}_j^{c_j} \) are vectors from origin to points \( p_i^{c_i} \) and \( p_j^{c_j} \) respectively, \( \beta \) is the angle between the two vectors, and \( T_x \in \mathbb{N} \) is a positive integer multiplier determining how many times the vertical resolution of sensor can be fitted between the two points i.e. how many points can be missed between two points before they are qualified to be in two different clusters.

The final 3D clustering was done to all clusters that had too big height differences in at least \( N_{3D} \) cells, as was done in [27]. 3D clustering was done utilizing the same idea as in 2D cell clustering, BFS presented in Algorithm 4, but using 3D Euclidean distance to find connected points. All clusters that had points fewer a than predefined threshold \( T_{min\_points} \) were filtered out in order to filter out noise and too small objects. In the original algorithm [27] the 3D clustering was done using 3D voxel grid, however, in this work it was done using pointwise 3D Euclidean clustering, shown in Equation 16. The 3D clustering algorithm can be seen in Algorithm 6.

\[
    d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}
\]

**Algorithm 6 3D Clustering**

1. **procedure** 3DCLUSTERING(clusters)
2. \( \text{clusters}_{3D} = \emptyset \)
3. **for** cluster \( \in \) clusters **do**
4. \( \text{count} = 0 \)
5. \( \text{found} = \text{false} \)
6. **for** \( c_i \in C_{\text{cluster}} \) **do**
7. \( P_{c_i} = \text{sort}(p_{n_i}^{c_i} < p_{m_i}^{c_i}) \)
8. **for** \( (p_n, p_m) \) in \( P_{c_i} \) **do**
9. \( \beta = \frac{p_m - p_n}{||p_m - p_n||} \)
10. **if** \( \cos(\beta) \leq \cos(T_x \theta) \) **then**
11. \( \text{count} = \text{count} + 1 \)
12. \( \text{break} \)
13. **if** \( \text{count} \geq T_{N_{3D}} \) **then**
14. \( \text{found} = \text{true} \)
15. **if** found **then**
16. \( \text{clusters}_{3D} = \text{clusters}_{3D} \cup \text{3DEuclideanClustering(cluster)} \)
17. **else**
18. \( \text{clusters}_{3D} = \text{clusters}_{3D} \cup \text{cluster} \)
19. **return** clusters_{3D}
3.3 Camera detector

The camera detector module used in this work is based on the module presented by Peurasaari [45], which utilized a pre-trained network. The DCNN detector used in the camera detector module is the YOLOv3-416 [52], which used 416x416 pixel input images. There exists also a larger input version of YOLOv3: YOLOv3-608, which uses 608x608 pixel input images. Even though the YOLOv3-608 has better accuracy than the YOLOv3-416, it has almost double computational time compared to the YOLOv3-416 [52]. Thus, the YOLOv3-416 was selected with the trade-off of speed and accuracy. Furthermore, the camera detector module presumes that the intrinsic calibration of the camera is performed and the calibration matrix given.

YOLOv3 contains multiple Darknet [49] models, which is the network YOLOv3 uses, that are trained to detect different classes. There are models that are trained with large enough dataset that they can be applied to the domain of the robot described in Section 3.1: network used by YOLO9000 contains 9000 different classes [51] and there is one model used with YOLOv3 with 601 classes [49]. This work used in the experiments a Darknet version with 80 classes as it is enough for the test dataset.

As mentioned, YOLOv3-416 takes a 416x416 pixel image as an input, thus, the images are first resized to correct size and padding is added to match the ratio of the input size. YOLOv3 then outputs an array of classes, bounding boxes and confidence scores for each object detected. As mentioned in [45], only confidences above a threshold $\theta_D$ are accepted as measurements. Furthermore, Peurasaari [45] stated that each detector module requires 1.5 GB of graphics processing unit (GPU) memory, which limits the number of parallel detectors that can be used. However, the robot will have two front-facing cameras, thus, there is no need for more than a maximum of two camera detector modules running in parallel at any given time in the system.

3.4 Sensor Fusion

After the clusters have been extracted from the point clouds and the bounding boxes are extracted from the images, the modalities have to be fused together. Fusion is required in order to associate the clusters with classes and separate static objects from the dynamic ones, since the clustering of LiDAR points finds all static and dynamic objects and does not classify the clusters by any means. Since both sensors have different frequencies and the frequencies and delays of both sensors are constant, the sensing modalities have to be synchronized. Furthermore, there is no hardware synchronization available, thus the images and point clouds are synchronized on the software level. The synchronization is done by storing the images and finding an image with a timestamp closest to the timestamp of the point cloud every time a point cloud is received from the LiDAR.
The sensor fusion approach presumes that the extrinsic calibration between the camera and the LiDAR is performed. The extrinsic calibration is an important prerequisite for the system. Examples of proper calibration and improper calibration can be seen in Figures 22 and 23. In the successful calibration the points align with the objects in the image and in the unsuccessful calibration the points clearly do not match the objects in the image.

Figure 22: Successful extrinsic calibration between camera and LiDAR projected on top of the image. Data is from nuScenes [5] dataset.

Figure 23: Failed extrinsic calibration between camera and LiDAR projected on top of the image. Data is from nuScenes [5] dataset.
The point clouds and the images are processed in parallel and once both have been processed the sensor fusion step is performed. The fusion is done by creating a camera-plane map of the 3D point clusters by projecting the LiDAR points of the clusters to the camera frame, given the calibration matrix of the camera. The clusters are then associated with the Global Nearest Neighbour (GNN) data association method with the bounding boxes produces by the camera detector module. A cost matrix $C$ is calculated between the bounding boxes and point clusters.

The cost function is constructed of three parts: number of points inside the bounding box, the percentage of the points of the cluster that are inside the bounding box, and the distance of the cluster from the sensor. The idea behind the cost function is three fold: it gives a greater score for bigger clusters, since the biggest cluster inside the bounding box is often the object. However, the background clusters e.g. wall can be bigger than the actual object, thus the percentage of points that are inside the bounding box is taken into account. Lastly, the average range of cluster points from the sensor is taken into account, since often the closest object is the correct one as the clusters further away tend to be background points and objects behind the closer object are occluded by the closer object. The full cost function is

$$c_i = \frac{N_{Roi} \cdot N_{cluster} \cdot 100}{N_{cluster} \sum_{a=0}^{1} p_{a} z}, \quad (17)$$

where the $N_{Roi}$ is the number of points inside the region of interest i.e. bounding box, the $N_{cluster}$ is the amount of points in the cluster, $p_{a}$ is a cluster point, and the denominator is the average range of the points in the cluster from the sensor along $z-axis$, which is the normal vector of the camera-plane, i.e. the distance of the points from the camera-plane. The GNN association is then solved from cost matrix for maximum cost using Murty’s algorithm [42]. The data-association between a point cluster and a bounding box is illustrated in Figure 24.

After the GNN association between the clusters and the bounding boxes, the clusters are then formed into 3D bounding boxes with a class from the YOLO [52] detection. The bounding boxes are formed using principal component analysis (PCA) [29] on the cluster points. The PCA produces three 3D eigenvectors and eigenvalues that account for 100% of the spatial variance of the cluster points. The eigenvector with greatest projection along the $z$-axis of the LiDAR is chosen to represent the $z$-axis of the object. The $z$-axis of the object can be assumed to be perpendicular to the ground i.e. $xy$-plane, since the objects in traffic usually only rotate along the the $z$-axis i.e. only the yaw-angle of the object changes. In practice this mean that for example the floor and the roof of a car can be assumed to be parallel to the ground at all times. Furthermore, from the remaining two eigenvectors, the one with the greater eigenvalue i.e. the longer vector, is chosen to represent the objects $x$-axis and the final eigenvector is chosen to represent the $y$-axis. This assumption is made, since most of the dynamic objects found in traffic or in other domains, especially Ackermann vehicles, such as cars, have their length in the direction of their movement bigger than their width. Thus, their $x$-axis is chosen to be parallel to the direction of their heading. The only exception to this rule is pedestrians, as their length is
Figure 24: Data-association between a point cluster and a bounding box

similar to the their width. However, this is irrelevant, since pedestrians can move in any direction at any given time regardless of their orientation.

