FREIE UNIVERSITÄT BERLIN

INSTITUTE OF COMPUTER SCIENCE

Master’s Thesis in Data Science

Multimodal object detection based on deep learning

Author:	Evgenia Slivko
Supervisor:	Jessica Giovagnola, Miguel Molina Fernandez, Infineon Technologies AG
Advisor:	Prof. Dr. Daniel Göhring, Prof. Dr. Tim Landgraf
Submission Date:	14.10.2022

Abstract

Deep learning is widely used in autonomous vehicles' environment perception systems that utilize data from a variety of sensors such as camera, LiDAR, RADAR, and Time-of-Flight. In order to build a reliable environment perception system for real-life applications, a vast amount of multi-modal labeled data for different light, weather, and climatic conditions is necessary for the training of deep learning environment perception models. Preparing such datasets is a non-trivial task that requires a huge amount of resources. Moreover, annotation of LIDAR data requires the designation of object bounding boxes in 3D space and a yaw angle for each object, while manual annotation of nighttime camera data is complicated since the objects are not clearly distinguishable due to the lack of contrast of the images that were obtained without sufficient light. Besides, with the development of technology new sensors appear that have distinctive features which determine the specific characteristics of the data. Therefore, the problem of transferring knowledge between sensors of the same and different modalities arises.

This master thesis is prepared with the German semiconductor and sensor manufacturer Infineon Technologies AG, which conducts research in AI based on sensor data. The master thesis addresses the problem of the lack of labeled data for autonomous vehicles' environment perception deep learning-based models training and the problem of transferring knowledge between sensors on the example of a camera and LiDAR data.

While working on the thesis, a custom dataset for multimodal environment perception was collected with the use of the Infineon multi-sensor setup, which included a camera, LiDAR, and other sensors. A synchronization for Camera and LiDAR data was performed by extracting 2D depth maps from 3D LiDAR point cloud and by calibration of the sensors using a planar checkerboard pattern. Using the transfer learning approach, the YOLOv5 object detection model was trained on Infineon camera image data. The weights were initialized from the object detection model that was pretrained on the MS COCO dataset. The technique to extrapolate labels from the camera images to LiDAR 2D depth maps was determined and implemented. The resulting labels were used for training an independent object detection model for Infineon 2D depth map data. Using the late fusion approach a sensor fusion algorithm was implemented to provide a unified perception of the environment for the autonomous vehicle. This approach allows to label multimodal data automatically and, therefore, significantly decreases the time and resources for dataset annotation.

Contents

1 Introduction

Autonomous vehicles is a rapidly evolving field that could revolutionize the future of mobility and transportation. Autonomous vehicles provide the transportation capabilities of conventional vehicles but are largely capable of perceiving the environment and self-navigating with minimal or no human intervention [1].

Nowadays, autonomous vehicles are represented mostly by self-driving cars and autonomous drones. Self-driving cars have great potential for both improving road safety and reducing the chance of human error while driving, which is the leading cause of traffic accidents and fatalities on the road. In addition, autonomous vehicles can provide reliable mobility for the elderly, young people, and users with disabilities, improve transport interconnectivity, decrease traffic congestion in the cities and significantly reduce the energy consumption for transportation by using the least energy-consuming driving style.

Autonomous drones have limitless potential for application in various industries including infrastructure inspection, surveying, capturing of aerial data, delivery, emergency rescue, agriculture, traffic control, and many others.

From a functional perspective, autonomous vehicles are composed of functional blocks, which are defined based on the flow of information and the processing stages [2]:

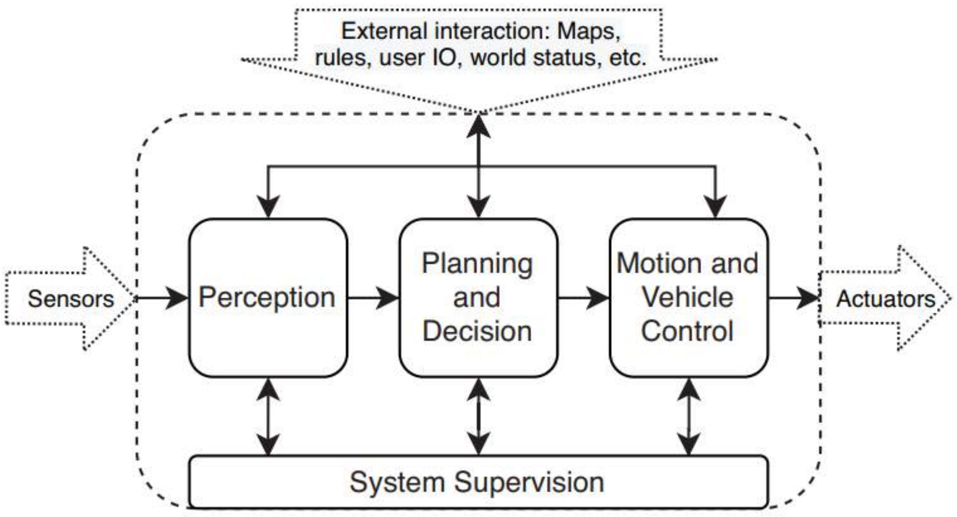

External interaction: Maps, rules, user IO, world status, etc.
Sensors
Perception
Planning and Decision
Motion and Vehicle Control
Actuators
System Supervision

Figure 1.1: Functional architecture of the autonomous driving system based on [2]

- Perception:
  Perception includes data collection from sensors and other sources in order to generate

a representation of the vehicle status and a world model, localization and mapping, and object detection.

- Planning and decision:
  Planning and decision include plan navigation based on the goal specified by the user, including long-term and short-term navigation plans, processing of information from external sources (traffic rules, maps updates, and speed limits) in order to ensure safety as well as Reactive behaviors such as collision-preventing features.

- Motion and vehicle control:
  Motion and vehicle control is related to how the trajectory generated in the previous step is performed on the platform.

- System supervision:
  System supervision is in charge of monitoring all the aspects of the vehicle, including hardware and software [2].

In order to ensure the primary functions described above, autonomous vehicles utilize various in-vehicle technologies, including hardware (sensors, processing units, communication technologies, internal networking interfaces, etc.) and software (operating systems, software for data collection, machine learning and deep learning algorithms, as well as real-time and safety-critical software, user interface software, etc.).

Overall performance and safety of the autonomous vehicle depend on the input data coming from a variety of sensors, that shape a reliable picture of the environment and the internal state of the vehicle. External state sensors such as monocular and stereo cameras, LiDAR, RADAR, ultrasonic sensors, etc. are often used for environment perception. The application of sensors of different modalities ensures a more reliable representation of the environment allowing to overcome the drawbacks of individual sensor types.

Current work is focused on the application of monocular camera and LiDAR data as well as sensor fusion of camera and LiDAR for object detection based on deep learning for autonomous vehicles and exploration of the possibilities of labeling of data from one modality based on data from another modality for the object detection model training.

The use of several sensors at the same time requires very precise synchronization and calibration among all the sensors and provides the opportunity to utilize joint representation of the data from multiple sensors for environment perception tasks, such as object detection and obstacle avoidance. Calibration between the camera and LiDAR is part of the current work and is described in sections section 2.3, section 3.1, section 4.1.

1.1 Goal

1.1.1 Thesis goal

Environmental perception algorithms are usually based on supervised learning techniques and thus require a vast amount of data for model training. In order to create a model that is well-performing in real life, a sufficient amount of high-quality labeled data that is recorded in various weather, climatic and illumination conditions and in different environments should be available.

ILSVRC ImageNet dataset [3] (more than 200 classes), the PASCAL VOC 2012 dataset [4] (20 classes) and the MS COCO dataset [5] (80 classes) became very popular for 2D object detection models training and performance evaluation. At the same time the datasets KITTI [6], Waymo [7], nuScenes [8] and others are used for the development and evaluation of 2D, 3D and multimodal deep learning models for environment perception tasks such as object detection, object tracking, instance segmentation, etc. Annotation, collection and retrieval of multimodal datasets is a key obstacle in the overall object detection pipeline, since acquiring and labeling such data is a time-consuming process that requires the availability of expensive sensors and other technologies as well as a highly professional team that will ensure data quality.

This thesis has two main goals: the first goal is to propose and demonstrate an approach to the creation of the annotated multimodal dataset using a monocular camera and LiDAR, which will be used for the development of the deep learning based models for object detection, meanwhile, the second goal is the development of the object detection tool for automatic data labeling from several modalities (Camera, LiDAR, etc.) in order to enable an autonomous vehicle to avoid obstacles such as pedestrians, cyclists, trees, buildings, etc.

The goals stated above have been achieved through the following steps:

- Collection of the multi-modal synchronous data from LiDAR and monocular camera,
- Development and implementation of the approach for autolabeling of multi-modal data from LiDAR and camera,
- Implementation of the sensor fusion algorithm to determine the size of resulting bounding boxes and object classes based on late fusion approach (See section 3.5).
- Analysis of the advantages of fusion of several modalities (camera and LiDAR) in comparison to uni-modal approaches.

1.1.2 Own contribution

Traditional data collection and annotation assume the annotation of data from each modality independently. In this work, the approach to simultaneous data labeling from several modalities via the transfer learning approach is demonstrated.

Multi-modal synchronous data has been acquired using an experimental multisensor setup (see section 2.4) that has been developed by researchers of Infineon Technologies AG. With help of the transfer learning approach, Infineon 2D image data were labeled using the YOLOv5 object detection model, pretrained on the MS COCO dataset [5]. Instead of MS COCO labels, a custom set of 5 labels which is suitable for application in autonomous vehicles was used. The set of labels includes the following object classes: pedestrian, tree, building, fence, and cyclist.