The size of the object is then calculated using the eigenvectors and eigenvalues. As the eigenvalues represent the variance $\sigma^2$ of the spatial variance of the cluster points, the standard deviation $\sigma$ can be calculated by taking the square root of the variance. Thus, the length of the vector can be calculated according to how many $\sigma$ of the data is wanted. This work used $3\sigma$, since it explains 99.73% of the clusters distribution and, since the LiDAR accuracy is high, the amount of outliers in the clusters is small. Thus, the size of the 3D bounding box is $3\sigma$ along the direction of the eigenvectors from the center of the box. Furthermore, the center of the box is calculated as a mean of the points of the cluster. Moreover, the size of the 3D bounding box is calculated using the whole LiDAR point cluster instead of the points inside the bounding box given by YOLO [52], since the whole cluster may not fit inside the bounding box. The bounding boxes produced by YOLO often do not cover 100% of the object. Furthermore, the object may be only partly visible to the camera. Hence, the true size of the object is measured using only LiDAR, since LiDAR has a 360 ° horizontal FOV. Lastly, the orientation of the bounding box is calculated as the angle between the $x$-axis of the point cloud and $x$-axis of the object.
4 Results

In this section, the experiments performed for the detector and their results are described. First, the experiments done to test the system are described. Second, the dataset used for testing is described. Third, the parameters used for the algorithm are given. Fourth, the results of the detection experiments are discussed. Fifth, the results are going to be evaluated. Lastly, the computational performance of the system is discussed.

4.1 Experiments

The experiments done in this work aim to measure the accuracy and computational time of the implemented multi-modal object detector. The goal of the experiments is to find out how accurately objects are found and how accurately their spatial location and size is estimated. Furthermore, the detector must fill the soft real-time requirements described in the Section 3.1. Hence, the computational time is measured throughout all experiments.

All experiments in this work are tested against nuScenes sample dataset, which is described in the Section 4.2. There are 24 different adjustable parameters in the algorithms used, thus the first step for experiments is to find best parameter combination for the system. Parameters were tuned by running a scene from the nuScenes dataset and using a grid search on different parameter combinations. Performance was measured with the PASCAL criterion [14] i.e. the intersection over union (IoU) between the 3D bounding boxes of the detections and the ground truth annotations of the dataset

$$\text{IoU} = \frac{D \cap A}{D \cup A},$$

where $D$ is the detection bounding box and $A$ is the annotated ground truth bounding box. Since, only ground truth available is the 3D bounding boxes, the individual components of the system were not tested by themselves.

All experiments were performed with a computer with Intel i7-7700HQ central processing unit (CPU) @ 2.80 GHz x8, 24 GB RAM and NVIDIA GeForce GTX 1050 Ti GPU. The parameters used, results and evaluation of the results are presented in Section 4.

4.2 Dataset

The experiments done in this work for the detector, were done using nuScenes open road dataset [5]. The dataset was selected, due to it containing multiple different weather conditions in different cities. The dataset is collected in Boston and Singapore. Furthermore, it contains also radar data, which could be used in future work. In contrast, KITTI dataset, which is widely used especially in autonomous driving research, has low diversity as stated by Feng et al. [15]. Even though the nuScenes dataset is developed for autonomous driving i.e. it contains only traffic data, it
is valid for testing the system, since the LiDAR based detection methods are not dependent on the type of data, as they are have not been taught to any specific data. Furthermore, the YOLOv3 [52] camera detector has been taught with large enough data set to be applicable to the target domain of the robot, as mentioned in Section 3.1.

The nuScenes dataset [5] consists of 1000 20 s scenes from different situations and environments. However, in this work only the sample dataset was used containing 10 different scenes from different situations and environments. The scenes have 32-beam LiDAR with 360° horizontal FOV and +10° to −30° vertical FOV, six cameras and five radars to have complete 360° vision, and the transformations for the sensor positions wrt base link with the calibration and distortion parameters of each camera. Sensor setup of the vehicles used in data gathering is presented in Figure 25. Furthermore, the sensors are synchronized on the hardware level. The cameras are triggered when the LiDAR sweeps the center point of the FOV of each camera.

![Sensor setup of nuScenes [5] data gathering vehicles.](image)

Figure 25: Sensor setup of nuScenes [5] data gathering vehicles.

Furthermore, the dataset [5] contains 1.4 million 3D bounding box annotations for 23 different object classes in the surroundings, including static and dynamic objects, visibility, activity and pose information, and maps of the environments. Annotations are given at 2 Hz frequency. Examples of annotations can be seen in Figures 26 and 27.
Figure 26: Example of annotated bounding boxes of a single frame in nuScenes [5] dataset.

Figure 27: Example of an annotated bounding box of a single object in a single frame in both image and point cloud in nuScenes [5] dataset.
4.3 Parameters

The parameters were found using grid search with the Pascal criterion [14] as the evaluation criterion for the best combination. The parameters found through iteration can be found in Tables 1 and 2. These parameters were used in all experiments. Explanation of the parameters can be found in Sections 3.2.1 and 3.2.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \alpha$</td>
<td>0.5</td>
<td>$^\circ$</td>
<td>Sector size</td>
</tr>
<tr>
<td>$B$</td>
<td>300</td>
<td></td>
<td>Number of bins in one sector</td>
</tr>
<tr>
<td>$r_1^{\text{min}}$</td>
<td>0.5</td>
<td>$m$</td>
<td>Distance to the first bin</td>
</tr>
<tr>
<td>$r_1$</td>
<td>0.1</td>
<td>$m$</td>
<td>Distance covered by the first bin</td>
</tr>
<tr>
<td>$r_B$</td>
<td>2.0</td>
<td>$m$</td>
<td>Distance covered by the last bin</td>
</tr>
<tr>
<td>$T_m$</td>
<td>10.0</td>
<td>$^\circ$</td>
<td>Maximum slope of a line</td>
</tr>
<tr>
<td>$T_{m_{\text{small}}}$</td>
<td>2.0</td>
<td>$^\circ$</td>
<td>Threshold for small slopes</td>
</tr>
<tr>
<td>$T_b$</td>
<td>0.5</td>
<td>$m$</td>
<td>Maximum absolute $y$ intercept for small slopes</td>
</tr>
<tr>
<td>$T_{\text{rmse}}$</td>
<td>0.5</td>
<td></td>
<td>Maximum RMSE of the line fit</td>
</tr>
<tr>
<td>$T_{d_{\text{prev}}}$</td>
<td>0.5</td>
<td>$m$</td>
<td>Maximum distance between two lines</td>
</tr>
<tr>
<td>$T_{d_{\text{ground}}}$</td>
<td>0.1</td>
<td>$m$</td>
<td>Maximum distance between a ground point and a line</td>
</tr>
</tbody>
</table>

Table 1: Ground removal parameters used

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \gamma$</td>
<td>2.0</td>
<td>$^\circ$</td>
<td>Sector size</td>
</tr>
<tr>
<td>$B$</td>
<td>100</td>
<td></td>
<td>Number of bins in one sector</td>
</tr>
<tr>
<td>$r_1^{\text{min}}$</td>
<td>0.5</td>
<td>$m$</td>
<td>Distance to the first bin</td>
</tr>
<tr>
<td>$r_1$</td>
<td>0.1</td>
<td>$m$</td>
<td>Distance covered by the first bin</td>
</tr>
<tr>
<td>$r_B$</td>
<td>2.0</td>
<td>$m$</td>
<td>Distance covered by the last bin</td>
</tr>
<tr>
<td>$T_{n3d}$</td>
<td>2.0</td>
<td></td>
<td>Minimum number of cells with big enough height difference for 3D clustering</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0.8</td>
<td>$^\circ$</td>
<td>Vertical FOV angle of the LiDAR</td>
</tr>
<tr>
<td>$T_z$</td>
<td>2.0</td>
<td></td>
<td>Minimum number of $\theta$ angles between two points in a cell for 3D clustering</td>
</tr>
<tr>
<td>$T_{\text{clustering}}$</td>
<td>0.5</td>
<td>$m$</td>
<td>Euclidean clustering distance threshold</td>
</tr>
<tr>
<td>$T_{\text{min_points}}$</td>
<td>3.0</td>
<td></td>
<td>Minimum number of points in a cluster</td>
</tr>
<tr>
<td>$T_{z_{\text{max_eff}}}$</td>
<td>0.1</td>
<td>$m$</td>
<td>Maximum point height difference in a cell to be filtered</td>
</tr>
<tr>
<td>$T_{z_{\text{min}}}$</td>
<td>0.0</td>
<td>$m$</td>
<td>Maximum point height in a cell to be filtered</td>
</tr>
</tbody>
</table>

Table 2: Foreground segmentation parameters used
4.4 Detection results

The detector was tested against 10 different scenes with different weather conditions from the nuScenes [5] sample dataset. The detector was tested only in 10 different scenes, since that was the size of the sample set. The results can be found in the Tables A1 - A12, which can be found in Appendix A, for different evaluation metrics. Furthermore, mA, mIoU, SE results across all datasets can be found plotted in Figures A1, A2 and A3 respectively, which can be found in Appendix A. The detector was tested with two different detection ranges: 40 m and 65 m, since the accuracy of both the point cloud clustering and YOLOv3-416 [52] deteriorate significantly after around 65 m. This is due to the limits imposed by the image resolution used in the DCNN, and the limits imposed by the LiDAR. The LiDAR vertical resolution is 2.0° and horizontal resolution from 0.1° - 0.4° [64]. Thus, at 65 m distance the vertical distance between two points is 2.27 m and the horizontal distance is up to 0.45 m making the detection of pedestrians, vehicles or machinery difficult. Furthermore, the other range, 40 m, was selected, due to it being reasonable detection range for objects in the domain of the robot. The areas, where the robot operates, are not huge as they are mostly warehouses and stockyards and the speed of the robot is around 1 - 2 m/s. Hence, the 40 m detection distance is enough for most of the situations.

The ground truth data [5] that the detector was tested against provided visibility information of the objects, which was used to filter the ground truth annotations. The detector was tested with two visibility thresholds: 40% and 80%, where the threshold means that all objects that have visibility over the threshold are taken into account. For example, 80% means that only objects with the visibility of 80 - 100% are taken into account. Hence, 80% threshold contains only easy-to-detect objects and 40% threshold contains easy and medium objects, but not the hardest ones. The hardest ones were filtered out since, for the detector implemented in this work, it is unrealistic to detect those.

Furthermore, only the ground truth annotations of pedestrians, bicycles and vehicles were taken into account in the evaluation of the detector. The ground truth data contained annotations of many static objects such as traffic lights, light poles and road blocks, but they are not classified by YOLOv3 [52] and they are not of interest as they are not dynamic objects. Hence, they were left out from the evaluation.

The results are presented with three different evaluation criteria, since the frequency of the detector is around $f_d \approx 10$ Hz and the ground truth annotations are given at frequency of $f_{gt} = 2$ Hz. Hence, the detection results are presented as strict, combined and mean results. The strict results are calculated only from frames with matched timestamps i.e. detections frames are used only with 2 Hz frequency and rest of the frames as discarded. The idea behind this approach is that the detector should be able to detect all objects in each frame at any given time, and thus only the detection frames that have matching timestamps with the ground truth are taken into account. The combined results were combined from all detections around the ground truth timestamp. If the ground truth frame timestamp is $T_{gt}$,
then all detection frames with timestamp $T_d$, which fulfill $-0.25 \leq T_d - T_{gt} < 0.25$, were combined to one frame. The combined frame was then compared to the ground truth data. The idea behind this approach is to accept the detection, if it has been detected at least once within the $T_{gt}$, since this can be enough for the target tracker of the robot. The more strict mean results were calculated by taking the average of the detection frames across the $T_{gt}$. This means that, if the object is detected once in 5 frames within the $T_{gt}$, it is calculated as $\frac{1}{5}$ detection. The idea behind this approach is between the previous two. All objects should be detected in each frame. However, as the first method discards the detections from other timestamps completely, this one takes them into account. Thus, the results are presented with three evaluation criteria for both distances and visibility thresholds.

On the rows of the tables is the results for the scenes, numbered from 1 to 10. The $GT$ represents the ground truth object count i.e. how many objects were annotated in the scenes. The $MD$ stands for matched detections i.e. detections that had a matching ground truth detection according to the Pascal [14] criterion. The usual Pascal criterion is counted with $IoU > 0.5$. However, the IoU criterion was used with $IoU > 0.1$ to count the detection as a match, since the detector underestimates the sizes of the bounding boxes, as discussed in more detail in Section 4.5. The $TD$ is the total detection count of the detector i.e. matched detections and false positives. The $FP$ stands for false positives i.e. not matched detections. The $mA$ represents the mean detection accuracy i.e. the percentage of the matched detections out of all ground truth detections. The $mIoU$ means the average of the IoU of the bounding boxes, Equation 18 i.e. the Pascal criterion. Furthermore, this is the metric that is mostly used in the literature for evaluation of the geometric accuracy. The $CE$ represents the center mean absolute error in meters ($m$), which is the average absolute deviation of the center point of the bounding box from the ground truth value. Lastly, the $SE$ is the mean size error measured as percentages of the volumes of the bounding boxes between the measurement and the ground truth, which represents the average error in the bounding box volume. The size error is calculated

$$SE = \frac{V_D - V_{GT}}{V_{GT}} \times 100 . \quad (19)$$

Moreover, the sign of the value represents the direction of the error i.e. positive values mean bigger and negative values smaller bounding box than the ground truth.

Figures 28 and 29 represent the average strict, combined and mean values and their standard deviations for $mA$ and $mIoU$ respectively across all datasets. Furthermore, Figures 30 and 31 represent the average strict, combined and mean values and their standard deviations for $mA$ and $mIoU$ across all datasets with the easiest and the hardest evaluation criteria, 40 $m$ distance and $> 80\%$ visibility, and 65 $m$ distance and $> 40\%$ visibility respectively.

The categories of the objects are not taken into account in the results due to the YOLOv3 [52] being the only source of classification in this work and it has not being trained with the dataset. The classes provided by YOLO mostly likely would not match the nuScenes [5] categories. The choice of words used for classes can differ enough to not make it applicable.
Figure 28: Average strict, combined and mean \( mA \) values across all datasets and their standard deviations \( \sigma \).

Figure 29: Average strict, combined and mean \( mIoU \) values across all datasets and their standard deviations \( \sigma \).
Figure 30: Average strict, combined and mean mA values across all datasets and their standard deviations $\sigma$ for the easiest evaluation criteria, 40 m distance and $> 80\%$ visibility, and the hardest evaluation criteria, 65 m and $> 40\%$.

Figure 31: Average strict, combined and mean mIoU values across all datasets and their standard deviations $\sigma$ for the easiest evaluation criteria, 40 m distance and $> 80\%$ visibility, and the hardest evaluation criteria, 65 m and $> 40\%$. 
4.5 Evaluation of the results

There are two significant things in the result Tables to be evaluated. The mA and mIoU columns, since they represent the accuracy of the detector. The mA represents, how accurately objects are found and mIoU represents, how accurately the spatial locations and volumes of the bounding boxes are estimated.

The mA values range from 14% to 36% with mean of 22% and standard deviation $\sigma = 6\%$ for strict values, from 61% to 100% with mean of 89% and $\sigma = 18\%$ for combined values and from 18% to 50% with mean of 30% and $\sigma = 9\%$ for the mean values across all datasets. There is a clear difference between the three evaluation criteria, which indicates that the detector is able to find most of the objects around the vehicle during the $T_{TE}$ period. However, the detector is clearly not able to find all objects or even the same objects in every single frame. This is illustrated in Figure 28, which represents the values and their. In the graph the combined values are significantly better than the mean or the strict values. Furthermore, the mean values are better than the strict values across all datasets. From target tracking perspective the 80% accuracy at 2 Hz rate is enough for many applications, where the targets or the system is moving at a slow pace. However, for traffic detection, where the vehicle speeds are over $40 \frac{km}{h}$ the rate of 2 Hz is significantly too slow as a vehicle moving at $40 \frac{km}{h}$ moves 5.5 m in 0.5 s. Furthermore, from detector perspective the results have room for improvement, since an ideal detector would detect all objects in every single frame.