2D depth maps were extracted from 3D point cloud data obtained by LiDAR sensor and projected into an image after sensor calibration. This calibration is performed by extraction of the intrinsic parameters of the camera as well as extrinsic parameters of the camera and LiDAR. Therefore, it became possible to apply labels, obtained based on camera image data, for depth maps. Due to the distinction in fields of view of the camera and LiDAR, as well as a difference in technology and the time of receiving the frame, the location of the objects in the corresponding image and depth map is slightly different. Thus, filtering and bounding box adaptation (in terms of location and size) of the labels was conducted. Filtering included the elimination of labels outside of LiDAR field of view, analysis of 3D point cloud and ground plane fitting and extraction in order to enable more accurate clustering, clustering of the rest of the scene in 3D point cloud based on DBSCAN, and allocation of 2D projection of 3D clusters to 2D image labels. Despite a comprehensive approach to label filtering, partial manual adjustments of about 10% of filtered labels were made in order to ensure the high quality of the training data for the depth maps-based model.

As a result, a dataset for training and validation of the LiDAR depth maps-based object detection model was shaped. The resulting object detection model for LiDAR depth maps data was trained and demonstrated high performance. The last step of the pipeline is sensor fusion which was developed and implemented using the late fusion strategy.

The choice of architecture was made due to the interest of Infineon in the independent use of LiDAR in autonomous drones since LiDAR ensures a high level of data privacy that is necessary for the real-life application of drones. The creation of the dataset (depth maps and labels) for LiDAR-based object detection will enable further research in environment perception using LiDAR.

1.2 Overall approach and thesis implementation pipeline

The thesis implementation pipeline includes four parts:

- Data collection:
  Collection of multimodal data with the use of Infineon multisensor setup (See section 2.4).

- Data preprocessing:
  Data preprocessing step includes LiDAR-Camera sensor calibration (See section 3.1, section 4.1) as well as depth maps extraction (See section 3.4).

- Images based object detection model development:
  An object detection model for camera image data is implemented via a transfer learning approach. YOLOv5 [9] model pretrained on MS COCO dataset was used (See section 3.2, section 4.2).

- LiDAR depth maps based object detection model development:
  In order to prepare training data for depth maps based object detection model training, labels obtained by images based model were used and adjusted at the labels filtering

step (See section 3.3, section 4.3). YOLOv5 [9] architecture is used for depth maps based object detector (See section 3.4, section 4.4).

- Sensor fusion based on late fusion approach (See section 3.5, section 4.5).

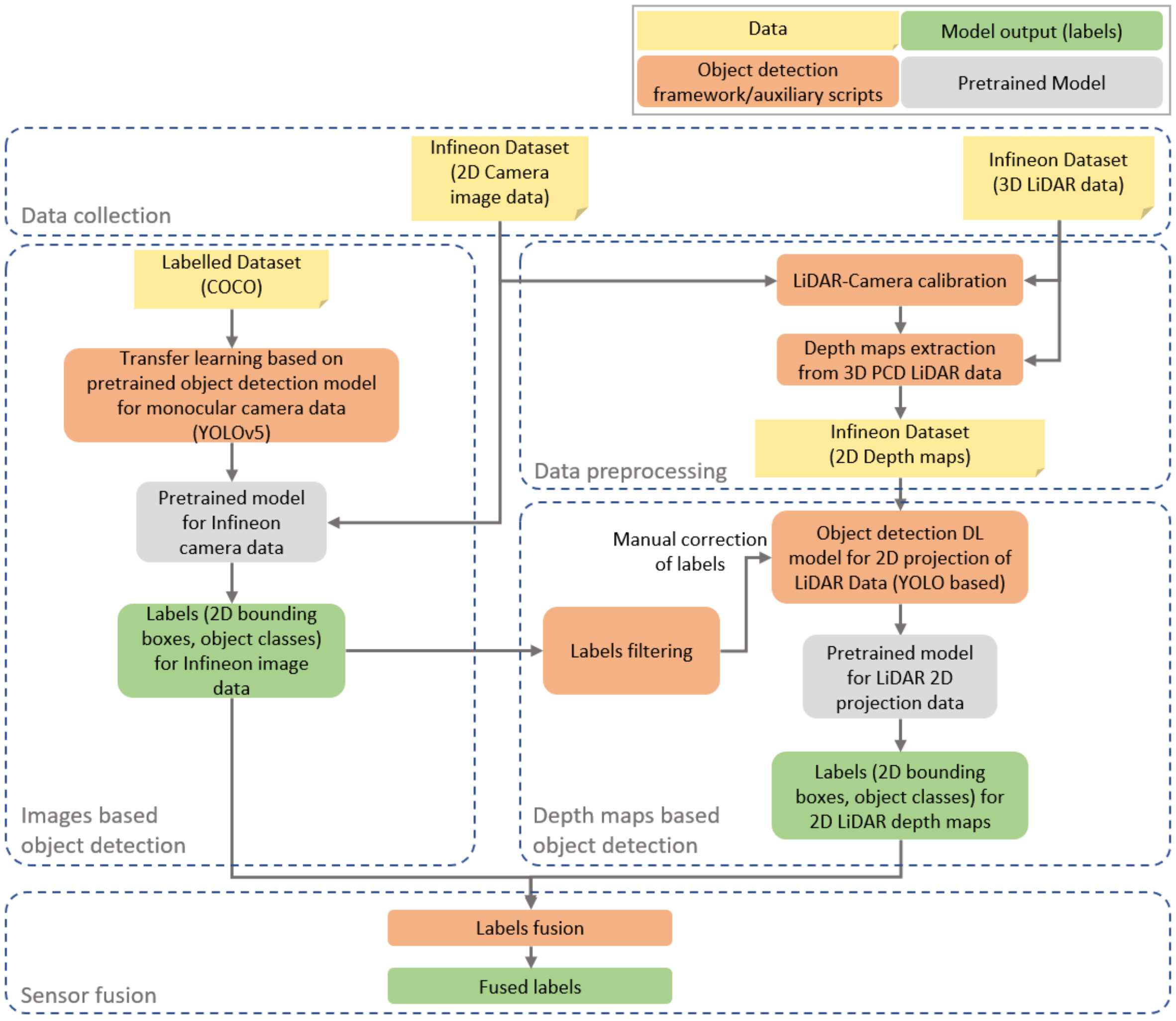

Data
Model output (labels)
Object detection framework/auxiliary scripts
Pretrained Model
Data collection
Infineon Dataset (2D Camera image data)
Infineon Dataset (3D LiDAR data)
Labelled Dataset (COCO)
Transfer learning based on pretrained object detection model for monocular camera data (YOLOv5)
Pretrained model for Infineon camera data
Labels (2D bounding boxes, object classes) for Infineon image data
Images based object detection
LiDAR-Camera calibration
Depth maps extraction from 3D PCD LiDAR data
Infineon Dataset (2D Depth maps)
Data preprocessing
Manual correction of labels
Object detection DL model for 2D projection of LiDAR Data (YOLO based)
Labels filtering
Pretrained model for LiDAR 2D projection data
Labels (2D bounding boxes, object classes) for 2D LiDAR depth maps
Depth maps based object detection
Labels fusion
Fused labels
Sensor fusion

Figure 1.2: Thesis implementation pipeline

As a result of the work conducted, two independent unimodal object detection models were trained based on a custom dataset, which has been annotated using the framework described above. Sensor fusion which is carried out at the last step shapes a single perception of the environment based on multimodal data, including classes and bounding boxes of objects. The sensor fusion approach is shown in the Figure 1.3 below: data of each modality are fully processed independently. In the last step, sensor fusion is applied to the labels obtained for each modality.

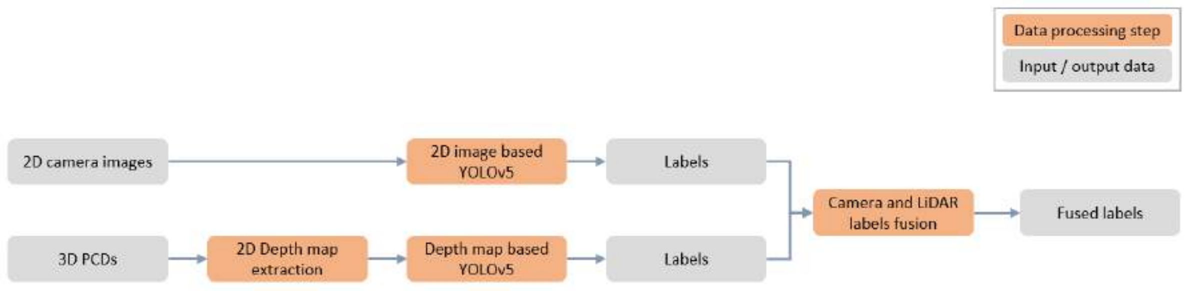


Figure 1.3: Object detection pipeline with late fusion approach

2 Preliminaries

2.1 Object detection in autonomous vehicles

In perception, autonomous vehicles (AVs) rely on a variety of sensors such as cameras, LiDARs, and radars to detect, understand and interpret the environment, including static and moving objects in order to avoid obstacles and ensure safe moving.

Object detection is one of the core tasks in the environment perception of AV and is a foundation for other AV functions such as path planning and motion control. Object detection is a computer vision task of detecting instances of objects of certain classes (such as humans, trees, buildings, cars, etc.) and localizing objects in digital images or videos.

Over the years, a wide variety of object detection algorithms have been developed. Earlier works use the approach of geometric features extraction. For example, In 2001 Viola and Jones developed an object detector for the detection of a non-occluded, upright face in frontal view [10]. Viola-Jones object detector utilized "integral image" representation that enabled fast processing of the image, Ada-boost algorithm on Haar feature classifiers in order to select a small number of critical visual features from a large set of potential features and a method for combining classifiers in "cascade" that allows quick extraction of background.

In 2004, Dalal et. al. suggested using a Histogram of Oriented Gradient (HOG) descriptors in order to extract a feature set to discriminate a human in the image [11] for pedestrian detection. SVM (Support Vector Machine) classifier was used for the final classification decision (person or non-person). HOG detectors were later extended to Deformable Part-based Models (DPMs), which were capable to detect multiple objects in the image.