Furthermore, the effects of distance and visibility changes are significant, which can be seen by comparing the results for different ranges and visibilities. The hardest criteria are $>40\%$ visibility and 65 m range, since it contains the furthest and the most occluded objects. Furthermore, the easiest criteria is the $>80\%$ visibility and 40 m range. The average detection accuracy for the hardest criteria across all datasets is 21.8%, 80.9% and 27.7% for the strict, the combined and the mean frames respectively. On the other hand, the respective accuracies are 23.2% 90.2% and 33.7% for the easiest criteria. This can be seen in Figure 30 as the easiest criteria has the best results in almost all situations across all datasets. Hence, the detector clearly struggles with occluded objects and with longer distance. This is illustrated in Figure 32.

The missed detections are most likely due to the LiDAR clustering not being accurate enough to separate close objects from each other. Moreover, the YOLOv3 is not trained with nuScenes dataset, which could cause some missed detections as the DCNNs have generalization issues when changing domains or environments, as discussed in Section 2. Furthermore, the deterioration of the results with longer distances is most likely due to the resolution of image not being high enough and the dispersion of the LiDAR points. Since the input resolution of the DNN is 416x416 pixels, further object are increasingly difficult to detect correctly by the YOLOv3. Further, the dispersion of LiDAR beams is most likely an even more significant factor, since a minimum number of points is needed for a cluster to be taken into account. The LiDAR points far away are also more likely to be classified in a bigger cluster in their vicinity. For example, a pedestrian near a wall might get clustered to same
Figure 32: Vehicle closer to the sensors is clustered well, while the truck further away from the sensors have fewer points associated to its cluster.

cluster as the wall. This is illustrated in Figures 33 and 34, which represent a failed clustering and successful clustering. Moreover, occluded objects are difficult for the detector for the same reason as far away objects: the number of points is lower. Since the system uses a traditional clustering method without heuristics or machine learning, it is highly difficult for the detector to estimate occluded objects.

Figure 33: Two vehicles clustered as one and associated as the smaller vehicle in the front.
The results could be made better with a higher resolution image used for the DCNN and with a higher resolution LiDAR. However, as discussed in Section 3, the performance time of the higher resolution YOLOv3 is significantly higher than the 416x416 version, hence, it is not applicable to this work. Moreover, the higher resolution LiDARs are significantly more expensive than lower resolution ones.

There are a significant number of false positives in the detections, which can be seen in the FP column of the Tables A1 - A12 in Appendix A. This is affected largely by the fact that YOLOv3 [52] detects also other objects in addition to the objects selected for the ground truth data and the point cloud clustering does not filter out clusters by any means except by the size of the cluster. Hence, there are detections of other objects that are counted as false positives in the results. Thus, it is not possible to evaluate the amount of real false positives.

The mIoU results do not show as big of a difference between different evaluation criteria as mA results. The results range across datasets from 43% to 61% with the average of 54% for the strict frames, from 39% to 53% with the average of 45.1% for the combined frames and from 39% to 58% with the average of 48% for the mean frames. In fact, the results are highly similar for each criteria. This can be seen in Figure 29 as the differences between different evaluation criteria are small compared to the differences in mA values. Furthermore, the difference between the ranges and visibilities are even smaller with the hardest criteria average accuracies being 44.9% and 47.9% for combined and mean frames respectively and the easiest criteria average accuracies being 45.2% and 48.2% respectively. For the strict frames easiest and hardest criteria had the same average accuracies of 54% for both. This can be seen in Figure 31. Interestingly, for the mIoU the strict frames had the best results while the combined frames had the worst results, which is completely opposite of the
mA results.

However, the results are overall a bit underwhelming as the mIoU represents the average bounding box IoU values, which means that even the best performing dataset had average IoU of 58%. In comparison to the nuScenes [5] leaderboard, the best detector has mIoU of 75.3% with all objects included without visibility or range restrictions. Furthermore, the IoU > 0.1 criterion was used for calculating mA in this work, since the detector on average underestimates the size of the bounding box by a significant margin. This can be seen from SE columns of the results Tables, which represent the average size error in the dataset, and in Figure A3, which can be found in Appendix A, which represents the SE values across all datasets. The mean of the average bounding box size errors across datasets is $-18.18\%$ with $\sigma = 73.03\%$ with the values ranging from $-91.62\%$ to $191.91\%$. The deviation of the values of the average difference in bounding box size is quite high. Most of the datasets had the bounding box size underestimated on average and the values were positive only in three different scenes out of all the ten scenes, which can be seen in Figure A3. This could be caused by the scenes having high amount of pedestrians, which are clustered with a wall or other structure next to them.

The mIoU, CE and SE results are most likely caused by the way that the bounding boxes are calculated. As mentioned in Section 2.3.1, the bounding boxes are formed from the cluster with PCA. Since, the LiDAR only sees objects from one direction at a time, it is highly difficult to estimate the true size of the object from the cluster without heuristics. There are no machine learning methods, which would learn the heuristic object sizes from data, or human engineered heuristics on the system, thus, the system underestimates the bounding box sizes constantly. For example, if the LiDAR only sees the back of a vehicle, the bounding box is calculated to be flat, thus, the length of the vehicle gets underestimated. This problem is illustrated in Figure 35, which shows a BEV view of a vehicle seen by the LiDAR only diagonally from the side and the ellipsoid created on top of with PCA. Furthermore, this largely explains also the CE values, which range from 0.99 m to 1.95 m with the average of 1.39 m and $\sigma = 0.28$ m. As the bounding box dimensions are underestimated, the center point moves with the reduction of bounding box volume, thus, creating errors in the measurements. This causes problems in the target tracking side, since target trackers usually track pointlike objects i.e. the center point of the objects. Only solution to this problem is to use some heuristics, whether they are human engineered or learned directly from data.

Overall, the detector fulfills the requirements for robustness by fusing two different sensing modalities and was tested with data scenes of different weather conditions and the detector performed roughly with similar results across all datasets. On the other hand, the detection accuracy has a lot of room for improvement as it does not perform nearly as well as the state-of-the-art detectors. However, the detector had restrictions in the methods used imposed by the application and there is no data from the domain of the robot to train machine learning methods. Thus, the performance can not be expected to be on par with the state-of-the-art detectors. Hence, the detector performed well enough considering the restrictions.
Figure 35: PCA of a 3D vehicle point cluster from a bird’s-eye view. Blue points represent the LiDAR points from a car and the purple ellipsoid represents the results of the PCA of the points. The dimensions of the ellipsoid ellipsoid compose of the eigenvectors and eigenvalues of the point cluster.

However, there is a lot room for improvement in the spatial location and size estimations of the objects, which could be improved with some heuristics. For example, a SVM [63] could be implemented to classify the clusters before the sensor fusion with the camera. Initial classification could be used for more accurate size estimation of the object, as the real size of the object could be estimated by the SVM.

Based on the results of the experiments, it is difficult to generalize how well the detector would perform on the hardware of the robot as the robot has sparser LiDAR than the test dataset. This most likely will reduce the accuracy of the detector especially at longer distances. However, the robot will move at a slower pace than a car and the area is smaller and more restricted than a traffic setup. Hence, the distance required for an object to be detected is smaller than in traffic environments. Furthermore, the robot has a less powerful computer than the test setup, which causes the computational time to be longer than in the test setup. Moreover, the domain is slightly different than the dataset as the objects in the domain are different. This poses some questions such as is the fork of a forklift filtered out as ground points in down position. Thus, the detector must be tested on the actual robot in the real environment before it can be evaluated for the logistics purposes.
4.6 Computational performance evaluation

As one of the criterion for object detection system stated in Section 2 was real-time, the performance of the detector must be evaluated by the computational time of the system. Most of the computational time of the detector is used in the YOLOv3 [52] detection pipeline and the LiDAR point cloud foreground segmentation. The remaining parts of the algorithm perform in few milliseconds, hence, they are not significant for performance evaluation. The LiDAR rectification is done before the detector receives the point cloud, hence, it is not calculated in the performance time of the system. The performance time was measured across all experiments. Minimum, maximum and average run times in milliseconds of the YOLOv3 and the foreground segmentation algorithm can be seen in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>YOLOv3 (ms)</th>
<th>Foreground segmentation (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>75.54</td>
<td>68.89</td>
</tr>
<tr>
<td>Max</td>
<td>95.70</td>
<td>122.93</td>
</tr>
<tr>
<td>Mean</td>
<td>80.24</td>
<td>86.40</td>
</tr>
</tbody>
</table>

Table 3: Computational times of the YOLOv3 [52] and the foreground segmentation algorithm in milliseconds.

The YOLOv3 [52] takes roughly the same time to process the images every time, which is just under the real-time requirements stated in Section 2, even with the maximum performance time. On the other hand, the LiDAR foreground segmentation performance deviates more heavily, which is mostly dependent on the amount of foreground points needed to be clustered. The ground removal part of the algorithm takes on average 28.58 ms and the foreground segmentation part takes on average 57.82 ms. Most of the time in the algorithm was used by the 2D grid clustering part. This is dependent on $\Delta \gamma$ and $B$ parameters in the foreground segmentation part, since these control the size of the used grid. The complexity of the BFS algorithm used in the clustering is $O(V + E)$, where the $V$ is the amount of the cells in the grid and $E$ is the amount of vertices between the cells, which in case the case of 2D grid is 8 per cell. As the amount of vertices is dependent on the amount of the grid cells, the complexity of the algorithm is only dependent on the amount of cells. Hence, it is trivial to adjust the foreground segmentation to work in real time by adjusting the grid size. However, it will decrease the detection accuracy as a sparser grid is not as accurate as a denser grid. Overall, the detector fulfills the soft real-time requirements as it was able to operate most of the time faster than the maximum performance time required in the test setup.
5 Conclusions

In this work, a generic dynamic object detector for autonomous outdoor logistics robot was implemented. The system utilized the sensor fusion of a state-of-the-art DCNN image detector and 3D range data produced by a 3D LiDAR. The performance of the detector was evaluated by the computational time, the accuracy and the robustness of the detector. The system was tested with a public nuScenes [5] dataset.

Autonomous mobile robots require an accurate perception of the surrounding world for safe and reliable operation. The perception system often fuses different sensor modalities for robust and accurate 3D perception of the environment. Most often used sensors are cameras and 3D LiDARs, which were also utilized in this work. State-of-the-art generic object detection methods utilize deep neural networks, which outperform traditional methods by a significant margin. However, due to the domain constraints fused DCNN approaches were not applicable in this work. Hence, the system utilized the fusion of a state-of-the-art DCNN image detector with a traditional foreground segmentation method for 3D point clouds. The foreground segmentation of the 3D point clouds removed the ground points from the point cloud and clustered the remaining points as foreground objects. These clusters were associated with the bounding boxes created by the DCNN detector by utilizing GNN data-association method.

The detector could perform in near real-time in the experiments fulfilling the soft real-time requirements. On the other hand, the detection precision and spatial location and size estimation were not nearly as good as a state-of-the-art multi-modal object detector. However, this was expected as the robots domain and hardware imposed restrictions on the methods used. Hence, the results were acceptable the restrictions considering even though there is a lot of room for improvement.

The system was evaluated against an open road dataset, and thus conclusions can not be directly made about the systems applicability to the actual robot. However, without the robot or data it is nearly impossible to evaluate the performance of the system on the robot. Hence, the next step in future work is to apply the system to the actual hardware.

The system could be improved in future work by utilizing heuristics on the size estimation of the objects, which could be created by machine learning methods or by human engineering, if machine learning methods are not applicable. For example, a SVM [63] could be used for initial classification and size estimation of the objects before the fusion with the image data. Furthermore, the SVM could help to classify objects outside the FOV of the camera. Moreover, the system could be improved by properly handling the overlap of the camera images as they were not handled at all in this work.
References


[37] Li, Y., Bu, R., Sun, M., and Chen, B. PointCNN: Convolution On χ-Transformed Points.


# Detection results

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>575</td>
<td>114</td>
<td>134</td>
<td>20</td>
<td>20</td>
<td>59</td>
<td>1.06</td>
<td>-27.55</td>
</tr>
<tr>
<td>2</td>
<td>291</td>
<td>45</td>
<td>53</td>
<td>8</td>
<td>15</td>
<td>57</td>
<td>1.16</td>
<td>-87.77</td>
</tr>
<tr>
<td>3</td>
<td>303</td>
<td>67</td>
<td>98</td>
<td>31</td>
<td>22</td>
<td>47</td>
<td>1.64</td>
<td>-51.26</td>
</tr>
<tr>
<td>4</td>
<td>796</td>
<td>138</td>
<td>214</td>
<td>76</td>
<td>17</td>
<td>58</td>
<td>0.98</td>
<td>86.43</td>
</tr>
<tr>
<td>5</td>
<td>732</td>
<td>184</td>
<td>228</td>
<td>44</td>
<td>25</td>
<td>51</td>
<td>1.58</td>
<td>-77.29</td>
</tr>
<tr>
<td>6</td>
<td>220</td>
<td>57</td>
<td>75</td>
<td>18</td>
<td>26</td>
<td>51</td>
<td>1.45</td>
<td>-30.14</td>
</tr>
<tr>
<td>7</td>
<td>688</td>
<td>106</td>
<td>120</td>
<td>14</td>
<td>15</td>
<td>61</td>
<td>0.99</td>
<td>-62.36</td>
</tr>
<tr>
<td>8</td>
<td>506</td>
<td>101</td>
<td>116</td>
<td>15</td>
<td>20</td>
<td>43</td>
<td>1.21</td>
<td>191.91</td>
</tr>
<tr>
<td>9</td>
<td>1131</td>
<td>229</td>
<td>257</td>
<td>28</td>
<td>20</td>
<td>59</td>
<td>1.44</td>
<td>-4.32</td>
</tr>
<tr>
<td>10</td>
<td>489</td>
<td>177</td>
<td>187</td>
<td>10</td>
<td>36</td>
<td>53</td>
<td>1.16</td>
<td>-66.38</td>
</tr>
</tbody>
</table>

Table A1: Strict results for 40 m range with ground truth object visibility > 40%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>575</td>
<td>386</td>
<td>659</td>
<td>273</td>
<td>67</td>
<td>50</td>
<td>0.97</td>
<td>-39.21</td>
</tr>
<tr>
<td>2</td>
<td>291</td>
<td>291</td>
<td>403</td>
<td>112</td>
<td>100</td>
<td>39</td>
<td>1.49</td>
<td>-91.62</td>
</tr>
<tr>
<td>3</td>
<td>303</td>
<td>293</td>
<td>528</td>
<td>235</td>
<td>97</td>
<td>47</td>
<td>1.63</td>
<td>-82.48</td>
</tr>
<tr>
<td>4</td>
<td>796</td>
<td>681</td>
<td>1582</td>
<td>901</td>
<td>86</td>
<td>43</td>
<td>1.20</td>
<td>-73.42</td>
</tr>
<tr>
<td>5</td>
<td>732</td>
<td>619</td>
<td>1242</td>
<td>623</td>
<td>85</td>
<td>39</td>
<td>1.86</td>
<td>-62.56</td>
</tr>
<tr>
<td>6</td>
<td>220</td>
<td>220</td>
<td>471</td>
<td>251</td>
<td>100</td>
<td>43</td>
<td>1.59</td>
<td>-41.85</td>
</tr>
<tr>
<td>7</td>
<td>688</td>
<td>464</td>
<td>756</td>
<td>292</td>
<td>67</td>
<td>48</td>
<td>1.14</td>
<td>-43.45</td>
</tr>
<tr>
<td>8</td>
<td>506</td>
<td>344</td>
<td>467</td>
<td>123</td>
<td>68</td>
<td>40</td>
<td>1.35</td>
<td>-104.69</td>
</tr>
<tr>
<td>9</td>
<td>1131</td>
<td>814</td>
<td>1361</td>
<td>547</td>
<td>72</td>
<td>46</td>
<td>1.40</td>
<td>-36.79</td>
</tr>
<tr>
<td>10</td>
<td>489</td>
<td>489</td>
<td>633</td>
<td>144</td>
<td>100</td>
<td>53</td>
<td>1.00</td>
<td>-69.16</td>
</tr>
</tbody>
</table>