Deep learning is one of the key techniques in the field of artificial intelligence research nowadays. Over the last decade, many deep learning-based solutions have been presented in the field of autonomous vehicles and have achieved outstanding results. Deep learning is part of machine learning methods that are based on artificial neural networks and imitates functionality and the process of learning of the human brain. Interest in deep learning significantly increased in 2012 with the introduction of AlexNet [12] - convolutional neural network architecture built by A.Krizhevsky, I.Sutskever, G.Hinton and trained on GPU. AlexNet achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry at ImageNet Large Scale Visual Recognition Challenge 2012.

Deep learning utilizes a variety of architectures such as convolutional neural networks, transformers, recurrent neural networks, etc., and is widely applied nowadays in various domains, such as image processing (object detection, semantic segmentation, object tracking, etc.), speech recognition, natural language processing, time series predictive modeling, autonomous vehicles, etc. The recent rapid development of deep learning became possible with advances in GPU technology and the availability of big datasets that are necessary to train deep learning models.

2.1.1 2D camera image based object detection

There are two most famous and widely applied types of image-based object detector architectures nowadays:

- two stages Region proposal Networks (RPN),
- single-stage detectors.

Two-stage object detectors perform a proposal of the region that might contain objects in the first stage and make a prediction of the object class and location in the second stage. In the first stage, the object detection model suggests several Regions of Interest (ROIs) of the image which are likely to contain objects of interest. In the second stage, the ROIs that are more likely to contain objects of interest are selected while other ROIs are discarded. Objects within selected ROIs are classified. The main drawback of two-stage object detection is inference speed, nevertheless, two-stage object detectors show high accuracy of predictions.

Single-stage object detectors utilize a single feed-forward neural network that creates bounding boxes and predicts objects in the same stage. Single-stage object detectors are faster than two-stage detectors and much more suitable for application in autonomous vehicles. At the same time, they are typically less accurate than two-stage object detectors.

Each of these types is described in the following sections (See subsubsection 2.1.1, subsubsection 2.1.1, subsubsection 2.1.1).

Region proposal detectors

Popular two-stage object detectors include RCNN, Fast R-CNN, and Faster R-CNN.

R-CNN

R-CNN architecture was suggested by Girshik et al. in 2014 [13]. R-CNN generates around 2000 category-independent region proposals for the input image via a selective search algorithm. Affine image warping is used to compute a fixed-size CNN input of 227x227x3 from each region proposal regardless of the input shape. A 4096-dimensional feature vector is extracted from each region proposal with the use of the Convolutional Neural Networks (CNN). At the next step, the model classifies each region with a set of category-specific linear SVMs [13].

Non-maximum suppression (NMS) step is performed at the end of the object detection pipeline. Non-maximum suppression is an algorithm that is used in both two-stages and single-stage object detectors in order to eliminate duplicate predictions for one object. NMS sorts all detection boxes based on their confidence scores, recursively selects detection with maximum score M, identifies detections that have an overlap with selected detection that is greater than a predefined threshold, and suppresses such detections.

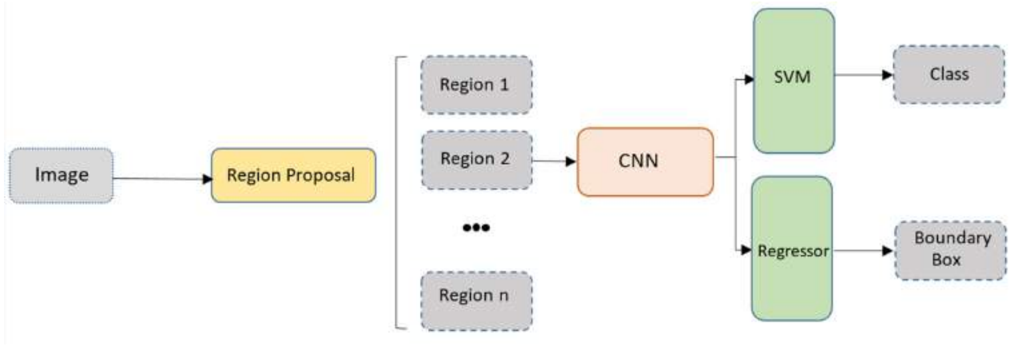


Figure 2.1: R-CNN architecture based on [14]

Fast R-CNN

In 2015 R.Girshick proposed an improved version of R-CNN - Fast R-CNN [15] that solved some of the problems of the R-CNN model and demonstrated increased inference speed and accuracy in comparison to R-CNN. This was achieved by computing the convolutional feature map and ROI proposals for the entire input image. For each object proposal, a ROI pooling layer extracted a fixed-length feature vector from the feature map. Then each feature vector is fed into a sequence of fully connected layers that have two outputs that produce softmax probability for possible object classes and a set of 4 values that encodes the refined bounding box location of the object.

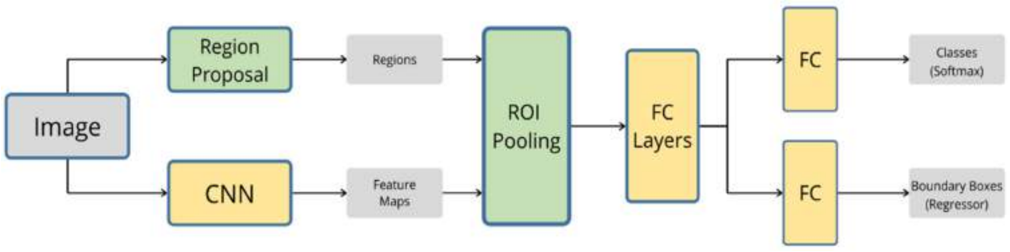


Figure 2.2: Fast RCNN architecture based on [14]

Faster R-CNN

In order to solve the computation bottleneck of R-CNN and Fast R-CNN at the region proposal step, authors of Faster R-CNN [16] employed a Region Proposal Network (RPN) that shared convolutional layers with the object detection network. RPN is a fully convolutional network and is designed to efficiently predict region proposals with a wide range of scales and aspect ratios. On the top of RPN, there are several additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid.

Faster R-CNN paper introduces the idea of anchors - bounding boxes that are placed throughout the image with different sizes and ratios and that are referenced when predicting object locations. As a result of the application of anchors, many-to-one predictions could occur for one object. Therefore, NMS post-processing step is necessary to remove duplicates.

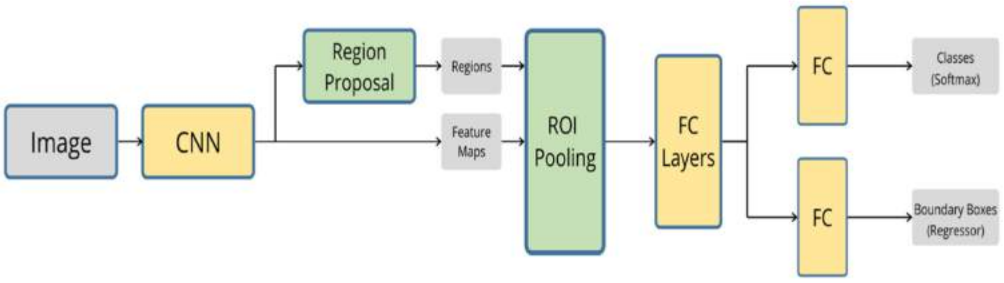


Figure 2.3: Faster RCNN architecture based on [14]

Single stage detectors

Single-stage object detectors are faster in comparison to two-stage detectors since they perform only a single pass in order to predict objects in the input image. Nevertheless, earlier versions of single-stage object detectors were less accurate in comparison to two-stages object detectors. Commonly used one-stage object detectors are SSD, YOLO, EfficientDet, RetinaNet, FCOS, OneNet, DETR, etc.

Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector (SSD) architecture was proposed in 2015 by Liu et al [17]. SSD encapsulates all computation of the object detection pipeline in a single network. The model architecture consists of a backbone network and detection head. The backbone model usually is a pre-trained image classification network with truncated classification layers that is used as a feature extractor. SSD detects objects at multiple scales by applying predictors to multiple feature maps from the later stages of a network. It distinguishes SSD from YOLO which operates on a single-scale feature map. SSD predicts object categories and offsets in bounding box locations for a fixed set of default bounding boxes and uses separate predictors (filters) for different aspect ratio detections. The model loss is calculated as a weighted sum between localization loss and confidence loss. At the last step of the object detection pipeline, NMS is performed in order to produce final predictions. The architecture of the SSD is shown in the Figure 2.4 below.

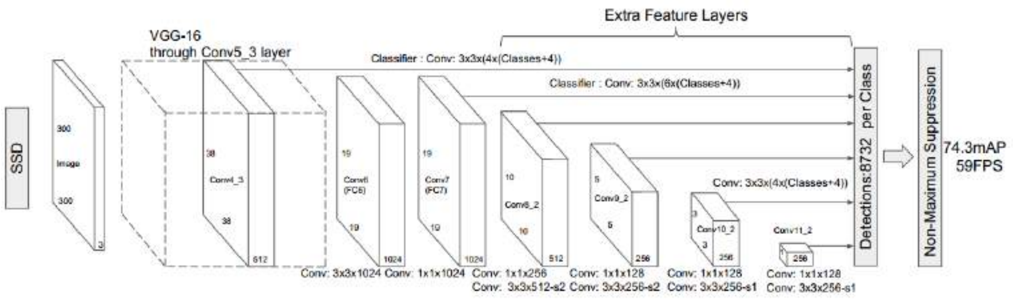


Figure 2.4: SSD architecture based on [17]

RetinaNet

While in two-stage object detectors classifier is applied to a limited sparse set of candidate object locations, in single-stage object detectors, on the contrary, dense regular samples of possible object locations are given to the classifier and thus most of the samples contain background. As a result, single-stage object detectors had a common problem that significantly decreased accuracy - foreground-background class imbalance, when foreground classes are under-represented and background class is extremely over-represented.

Due to foreground-background class imbalance training becomes inefficient as most locations are so called easy negatives and do not contribute any useful information to the model to learn, the easy negatives can overwhelm training and lead to degenerate models that always predict background class.

Lin et al. suggested a way to mitigate this problem in [18] and developed RetinaNet object detection architecture that utilizes a focal loss function, that contains commonly used cross-entropy loss multiplied by a modulating factor γ. Focal loss is defined as following:

$$FL(p_t) = -(1 - p_t)^{\gamma} \log p_t \tag{2.1}$$

Thus, if the confidence score p_t is low, the modulating factor $-(1 - p_t)^{\gamma}$ is approximately equal to 1 and the loss is unaffected. If p_t is close to 1, the factor $-(1 - p_t)^{\gamma}$ goes to 0 and the loss for well-classified examples is down-weighted. Authors analysed $\gamma \in [0, 5]$ and found that $\gamma = 2$ works best in their experiments [18]. Modulating factor reduces loss from "easy" examples with high confidence score p_t and helps the model to focus on learning more difficult ones.

RetinaNet object detection consists of backbone network ResNet-101, FPN (Feature Pyramid Network) as a neck and detection head that is represented by two task-specific subnetworks for object classification and bounding box regression. The backbone is responsible for computing a convolutional feature map over an entire input image while augmenting a standard convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a multi-scale feature pyramid from a single resolution input image and improves predictions for small-size objects [18]. Architecture of the RetinaNet is demonstrated in the Figure 2.5 below.

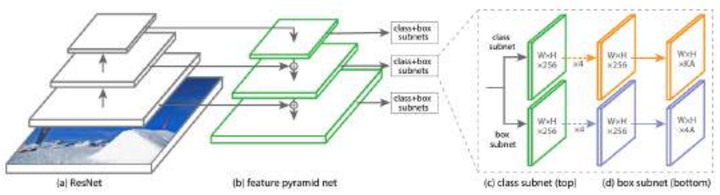
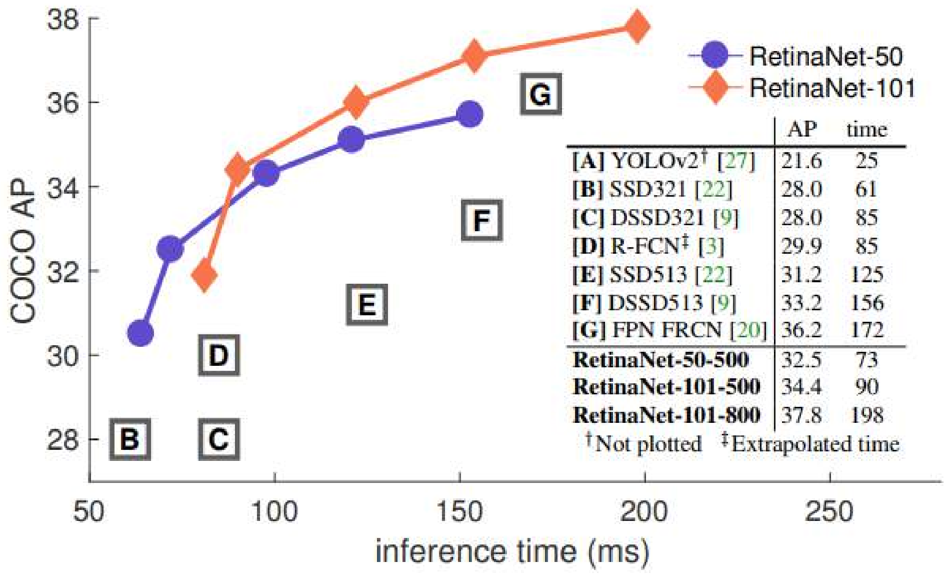


Figure 2.5: RetinaNet architecture based on [18]

Authors of RetinaNet architecture have demonstrated that RetinaNet outperformed other two-stage and single-stage object detectors that existed at the time in speed vs accuracy and reached average precision of AP = 37.8 which is higher than the closest result of FPN FRCN based on MS COCO test-dev dataset (See [18]).



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Figure 2.6: Speed (ms) versus accuracy (AP) on COCO test-dev based on [18]

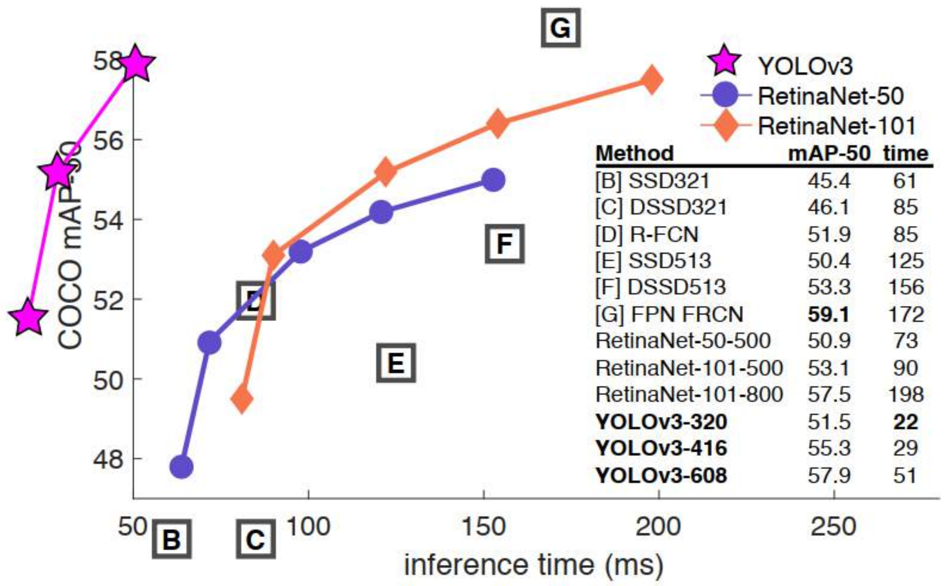
YOLO for 2D image based object detection

The first version of the single-stage object detector YOLO (You only look once) YOLOv1 was proposed by Redmon et al in 2016 [19]. YOLOv1 predicts the class and location for each object in the image in one pass and demonstrates high generalization ability and inference speed in comparison to current two-stages object detectors [15]. In YOLOv1 entire image is divided into an SxS grid (7 × 7 default). For each grid cell, B bounding boxes and corresponding confidence scores are predicted. Regardless of the number of bounding boxes per cell B, only one set of C conditional class probabilities is predicted for each grid cell, where C is the number of possible classes.

The number of bounding boxes B, as well as the ability to predict only one class of objects for each grid cell, is a strong limitation of the model. In addition, predictions are performed on the feature map that is obtained by several convolutional layers and multiple downsampling layers. Thus, the model extracts coarse features that are used for bounding box prediction. Such an architecture limits model performance for the small objects prediction.

In 2017 YOLO9000 paper (YOLOv2) was published [20]. YOLOv2 is a modified version of YOLOv1 that had improved overall performance via several new approaches and techniques. YOLOv2 is capable to predict 9000 classes due to training on both classification and object detection datasets. In YOLOv2 Darknet-19 is used as a backbone network. All fully connected layers are removed and anchor boxes are used to predict bounding boxes. K-means clustering is used in order to identify anchor box dimensions. The authors showed that k=5 gives a good trade-off between recall and complexity of the model. In order to increase the resolution of the resulting feature map, one of the pooling layers is removed. Authors used batch normalization on all convolutional layers in YOLOv2 and thus achieved faster convergence during training, regularization effect, and improvement of mAP by more than 2%. Furthermore, the authors suggested multi-scale training by changing of input size every 10 batches and expanding the 13x13 feature map, which is used for predictions, with features from an earlier layer at 26 × 26 resolution via concatenation through the pass-through layer.

In 2018 YOLOv3 paper [21] was published with further improvements of the You Only Look Once architecture. YOLOv3 uses a 53-layered backbone network Darknet-53 for feature extraction. Darknet-53 mainly consists of convolutional layers with kernel size 3x3 and 1x1 and has skip connections like the residual network ResNet [22]. As in YOLOv2, YOLOv3 predicted 4 coordinates for each bounding box: bounding box center along the x and y axis and width and height of the bounding box. The Sum of squared error is used as a loss function during training. In YOLOv3 softmax function for the class prediction is replaced with independent logistic classifiers for each class and binary cross-entropy loss for classification loss calculation during training. It allows to use the classes that are not mutually exclusive and thus to put several labels for each object. YOLOv3 makes predictions at 3 different scales via an approach that is similar to the feature pyramid network. In YOLOv3 predictions are performed on the output feature map that is concatenated with 2 previous layers after upsampling. Improvements that are described above enhanced the prediction of the overlapping bounding boxes and smaller objects. A comparison with other architectures that existed at the time of publication is shown in the graph below (See Figure 2.7). YOLOv3 demonstrated high accuracy together with a high speed of detection.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 2.7: Performance comparison (speed vs accuracy) of YOlOv3 with other SOTA object detection architectures based on [21]

In 2020, YOLOv4 [23] was introduced. Authors proposed two groups of techniques - “bag of freebies” and “bag of specials”. “Bag of freebies” – improvements in the training process that enhanced the accuracy of the model, and "bag of specials" - improvements that aimed mostly at increasing the inference speed:

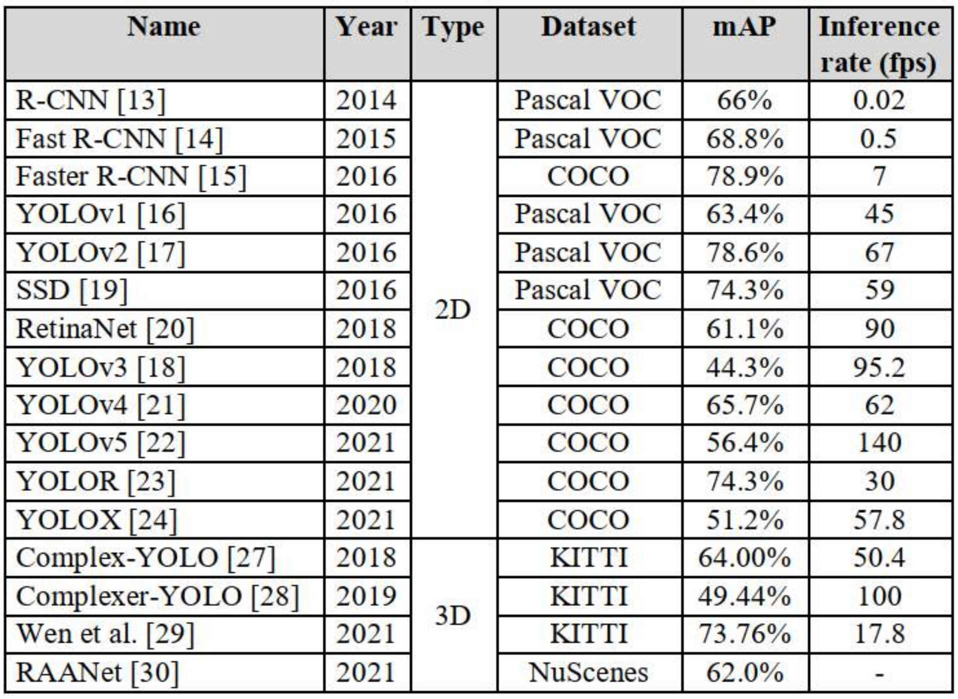
- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing;
- Bag of Specials (BoS) for backbone: Mish activation, Cross-stage partial connections (CSP), Multi-input weighted residual connections (MiWRC);
- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for single ground truth, Cosine annealing scheduler, Optimal hyperparameters, Random training shapes;
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS.