Table A2: Combined results for 40 m range with ground truth object visibility > 40%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>575</td>
<td>147.00</td>
<td>192.09</td>
<td>45.09</td>
<td>26</td>
<td>56</td>
<td>1.02</td>
<td>-39.87</td>
</tr>
<tr>
<td>2</td>
<td>291</td>
<td>69.12</td>
<td>103.40</td>
<td>34.27</td>
<td>24</td>
<td>45</td>
<td>1.48</td>
<td>-88.38</td>
</tr>
<tr>
<td>3</td>
<td>303</td>
<td>86.97</td>
<td>142.03</td>
<td>55.06</td>
<td>29</td>
<td>43</td>
<td>1.78</td>
<td>-80.55</td>
</tr>
<tr>
<td>4</td>
<td>796</td>
<td>235.55</td>
<td>395.43</td>
<td>159.87</td>
<td>30</td>
<td>49</td>
<td>1.10</td>
<td>71.79</td>
</tr>
<tr>
<td>5</td>
<td>732</td>
<td>255.03</td>
<td>394.47</td>
<td>139.43</td>
<td>35</td>
<td>45</td>
<td>1.79</td>
<td>-66.02</td>
</tr>
<tr>
<td>6</td>
<td>220</td>
<td>73.08</td>
<td>126.77</td>
<td>53.69</td>
<td>33</td>
<td>47</td>
<td>1.59</td>
<td>-3.21</td>
</tr>
<tr>
<td>7</td>
<td>688</td>
<td>131.57</td>
<td>180.76</td>
<td>49.19</td>
<td>19</td>
<td>54</td>
<td>1.07</td>
<td>-42.88</td>
</tr>
<tr>
<td>8</td>
<td>506</td>
<td>107.50</td>
<td>138.42</td>
<td>30.91</td>
<td>21</td>
<td>39</td>
<td>1.39</td>
<td>134.33</td>
</tr>
<tr>
<td>9</td>
<td>1131</td>
<td>371.69</td>
<td>505.97</td>
<td>134.28</td>
<td>33</td>
<td>50</td>
<td>1.45</td>
<td>-22.96</td>
</tr>
<tr>
<td>10</td>
<td>489</td>
<td>241.65</td>
<td>276.28</td>
<td>34.63</td>
<td>49</td>
<td>49</td>
<td>1.01</td>
<td>-64.14</td>
</tr>
</tbody>
</table>

Table A3: Mean results for 40 m range with ground truth object visibility > 40%.
<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>648</td>
<td>114</td>
<td>135</td>
<td>21</td>
<td>18</td>
<td>59</td>
<td>1.06</td>
<td>-27.55</td>
</tr>
<tr>
<td>2</td>
<td>329</td>
<td>45</td>
<td>54</td>
<td>9</td>
<td>14</td>
<td>57</td>
<td>1.16</td>
<td>-87.77</td>
</tr>
<tr>
<td>3</td>
<td>318</td>
<td>67</td>
<td>101</td>
<td>34</td>
<td>21</td>
<td>47</td>
<td>1.64</td>
<td>-51.26</td>
</tr>
<tr>
<td>4</td>
<td>830</td>
<td>138</td>
<td>222</td>
<td>84</td>
<td>17</td>
<td>58</td>
<td>0.98</td>
<td>86.43</td>
</tr>
<tr>
<td>5</td>
<td>790</td>
<td>185</td>
<td>238</td>
<td>53</td>
<td>23</td>
<td>51</td>
<td>1.58</td>
<td>-77.41</td>
</tr>
<tr>
<td>6</td>
<td>255</td>
<td>58</td>
<td>76</td>
<td>18</td>
<td>23</td>
<td>50</td>
<td>1.46</td>
<td>-31.33</td>
</tr>
<tr>
<td>7</td>
<td>722</td>
<td>106</td>
<td>124</td>
<td>18</td>
<td>15</td>
<td>61</td>
<td>0.99</td>
<td>-62.36</td>
</tr>
<tr>
<td>8</td>
<td>517</td>
<td>101</td>
<td>117</td>
<td>16</td>
<td>20</td>
<td>43</td>
<td>1.21</td>
<td>191.91</td>
</tr>
<tr>
<td>9</td>
<td>1168</td>
<td>229</td>
<td>259</td>
<td>30</td>
<td>20</td>
<td>59</td>
<td>1.44</td>
<td>-4.32</td>
</tr>
<tr>
<td>10</td>
<td>565</td>
<td>180</td>
<td>191</td>
<td>11</td>
<td>32</td>
<td>52</td>
<td>1.19</td>
<td>-64.91</td>
</tr>
</tbody>
</table>

Table A4: Strict results for 65 m range with ground truth object visibility > 40%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>648</td>
<td>397</td>
<td>699</td>
<td>302</td>
<td>61</td>
<td>50</td>
<td>1.01</td>
<td>-34.94</td>
</tr>
<tr>
<td>2</td>
<td>329</td>
<td>307</td>
<td>438</td>
<td>131</td>
<td>93</td>
<td>39</td>
<td>1.49</td>
<td>-91.57</td>
</tr>
<tr>
<td>3</td>
<td>318</td>
<td>294</td>
<td>558</td>
<td>264</td>
<td>92</td>
<td>47</td>
<td>1.63</td>
<td>-82.53</td>
</tr>
<tr>
<td>4</td>
<td>830</td>
<td>686</td>
<td>1810</td>
<td>1124</td>
<td>83</td>
<td>43</td>
<td>1.20</td>
<td>72.47</td>
</tr>
<tr>
<td>5</td>
<td>790</td>
<td>627</td>
<td>1371</td>
<td>744</td>
<td>79</td>
<td>39</td>
<td>1.87</td>
<td>-63.39</td>
</tr>
<tr>
<td>6</td>
<td>255</td>
<td>255</td>
<td>500</td>
<td>245</td>
<td>100</td>
<td>43</td>
<td>1.59</td>
<td>-42.66</td>
</tr>
<tr>
<td>7</td>
<td>722</td>
<td>466</td>
<td>790</td>
<td>324</td>
<td>65</td>
<td>48</td>
<td>1.14</td>
<td>-43.63</td>
</tr>
<tr>
<td>8</td>
<td>517</td>
<td>346</td>
<td>484</td>
<td>138</td>
<td>67</td>
<td>39</td>
<td>1.34</td>
<td>103.51</td>
</tr>
<tr>
<td>9</td>
<td>1168</td>
<td>826</td>
<td>1500</td>
<td>674</td>
<td>71</td>
<td>48</td>
<td>1.41</td>
<td>-37.50</td>
</tr>
<tr>
<td>10</td>
<td>565</td>
<td>553</td>
<td>717</td>
<td>164</td>
<td>98</td>
<td>53</td>
<td>1.06</td>
<td>-65.75</td>
</tr>
</tbody>
</table>

Table A5: Combined results for 65 m range with ground truth object visibility > 40%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>648</td>
<td>148.88</td>
<td>198.65</td>
<td>49.77</td>
<td>23</td>
<td>57</td>
<td>1.03</td>
<td>-38.43</td>
</tr>
<tr>
<td>2</td>
<td>329</td>
<td>70.12</td>
<td>110.56</td>
<td>40.44</td>
<td>21</td>
<td>45</td>
<td>1.48</td>
<td>-88.01</td>
</tr>
<tr>
<td>3</td>
<td>318</td>
<td>87.29</td>
<td>149.15</td>
<td>61.87</td>
<td>27</td>
<td>43</td>
<td>1.79</td>
<td>-80.61</td>
</tr>
<tr>
<td>4</td>
<td>830</td>
<td>236.58</td>
<td>431.57</td>
<td>194.99</td>
<td>29</td>
<td>50</td>
<td>1.10</td>
<td>71.23</td>
</tr>
<tr>
<td>5</td>
<td>790</td>
<td>257.25</td>
<td>422.65</td>
<td>165.40</td>
<td>33</td>
<td>45</td>
<td>1.80</td>
<td>-65.88</td>
</tr>
<tr>
<td>6</td>
<td>255</td>
<td>74.40</td>
<td>132.23</td>
<td>57.83</td>
<td>29</td>
<td>46</td>
<td>1.59</td>
<td>-4.90</td>
</tr>
<tr>
<td>7</td>
<td>722</td>
<td>131.79</td>
<td>186.04</td>
<td>54.25</td>
<td>18</td>
<td>54</td>
<td>1.07</td>
<td>-42.95</td>
</tr>
<tr>
<td>8</td>
<td>517</td>
<td>107.96</td>
<td>141.81</td>
<td>33.85</td>
<td>21</td>
<td>39</td>
<td>1.39</td>
<td>133.35</td>
</tr>
<tr>
<td>9</td>
<td>1168</td>
<td>374.15</td>
<td>539.54</td>
<td>165.39</td>
<td>32</td>
<td>51</td>
<td>1.45</td>
<td>-23.41</td>
</tr>
<tr>
<td>10</td>
<td>565</td>
<td>249.35</td>
<td>297.34</td>
<td>48.00</td>
<td>44</td>
<td>49</td>
<td>1.05</td>
<td>-61.65</td>
</tr>
</tbody>
</table>

Table A6: Mean results for 65 m range with ground truth object visibility > 40%.
<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>424</td>
<td>109</td>
<td>134</td>
<td>25</td>
<td>26</td>
<td>58</td>
<td>1.08</td>
<td>-27.93</td>
</tr>
<tr>
<td>2</td>
<td>273</td>
<td>45</td>
<td>53</td>
<td>8</td>
<td>16</td>
<td>57</td>
<td>1.16</td>
<td>-87.77</td>
</tr>
<tr>
<td>3</td>
<td>265</td>
<td>66</td>
<td>98</td>
<td>32</td>
<td>25</td>
<td>48</td>
<td>1.63</td>
<td>-50.66</td>
</tr>
<tr>
<td>4</td>
<td>688</td>
<td>131</td>
<td>214</td>
<td>83</td>
<td>19</td>
<td>60</td>
<td>0.96</td>
<td>95.96</td>
</tr>
<tr>
<td>5</td>
<td>476</td>
<td>159</td>
<td>228</td>
<td>69</td>
<td>33</td>
<td>53</td>
<td>1.68</td>
<td>-75.38</td>
</tr>
<tr>
<td>6</td>
<td>188</td>
<td>48</td>
<td>75</td>
<td>27</td>
<td>26</td>
<td>50</td>
<td>1.47</td>
<td>-19.98</td>
</tr>
<tr>
<td>7</td>
<td>595</td>
<td>102</td>
<td>120</td>
<td>18</td>
<td>17</td>
<td>61</td>
<td>0.98</td>
<td>-61.35</td>
</tr>
<tr>
<td>8</td>
<td>470</td>
<td>96</td>
<td>116</td>
<td>20</td>
<td>20</td>
<td>43</td>
<td>1.25</td>
<td>187.99</td>
</tr>
<tr>
<td>9</td>
<td>517</td>
<td>144</td>
<td>257</td>
<td>113</td>
<td>28</td>
<td>58</td>
<td>1.59</td>
<td>39.25</td>
</tr>
<tr>
<td>10</td>
<td>486</td>
<td>177</td>
<td>187</td>
<td>10</td>
<td>36</td>
<td>53</td>
<td>1.17</td>
<td>-70.65</td>
</tr>
</tbody>
</table>

Table A7: Strict results for 40 m range with ground truth object visibility > 80%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>424</td>
<td>321</td>
<td>659</td>
<td>338</td>
<td>76</td>
<td>52</td>
<td>0.99</td>
<td>-37.90</td>
</tr>
<tr>
<td>2</td>
<td>273</td>
<td>273</td>
<td>403</td>
<td>130</td>
<td>100</td>
<td>39</td>
<td>1.50</td>
<td>-91.49</td>
</tr>
<tr>
<td>3</td>
<td>265</td>
<td>265</td>
<td>528</td>
<td>263</td>
<td>100</td>
<td>47</td>
<td>1.63</td>
<td>-82.17</td>
</tr>
<tr>
<td>4</td>
<td>688</td>
<td>622</td>
<td>1582</td>
<td>960</td>
<td>90</td>
<td>44</td>
<td>1.19</td>
<td>88.88</td>
</tr>
<tr>
<td>5</td>
<td>476</td>
<td>470</td>
<td>1242</td>
<td>772</td>
<td>99</td>
<td>42</td>
<td>1.92</td>
<td>-64.33</td>
</tr>
<tr>
<td>6</td>
<td>188</td>
<td>188</td>
<td>471</td>
<td>283</td>
<td>100</td>
<td>43</td>
<td>1.65</td>
<td>-41.29</td>
</tr>
<tr>
<td>7</td>
<td>595</td>
<td>443</td>
<td>756</td>
<td>313</td>
<td>74</td>
<td>48</td>
<td>1.13</td>
<td>-46.18</td>
</tr>
<tr>
<td>8</td>
<td>470</td>
<td>331</td>
<td>467</td>
<td>136</td>
<td>70</td>
<td>40</td>
<td>1.39</td>
<td>105.40</td>
</tr>
<tr>
<td>9</td>
<td>517</td>
<td>483</td>
<td>1361</td>
<td>878</td>
<td>93</td>
<td>44</td>
<td>1.50</td>
<td>-17.00</td>
</tr>
<tr>
<td>10</td>
<td>486</td>
<td>486</td>
<td>633</td>
<td>147</td>
<td>100</td>
<td>53</td>
<td>1.00</td>
<td>-70.59</td>
</tr>
</tbody>
</table>

Table A8: Combined results for 40 m range with ground truth object visibility > 80%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>424</td>
<td>132.51</td>
<td>192.09</td>
<td>59.57</td>
<td>31</td>
<td>58</td>
<td>1.04</td>
<td>-40.42</td>
</tr>
<tr>
<td>2</td>
<td>273</td>
<td>67.51</td>
<td>103.40</td>
<td>35.88</td>
<td>25</td>
<td>45</td>
<td>1.48</td>
<td>-88.21</td>
</tr>
<tr>
<td>3</td>
<td>265</td>
<td>85.14</td>
<td>142.03</td>
<td>56.89</td>
<td>32</td>
<td>44</td>
<td>1.78</td>
<td>-80.31</td>
</tr>
<tr>
<td>4</td>
<td>688</td>
<td>222.10</td>
<td>395.43</td>
<td>173.32</td>
<td>32</td>
<td>50</td>
<td>1.08</td>
<td>81.52</td>
</tr>
<tr>
<td>5</td>
<td>476</td>
<td>211.99</td>
<td>394.47</td>
<td>182.47</td>
<td>45</td>
<td>48</td>
<td>1.85</td>
<td>-65.40</td>
</tr>
<tr>
<td>6</td>
<td>188</td>
<td>63.75</td>
<td>126.77</td>
<td>63.02</td>
<td>34</td>
<td>46</td>
<td>1.63</td>
<td>7.45</td>
</tr>
<tr>
<td>7</td>
<td>595</td>
<td>126.32</td>
<td>180.76</td>
<td>54.44</td>
<td>21</td>
<td>54</td>
<td>1.07</td>
<td>-44.66</td>
</tr>
<tr>
<td>8</td>
<td>470</td>
<td>103.53</td>
<td>138.42</td>
<td>34.89</td>
<td>22</td>
<td>39</td>
<td>1.43</td>
<td>135.24</td>
</tr>
<tr>
<td>9</td>
<td>517</td>
<td>234.29</td>
<td>505.97</td>
<td>271.68</td>
<td>45</td>
<td>49</td>
<td>1.58</td>
<td>2.57</td>
</tr>
<tr>
<td>10</td>
<td>486</td>
<td>241.65</td>
<td>276.28</td>
<td>34.63</td>
<td>50</td>
<td>49</td>
<td>1.02</td>
<td>-66.21</td>
</tr>
</tbody>
</table>