YOLOv4's architecture consists of CSPDarknet-53 [24] as a backbone, modified SPP [25] (Spatial Pyramid Pooling), and PAN [26] (Path Aggregation Network) as a neck and YOLOv3 detection head [21].

Backbone network CSPDarknet-53 is inspired by ideas of Darknet-53 that was used as a backbone in YOLOv3 [21], and Cross Stage Partial DenseNet (CSPNet) [24]. Object detection neck collects feature maps from different levels of aggregation in the backbone. The neck models include top-down paths and bottom-up paths in order to provide spatial information from the bottom-up path and semantic information from the top-down path as input for the detection head. YOLOv4 includes SSP in order to increase receptive field and PAN - as a method for parameter aggregation from different backbone levels for different detector levels [23].

In 2020 one more version of YOLO - YOLOv5 was proposed by Ultralytics [9]. While previous versions of YOLO were developed based on Darknet (written in C), Ultralytics prepared implementations in Pytorch, a widely used open-source machine learning framework developed by Meta AI. Research and development of YOLOv4 and YOLOv5 by two different groups of researchers were carried out at the same time. Both groups of researchers included state-of-the-art techniques in object detection. As a result architectures of YOLOv4 and YOLOv5 are very similar. The publication of the YOLOv5 paper which was promised by the authors has never been published.

As well as YOLOv4, YOLOv5 has a CSP backbone and PAN neck. One of the improvements of YOLOv5 in comparison to YOLOv4 is an integration of the anchors' bounding boxes learning into the model pipeline. Therefore, no preliminary analysis of bounding boxes in the dataset is needed. Figure 2.8 summarizes information about the speed and mAP (mean average precision) of several widely used object detection models. This table demonstrates, that YOLOv5 outperforms other models in inference speed while showing reasonably high performance. Since both the accuracy of the model and its speed are important for the application in autonomous vehicles, YOLOv5 has been chosen for further experiments in the current work.



Name	Year	Type	Dataset	mAP	Inference rate (fps)
R-CNN [13]	2014	2D	Pascal VOC	66%	0.02
Fast R-CNN [14]	2015		Pascal VOC	68.8%	0.5
Faster R-CNN [15]	2016		COCO	78.9%	7
YOLOv1 [16]	2016		Pascal VOC	63.4%	45
YOLOv2 [17]	2016		Pascal VOC	78.6%	67
SSD [19]	2016		Pascal VOC	74.3%	59
RetinaNet [20]	2018		COCO	61.1%	90
YOLOv3 [18]	2018		COCO	44.3%	95.2
YOLOv4 [21]	2020		COCO	65.7%	62
YOLOv5 [22]	2021		COCO	56.4%	140
YOLOR [23]	2021		COCO	74.3%	30
YOLOX [24]	2021		COCO	51.2%	57.8
Complex-YOLO [27]	2018	3D	KITTI	64.00%	50.4
Complexer-YOLO [28]	2019		KITTI	49.44%	100
Wen et al. [29]	2021		KITTI	73.76%	17.8
RAANet [30]	2021		NuScenes	62.0%	-

Figure 2.8: Performance comparison (speed vs accuracy) of 2D camera image based object detection architectures based on [27]

2.1.2 3D Point cloud based object detection

3D object detection task is formulated as following: Given the point cloud of a scene formed by the LiDAR sensor and represented in the LiDAR coordinate frame, predict oriented 3d bounding boxes of the target objects (represented in the LiDAR coordinate frame) and corresponding class of the object. An oriented 3d bounding box is represented by coordinates of the center of 3d box x_c, y_c, z_c, its length, width, height w, h, l and augmented with its heading angle θ (yaw rotation around Z-axis) with respect to the body coordinate frame of the 3d box. Angle $\theta = 0$ of a box refers to the case when its length dimension (longer side on the X-Y plane) is parallel to the X-axis of the LiDAR coordinate frame.

There are three main directions for the development of object detectors based on LiDAR 3D point cloud data:

- Discretization (Voxelization) based methods,
- Architectures where 3D point cloud is directly processed (point based),
- Architectures where the 3D point is transformed to the 2D pseudo-image in order to apply 2D object detection methods, and
- Multi-modal fusion-based methods that combine data from several sensors, for example, camera and LiDAR data (This group of methods is described in the section section 2.2).

Historically object detection was developed for 2D images and a lot of methods were created for image data. Remarkable results in image-based object detection were achieved with Convolution Neural Networks (CNN), that use dense, ordered, regular representation as an input. The point cloud data, on the contrary, is unordered, sparse, and has varying density. These distinctive features of point clouds do not allow direct use of such methods as CNN.

Voxel based object detection architectures

Discretization-based methods transform unordered LiDAR data in order to create an ordered, grid-like representation (pseudo-image) that can be used as input to CNN.

The main idea of voxelization-based methods is to divide 3D space into 3D or 2D cells – voxels and allocate points to the voxels based on the coordinates. The features that describe the voxels are derived from each voxel and are represented as 4D tensors (in the case of 3D voxelization) or 3D tensors (in the case of 2D voxelization along the plane). Such an approach provides the opportunity for the application of convolutional neural networks for further processing and object detection. Widely used examples of such approach - VoxelNet and PointPillars. One of the major differences between VoxelNet and PointPillars is a voxelization method: In VoxelNet authors divide the space along all 3 axes X, Y and Z, while in PointPillars the space is divided only along X and Y-axes which leads to a forming of the columns alone Z-axis, so-called "pillars".

VoxelNet architecture was proposed in 2017 in [28] and become the first end-to-end trainable architecture for 3D object detection. In VoxelNet a point cloud is partitioned into voxels of the equal size and the features within each voxel are encoded with feature encoding layers based on PointNet. As a result, a 4D tensor is produced. 3D Convolution is applied to transform 4D tensor into a 3D tensor and prepared data for processing by the region proposal network that is used for object detection. 3D convolution layers aggregate local voxel features and transform the point cloud into a high-dimensional volumetric representation. These layers add more context to the shape description and prepare data for feeding into Region Proposal Network. A regional proposal network (RPN) is a fully convolutional network that simultaneously predicts object bounds and object class probabilities scores at each position. RPN in VoxelNet contains two sub-networks: one top-down network that produces features at increasingly small spatial resolution and a second network that performs upsampling and concatenation of the top-down features.

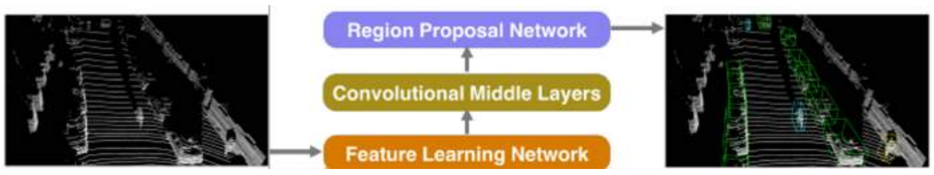


Figure 2.9: VoxelNet architecture

PointPillars paper [29] was published one year later. PointPillars architecture consists of 3 parts:

- a feature encoder network that converts a raw point cloud data to a pseudo-image (3D Tensor) suitable for processing by convolutional neural networks,
- a 2D convolutional backbone network to process the pseudo-image into a feature map, and
- a Single Shot Detection (SSD) head that regresses 3D bounding boxes and outputs class of the object.

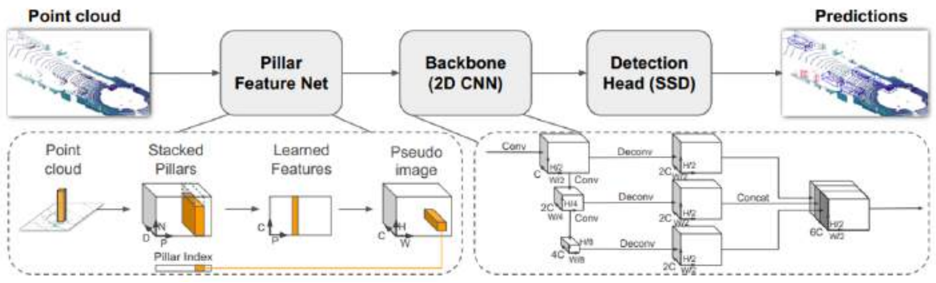


Figure 2.10: PointPillars architecture

In PointPillars 3D space is divided into voxels in the X-Y plane resulting in vertical columns (pillars). Each point in the pillar is encoded by a 9-Dimensional vector that is used for scene representation as a 3D tensor. A linear layer followed by BatchNorm and ReLU operators is applied on the 3D tensor with an output number of features of the fixed size. After the max-pooling operation over the channels of the tensor, the pillar feature vectors are brought back to the original pillar locations to create a pseudo-image. The structure of the backbone network in PoinPillars is similar to VoxelNet, and the main difference is Detection Head. While in VoxelNet two-stage regional proposal network is used, PointPillars uses Single Shot Detector (SSD).

In a single-stage architecture, a dense set of anchor boxes is regressed and classified in a single stage into a set of predictions providing a fast and simple architecture. With focal loss, it outperforms RPN in both speed and accuracy.

Point based object detection architectures

Despite the voxelization methods that are described in the subsubsection 2.1.2 above are efficient due to the opportunity to apply 3D or 2D convolutions on voxel representation, voxelization results in the loss of detailed information of the shape of the objects. In addition, voxelization takes a lot of computational resources. Point-based object detection methods attempt to reduce information loss caused by either projection or voxelization. Point-based object detection operates directly on raw point cloud and does not have such a disadvantage. Nevertheless, processing of raw point cloud is a challenging task since point cloud is an unevenly distributed, unordered, and thus permutation invariant set of points. At the same time using the whole point cloud as input can increase run-time. One of the fundamental point-based architectures is PointNet [30]. PointNet is a unified architecture that operates directly on the point cloud and outputs either class labels for the entire input or per point segment/part labels for each point of the input [30]. The network performs point-wise transformations using Fully-Connected layers and uses the max-pooling layer as a symmetric function to aggregate information from all points in order to ensure that the model is invariant to input permutation.

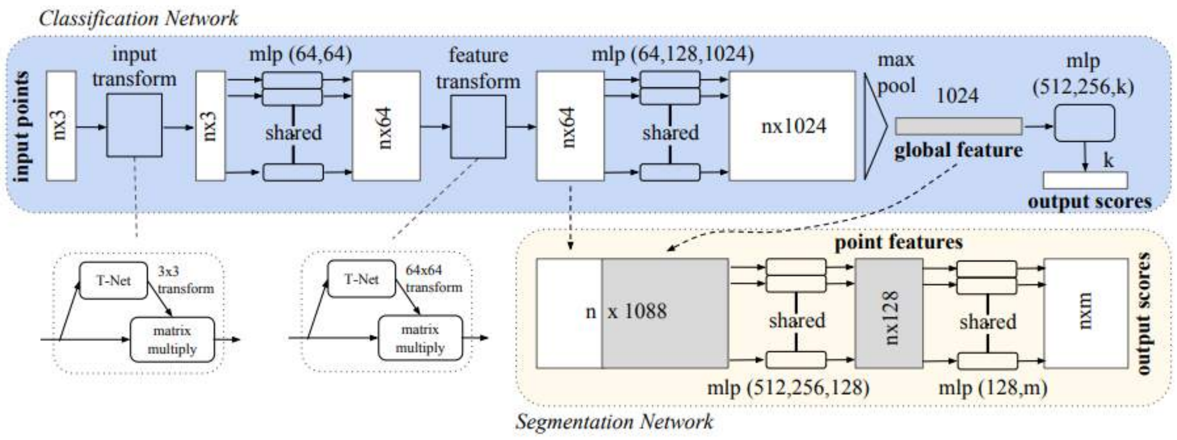


Figure 2.11: PointNet architecture based on [30]

PointNet-like architecture is used as a building block in many 3D object detection architectures. other examples of such architectures are PointRCNN [31], 3DSSD [32] and Point-GNN [33]. PointRCNN uses a block similar to PointNet to learn the semantic signals associated with foreground points from which 3D proposals are generated. In 3DSSD architecture, authors suggest a new sampling approach in order to make detection on less representative points feasible and reduce computational costs. The Point-GNN is single-stage detection architecture that has three main components: graph construction from a point cloud, a graph neural network for object detection, and bounding box merging and scoring [33].

2D depth maps based object detection architectures

A depth map is a robust representation that is highly invariant to lighting changes. 2D depth maps based object detection architectures transform the 3D point cloud into a 2D depth map representation first via plane, cylindrical, or spherical coordinate transformation. That allows to directly apply convolutional neural networks and, therefore, to use of methods that are developed for camera images.

Besides depth map representations are widely used in sensor fusion methods, including early and halfway fusion [34]. Sensor fusion methods are considered in the section 2.2. In comparison to the 3D point cloud, processing of 2D depth maps requires much less computational power and thus, is suitable for application on mobile devices and in real-time scenarios.

One of the examples of such methods is 2.5VoteNet [35] architecture that extracts features directly on depth maps and learns robust local features using relative depth convolution (RDConv) block. At the same time depth map is scaled to the size of the feature map and lifted into 3D space. The feature vectors from the feature map are then matched to the corresponding 3D point. After that, the detection and NMS are performed in 3D space in order to use the spatial information. 2.5VoteNet achieves a high inference speed of 69 FPS.

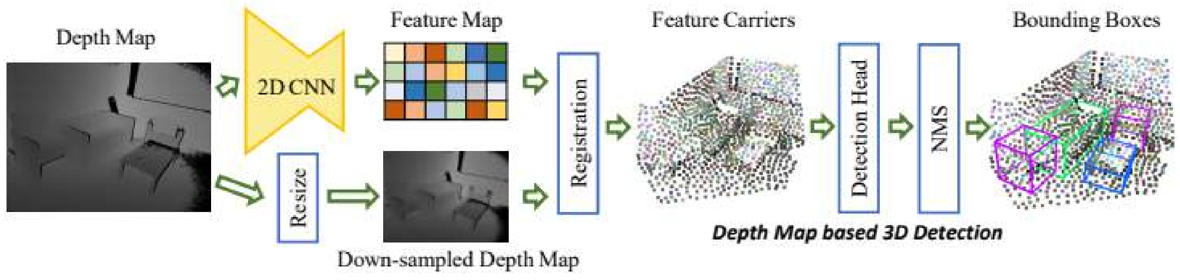


Figure 2.12: 2.5Votenet architecture based on [35]

2.2 Sensor fusion for object detection (Camera and LiDAR)

2.2.1 Sensor fusion definition

D.L.Hall et al. define data fusion in [36] in the following way:

> Data fusion techniques combine data from multiple sensors, and related information from associated databases, to achieve improved accuracy and more specific inferences than could be achieved by the use of a single sensor alone.

The authors of the paper characterize data fusion as a hierarchical transformation between observed energy or parameters (provided by multiple sources as input) and a decision or inference (produced by fusion estimation and/or inference processes) regarding the location, characteristics, and identity of an entity, and an interpretation of the observed entity in the context of a surrounding environment and relationships to other entities [36].

Bellot et al. in [37] define 4 categories of the gain obtained with the use of data fusion: gain in representation (abstraction level of data at the end of a fusion process), completeness (effectiveness of the fusion process to enhance the description of an environment), accuracy (reduction of noise and approximation on data after fusion) and certainty (an increase of the belief on data after the fusion process).

2.2.2 Commonly used modalities in sensor fusion

Data from several different modalities are commonly used for reliable environment perception in autonomous driving. Frequently used types of sensors include monocular and stereo cameras, LiDAR, RADAR, ultrasonic sensors, etc.

Monocular and stereo camera

The camera is the most used sensor for perceiving the environment. A camera is an optical instrument that captures a visual image. A camera lens takes the light rays bouncing around and uses glass to redirect them to a photosensitive surface, creating an image. Cameras could provide high-resolution 2D images of surroundings and are relatively inexpensive in comparison to other types of sensors. A monocular camera has one sensor and creates a sequence of images. A stereo camera has two sensors, that are spaced a small distance (baseline) apart from each other. Both sensors produce synchronized images. A stereo camera takes the two images from sensors and compares them. Since the distance between the sensors is known, the comparison gives depth information. The camera's performance strongly depends on light conditions, therefore camera data is often supplemented by data obtained from other types of sensors.

LiDAR

LiDAR is an acronym for "Light Detection and Ranging" or "Laser imaging, Detection, and Ranging". LiDAR is a technology for 3D scanning of the environment. Typically, LiDAR sensor emits pulsed light waves that are outside of the visible spectrum into the surrounding environment. These pulses bounce off surrounding objects and return to the sensor. The sensor uses the time it took for each pulse to return to calculate the distance it traveled. Unlike cameras, LiDAR sensor functions independently of the ambient lighting and provide robust results without any loss of performance due to disturbances such as shadows, sunlight, or headlight glare during both day and night time. LiDAR can target a wide range of materials, including non-metallic objects, rocks, rain, chemical compounds, aerosols, clouds, and even single molecules. LiDAR sensors have high resolution, operate at ranges up to 300 meters and demonstrate very precise results with the error up to 2cm. LiDAR technology provides a very accurate 3D reconstruction of a scene due to the high number of measurement points for each scene.

LiDAR sensor returns a point cloud, an unordered set of points that contains coordinates of each point in the 3D space in Cartesian (x, y, z) or spherical coordinate system (r, theta, phi), and reflectance information. Point cloud representation preserves the original geometric information in 3D space without any discretization. Therefore, it is the preferred representation for many scene understanding related applications such as autonomous driving and robotics. 3D point cloud has specific features that distinguish this data type from 2D image representation:

- Unorderdness:
  The point cloud of a scene is the set of points (usually represented by coordinates) obtained from the objects in the scene and are usually stored as a list in a file. The order in which the points are stored does not change the scene representation.

- Irregularity:
  The points in the point cloud are not evenly sampled across the scene. The resulting

point cloud could have areas that are represented by a high number of dense points as well as areas that are represented by the small number of sparse points.