Table A9: Mean results for 40 m range with ground truth object visibility > 80%.
<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>442</td>
<td>109</td>
<td>135</td>
<td>26</td>
<td>25</td>
<td>58</td>
<td>1.08</td>
<td>-27.93</td>
</tr>
<tr>
<td>2</td>
<td>304</td>
<td>45</td>
<td>54</td>
<td>9</td>
<td>15</td>
<td>57</td>
<td>1.16</td>
<td>-87.77</td>
</tr>
<tr>
<td>3</td>
<td>280</td>
<td>66</td>
<td>101</td>
<td>35</td>
<td>24</td>
<td>48</td>
<td>1.63</td>
<td>-50.66</td>
</tr>
<tr>
<td>4</td>
<td>719</td>
<td>131</td>
<td>222</td>
<td>91</td>
<td>18</td>
<td>60</td>
<td>0.96</td>
<td>95.96</td>
</tr>
<tr>
<td>5</td>
<td>509</td>
<td>159</td>
<td>238</td>
<td>79</td>
<td>31</td>
<td>53</td>
<td>1.68</td>
<td>-75.38</td>
</tr>
<tr>
<td>6</td>
<td>211</td>
<td>49</td>
<td>76</td>
<td>27</td>
<td>23</td>
<td>49</td>
<td>1.48</td>
<td>-21.60</td>
</tr>
<tr>
<td>7</td>
<td>608</td>
<td>102</td>
<td>124</td>
<td>22</td>
<td>17</td>
<td>61</td>
<td>0.98</td>
<td>-61.35</td>
</tr>
<tr>
<td>8</td>
<td>479</td>
<td>96</td>
<td>117</td>
<td>21</td>
<td>20</td>
<td>43</td>
<td>1.25</td>
<td>187.99</td>
</tr>
<tr>
<td>9</td>
<td>535</td>
<td>144</td>
<td>259</td>
<td>115</td>
<td>27</td>
<td>58</td>
<td>1.59</td>
<td>39.25</td>
</tr>
<tr>
<td>10</td>
<td>545</td>
<td>180</td>
<td>191</td>
<td>11</td>
<td>33</td>
<td>52</td>
<td>1.20</td>
<td>-69.11</td>
</tr>
</tbody>
</table>

Table A10: Strict results for 65 m range with ground truth object visibility > 80%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>442</td>
<td>332</td>
<td>699</td>
<td>367</td>
<td>75</td>
<td>52</td>
<td>1.05</td>
<td>-34.48</td>
</tr>
<tr>
<td>2</td>
<td>304</td>
<td>299</td>
<td>438</td>
<td>139</td>
<td>98</td>
<td>40</td>
<td>1.49</td>
<td>-91.45</td>
</tr>
<tr>
<td>3</td>
<td>280</td>
<td>280</td>
<td>558</td>
<td>278</td>
<td>100</td>
<td>47</td>
<td>1.63</td>
<td>-82.22</td>
</tr>
<tr>
<td>4</td>
<td>719</td>
<td>627</td>
<td>1810</td>
<td>1183</td>
<td>87</td>
<td>44</td>
<td>1.19</td>
<td>87.72</td>
</tr>
<tr>
<td>5</td>
<td>509</td>
<td>475</td>
<td>1371</td>
<td>896</td>
<td>93</td>
<td>42</td>
<td>1.95</td>
<td>-63.10</td>
</tr>
<tr>
<td>6</td>
<td>211</td>
<td>211</td>
<td>500</td>
<td>289</td>
<td>100</td>
<td>42</td>
<td>1.64</td>
<td>-42.17</td>
</tr>
<tr>
<td>7</td>
<td>608</td>
<td>443</td>
<td>790</td>
<td>347</td>
<td>73</td>
<td>48</td>
<td>1.13</td>
<td>-46.18</td>
</tr>
<tr>
<td>8</td>
<td>479</td>
<td>333</td>
<td>484</td>
<td>151</td>
<td>70</td>
<td>39</td>
<td>1.38</td>
<td>104.17</td>
</tr>
<tr>
<td>9</td>
<td>535</td>
<td>492</td>
<td>1500</td>
<td>1008</td>
<td>92</td>
<td>48</td>
<td>1.51</td>
<td>-18.17</td>
</tr>
<tr>
<td>10</td>
<td>545</td>
<td>545</td>
<td>717</td>
<td>172</td>
<td>100</td>
<td>53</td>
<td>1.06</td>
<td>-67.11</td>
</tr>
</tbody>
</table>

Table A11: Combined results for 65 m range with ground truth object visibility > 80%.

<table>
<thead>
<tr>
<th>#</th>
<th>GT</th>
<th>MD</th>
<th>TD</th>
<th>FP</th>
<th>mA</th>
<th>mIoU</th>
<th>CE m</th>
<th>SE %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>442</td>
<td>134.06</td>
<td>198.65</td>
<td>64.58</td>
<td>30</td>
<td>57</td>
<td>1.06</td>
<td>-39.41</td>
</tr>
<tr>
<td>2</td>
<td>304</td>
<td>68.51</td>
<td>110.56</td>
<td>42.05</td>
<td>23</td>
<td>46</td>
<td>1.48</td>
<td>-87.82</td>
</tr>
<tr>
<td>3</td>
<td>280</td>
<td>85.45</td>
<td>149.15</td>
<td>63.70</td>
<td>31</td>
<td>43</td>
<td>1.78</td>
<td>-80.38</td>
</tr>
<tr>
<td>4</td>
<td>719</td>
<td>223.13</td>
<td>431.57</td>
<td>208.44</td>
<td>31</td>
<td>50</td>
<td>1.08</td>
<td>80.88</td>
</tr>
<tr>
<td>5</td>
<td>509</td>
<td>213.37</td>
<td>422.65</td>
<td>209.28</td>
<td>42</td>
<td>48</td>
<td>1.87</td>
<td>-64.70</td>
</tr>
<tr>
<td>6</td>
<td>211</td>
<td>65.07</td>
<td>132.23</td>
<td>67.16</td>
<td>31</td>
<td>45</td>
<td>1.63</td>
<td>5.31</td>
</tr>
<tr>
<td>7</td>
<td>608</td>
<td>126.32</td>
<td>186.04</td>
<td>59.72</td>
<td>21</td>
<td>54</td>
<td>1.07</td>
<td>-44.66</td>
</tr>
<tr>
<td>8</td>
<td>479</td>
<td>103.98</td>
<td>141.81</td>
<td>37.83</td>
<td>22</td>
<td>39</td>
<td>1.43</td>
<td>134.22</td>
</tr>
<tr>
<td>9</td>
<td>535</td>
<td>236.05</td>
<td>539.54</td>
<td>303.49</td>
<td>44</td>
<td>51</td>
<td>1.58</td>
<td>1.89</td>
</tr>
<tr>
<td>10</td>
<td>545</td>
<td>249.25</td>
<td>297.34</td>
<td>48.09</td>
<td>46</td>
<td>49</td>
<td>1.06</td>
<td>-63.67</td>
</tr>
</tbody>
</table>

Table A12: Mean results for 65 m range with ground truth object visibility > 80%.
Figure A1: mA values across all datasets. Continuous line represents the strict values, dashed line the combined values and the dashed-dotted line the mean values. Red color represents the 40 m and 40% visibility evaluation criteria, magenta the 65 m and 40% criteria, cyan the 40 m and 80% criteria and blue the 65 m and 80% criteria.
Figure A2: mIoU values across all datasets. Continuous line represents the strict values, dashed line the combined values and the dashed-dotted line the mean values. Red color represents the 40 m and 40% visibility evaluation criteria, magenta the 65 m and 40% criteria, cyan the 40 m and 80% criteria and blue the 65 m and 80% criteria.
Figure A3: SE values across all datasets. Continuous line represents the strict values, dashed line the combined values and the dashed-dotted line the mean values. Red color represents the 40 m and 40% visibility evaluation criteria, magenta the 65 m and 40% criteria, cyan the 40 m and 80% criteria and blue the 65 m and 80% criteria. The dotted black line represent the 0% line.