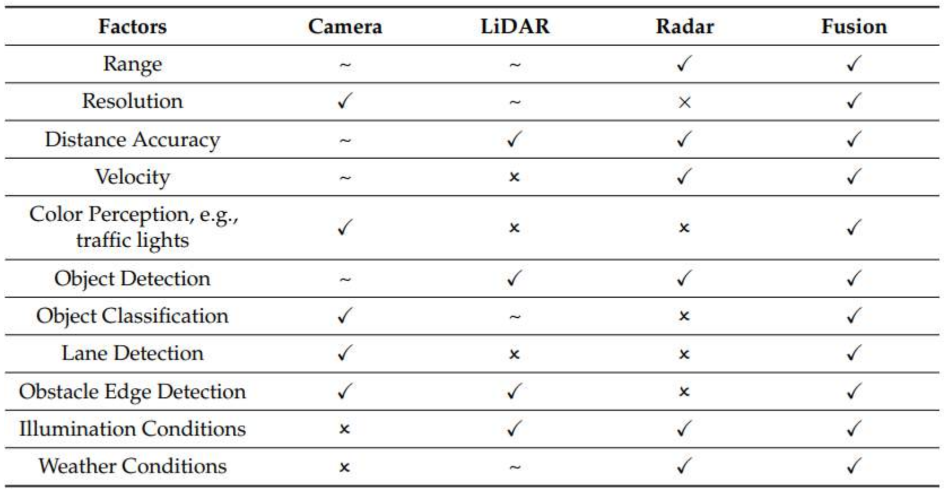
- Unstructureness:
  Points in the point cloud are not located on the regular grid. Each point is scanned independently and its distance to neighboring points is not fixed. In contrast, pixels in images are represented on a 2 dimension grid, and spacing between two adjacent pixels is always fixed.

RADAR

RADAR (Radio Detection and Ranging) works on the principle of radiating electromagnetic waves and receiving the scattered waves or reflections of targets for further signal processing and evaluation of range information about the target [1]. Based on the doppler property of electromagnetic waves, the relative speed and position of the target could be determined.

The propagation of electromagnetic waves is not affected by adverse weather conditions, and the operation of the radar is independent of environmental lighting conditions. Therefore, RADAR provides reliable data at any time of the day and in any weather conditions. Nevertheless, the detection of static objects based on RADAR data is challenging since RADAR sensors are commonly optimized to detect moving objects.

In [1] D.J. Yeong et al. provide a comparison of commonly used sensors in autonomous vehicles (Camera, LiDAR, RADAR):



Factors	Camera	LiDAR	Radar	Fusion
Range	~	~	✓	✓
Resolution	✓	~	×	✓
Distance Accuracy	~	✓	✓	✓
Velocity	~	×	✓	✓
Color Perception, e.g., traffic lights	✓	×	×	✓
Object Detection	~	✓	✓	✓
Object Classification	✓	~	×	✓
Lane Detection	✓	×	×	✓
Obstacle Edge Detection	✓	✓	×	✓
Illumination Conditions	×	✓	✓	✓
Weather Conditions	×	~	✓	✓

Figure 2.13: Comparison of commonly used sensors (Camera, LiDAR, RADAR) in AV based on technical characteristics and other external factors based on [1]

Camera data has a high resolution, color perception and is suitable for traffic light recognition, lane detection, and object classification, but under certain illumination or weather condition performance of the camera sensor could be poor. LiDAR is suitable for accurate distance estimation, object detection and does not depend on illumination conditions. Nevertheless, LiDAR does not provide color perception, velocity estimation, and is not suitable for some tasks such as line detection or traffic light recognition. RADAR provides range data, operates well in any illumination and weather conditions, and is suitable for velocity estimation.

The fusion of several modalities for environment perception in autonomous vehicles is beneficial. For example, using LiDAR in darkness can significantly improve object detection results, since there is not enough light to recognize objects based on camera data while LiDAR does not depend on the amount of light and the performance of the LiDAR-based object detection models does not decrease due to poor light conditions. At the same time, the quality of object detection based on LiDAR data could decrease during precipitation. Thus, with the use of several modalities, the ambiguities of individual data sources can be resolved and the performance of the overall system could be improved.

2.2.3 Sensor fusion strategies

Recent object detection architectures based on data from a single data source show high performance by utilizing the power of Convolutional Neural Networks (CNN). Nevertheless, a combination of data from several modalities could help to overcome shortages of individual modalities and improve the overall perception capabilities of the autonomous system.

Depending on the stage when the fusion step is performed, there are several fusion strategies: early fusion, middle (or halfway), and late fusion. In early fusion, raw data are merged at the beginning of the processing, while late fusion combines results after the processing of data in each modality. In middle fusion features that were extracted from different modalities are combined.

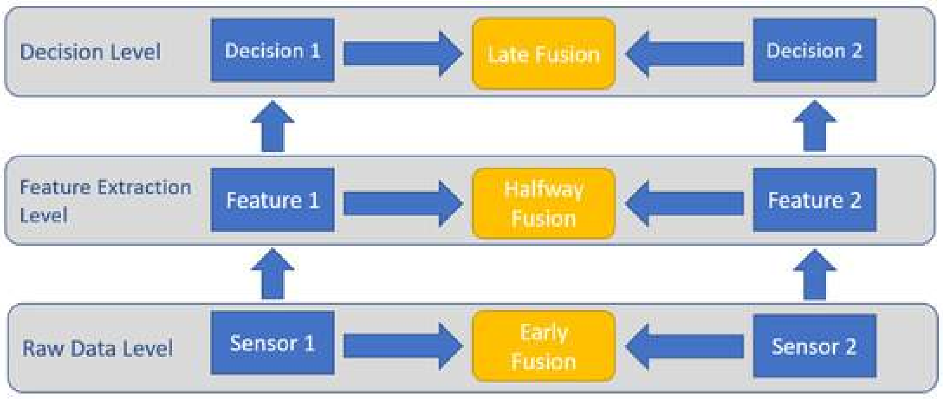


Figure 2.14: Sensor fusion strategies based on [14]

In early fusion, raw data or features extracted from the raw data of each modality could be concatenated into a joint representation and processed together. Data from each modality should be properly aligned in joint representation. Thus, cross-correlations between data from different modalities could be exploited, thereby providing an opportunity to increase the performance of the system. Nevertheless, the processing of joint representation might need more computational resources and time than the processing of data from each modality separately and performing fusion at the end of the process (late fusion).

In [38] authors of VoxelNet, first end-to-end 3D object detection achritecture [28] proposed PointFusion and VoxelFusion - 3D object detectors based on RGB camera and LiDAR PCD modalities. They extended VoxelNet, 3D voxel-based object detector that operates on LiDAR PCD, augmented PCD with semantic image features, and propose the following fusion techniques:

- PointFusion - an early fusion method where points from the LiDAR are projected onto the image plane, followed by image feature extraction from a pre-trained 2D detector. In PointFusion concatenation of image features and the corresponding points is processed by the VoxelNet architecture.
- VoxelFusion - a halfway fusion method where non-empty 3D voxels created by VoxelNet are projected to the image, followed by extracting the image features for every projected voxel using a pretrained CNN and further processing by voxel feature encoding layers. In the last stage region proposal network is utilized in order to produce 3D bounding boxes.

In [39] Kim et al. propose a YOLO-based object detection system and utilize a late-fusion strategy to combine results. YOLO-based object detectors are trained independently based on data from 3 modalities: RGB camera data, reflection data obtained from LiDAR PCD, and depth map data obtained from LiDAR PCD. Each object detector predicts bounding boxes and conditional class probability. Detection results from each modality are combined at the decision level to improve overall detection performance: weighted mean (WM) is calculated for each object to average the bounding box.

Depending on the task, one or another strategy may bring more benefits. In the current work, the late fusion strategy is utilized. This approach meets the requirements determined by Infineon Technology AG to prepare independent object detection models for each modality in order to have the opportunity to use each sensor for object detection independently. Furthermore, it is planned to use the object detection model on edge devices. Therefore, a less computationally expensive strategy is more beneficial.

2.3 Sensor Calibration for Camera and LiDAR

Sensor calibration is a foundation for sensor fusion and requires additional preprocessing in order to match data from several sensors that could be either same or from different modalities.

Precise calibration is a cornerstone for further processing steps, such as sensor fusion and implementation of such algorithms as obstacle detection. Sensor calibration notifies the autonomous system about the sensors' position and orientation in real-world coordinates by comparing the relative positions of known features as detected by the sensors [1]. In the [1] three categories of calibrations are mentioned: intrinsic calibration, extrinsic calibration, and temporal calibration.

Intrinsic calibration estimates the internal or intrinsic parameters of a sensor, e.g., focal lengths of a vision camera, which correct for systematic or deterministic aberrations (errors) [1]. Extrinsic calibration is a rigid transformation (or Euclidean transformation) that maps the points from one 3D coordinate system to another [1]. Intrinsic calibration is conducted before implementing extrinsic calibration. In this section two types of calibration that are necessary for LiDAR-Camera sensor fusion, are considered - single camera calibration based on a pinhole camera model and extrinsic LiDAR-Camera calibration.

2.3.1 Single camera calibration

The pinhole camera model is widely used for intrinsic camera calibration. The pinhole camera model describes the mathematical relationship of the projection of points in 3D world coordinate space onto a 2D image plane. The pinhole camera model assumes that the camera aperture is a point and no lenses are used and does not describe geometric distortion. Pinhole camera model visualisation[1]:

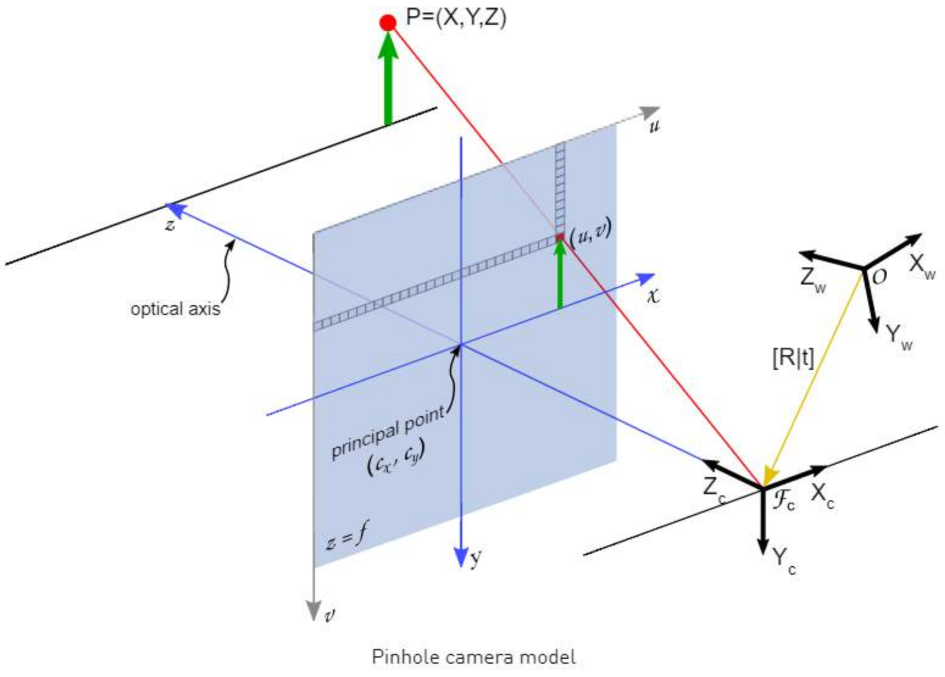


Figure 2.15: Pinhole camera model

The projection of the point from 3D world coordinates to 2D image coordinates is calculated using the following formula:

$$sm' = M_{int} M_{ext}^{W-C} M' \tag{2.2}$$

[1]Source: *https : //docs.nvidia.com/vpi/appendix_pinhole_camera.html*

Where: s - scale factor, M_{int} – intrinsic parameters of camera, M_{ext}^{W-C} – extrinsic parameters (rotation and translation) that describe transformation from 3D world coordinates to 3D camera coordinates, $m' = [u, v, 1]^T$ - homogeneous image coordinates, $M' = [x, y, z, 1]^T$ - homogeneous 3D world coordinates. Homogeneous coordinates m', M' equal to m, M correspondingly, augmented with additional coordinate W. W represents the distance from the sensor to the projection plane and reflects a scaling transformation. For the purposes of calibration W is set to 1. Homogeneous coordinates allow to express various coordinate transformations as matrix operations and therefore are used in computer vision.

$$M_{int} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ u_0 & v_0 & 1 \end{bmatrix} \tag{2.3}$$

where f_x, f_y - horizontal and vertical focal lengths respectively expressed in pixel units, u_0, v_0 - optical center (principal point) of the image.

$$M_{ext}^{W-C} = [R|T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.4}$$

where R - rotation matrix, T - translation vector.

One of the commonly utilized approaches is Zhang's model [40]. This is a camera calibration method that uses photogrammetric techniques (based on known calibration points) and self-calibration techniques (correspondence between the calibration points when they are in different positions). Zhang's model uses one of the repetitive calibration patterns with known geometry, for example, the checkerboard pattern.

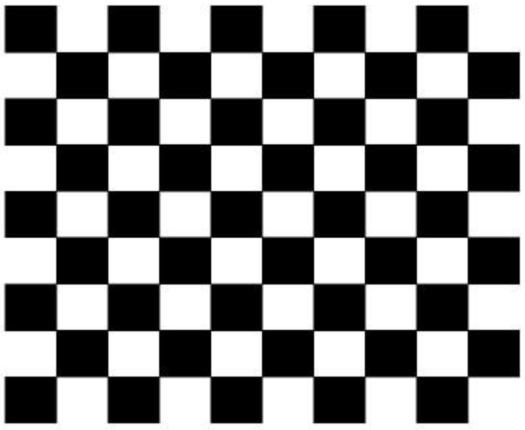


Figure 2.16: Checkerboard pattern widely used for calibration

The planar pattern is captured from several orientations (at least two). The dimensions of the checkerboard as well as the dimensions of each cell are known in advance. The known calibration points observed from a planar pattern and the correspondence between the calibration points in various positions are used to estimate the calibration matrix [40].

2.3.2 LiDAR-Camera Calibration

Monocular camera and LiDAR are the sensors that are commonly used in autonomous vehicles. Camera and LiDAR data representations have different coordinate systems and dimensionality. Calibration of camera and LiDAR is an example of extrinsic calibration and implies a rigid transformation of points from the 3D LiDAR coordinate system to the 3D camera coordinate system [1].

Transformation from 3D LiDAR coordinate space to 2D image space is defined as following:

$$sm' = M_{int} M_{ext}^{L-C} M' \tag{2.5}$$

Where: s – scale factor, M_{int} – intrinsic parameters of camera, M_{ext}^{L-C} – extrinsic parameters (rotation and translation) between LiDAR and Camera in 3D coordinate systems, $m' = [u, v, 1]^T$ - homogeneous image coordinates, $M' = [x, y, z, 1]^T$ - homogeneous 3D LiDAR coordinates.

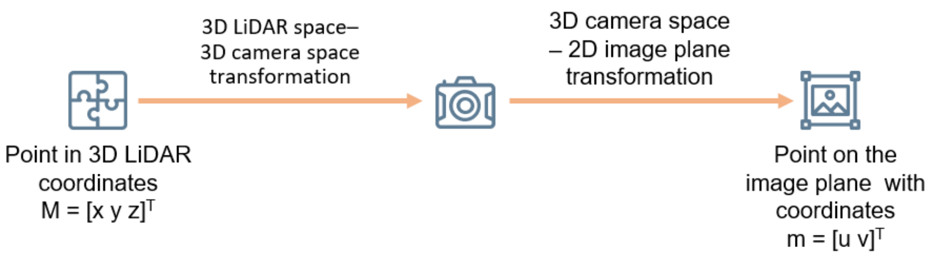


Figure 2.17: 3D LiDAR-2D image coordinate transformation

LiDAR-Camera Calibration estimates extrinsic parameters:

$$M_{ext}^{L-C} = [R|T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.6}$$

where R - rotation matrix, T - translation vector.

Intrinsic parameters of camera M_{int} are estimated using single camera calibration approaches and described above in subsection 2.3.1.

Contemporary approaches to LiDAR-Camera calibration include calibration based on repetitive planar pattern, for example [41], as well as targetless calibration, for example [42], [43].

Zhou et al. proposed a target-based extrinsic calibration approach based on a checkerboard pattern that is captured in different poses in order to obtain constraints of calibration parameters under different poses. The authors of the paper developed a calibration algorithm that combines 3D line and plane correspondences. This algorithm consists of several steps:

1. Detect the checkerboard and its boundaries in the image and calculate their corresponding 3D plane and 3D boundaries.

2. Specify a rough position for the checkerboard. Detect the checkerboard by RANSAC to get the plane and the boundary.

3. Estimate the rigid transformation ($\hat{R}^C_L, \hat{t}^C_L$), where $\hat{R}^C_L$ - estimation of the rotation matrix, $\hat{t}^C_L$ - estimation of the translation vector.

One of the disadvantages of the approach described above is that it is an iterative approach. Each new iteration of the calibration is performed independently of the others, and the result could be improved. However, the improvement of the result is not guaranteed and the required number of iterations is difficult to estimate.

G. Zhao et al proposed a deep learning-driven technique for multimodal calibration called CalibDNN [42] that require a single iteration.

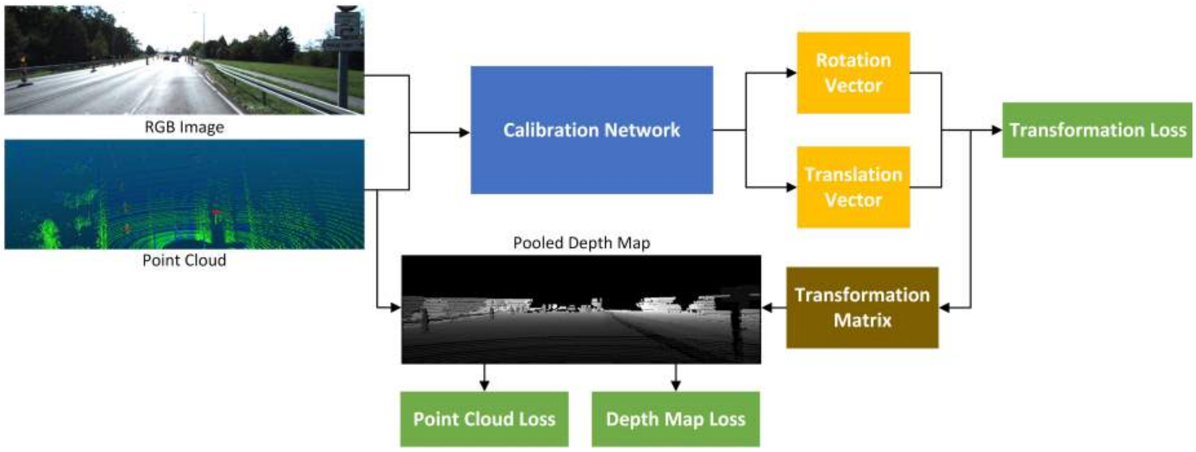


Figure 2.18: CalibDNN approach overview

This approach requires pre-calibrated data samples in order to generate ground truth data for training neural network - RGB images and 2D projections of the 3D point cloud into an image. CalibDNN architecture consists of feature extraction and feature aggregation blocks and predicts extrinsic parameters: rotation vector r =$(r_x; r_y; r_z)^T$ and translation vector t = $(t_x; t_y; t_z)^T$. The loss function is a weighted sum of three losses: transformation loss, depth map loss, and point cloud loss. Transformation loss is the L-2 norm between prediction and the ground truth separately on the rotation vector and translation vector. Depth map loss is a loss between predicted and ground truth depth maps obtained using predicted and ground truth extrinsic parameters correspondingly. Given intrinsic camera parameters, the predicted and ground truth depth maps are back-projected to the point cloud. The point cloud loss reflects the difference between the predicted point cloud and ground truth point cloud using Chamfer Distance.

The novel targetless LiDAR-Camera calibration approach proposed by B.Zhang et al. in [43] relies only on one shot and transforms multimodal calibration problem from 3D-3D to 2D-2D. The authors of the paper suggest to project LiDAR 3D point cloud into the image plane via a cylindrical projection model, to extract dense maps of depth, reflectivity, and object features and then to extract edge information from each of them based on the Canny edge detector. At the same time, edge information is extracted from the image as well based on the Sobel filter. The edge map is smoothed by applying a Gaussian kernel. Edge matching between multi-feature edge maps extracted from LiDAR point cloud and camera edge map is solved using optimization.

2.4 Infineon data acquisition setup

Researches of Infineon Technologies AG developed a multisensor setup and framework for data acquisition and storage. The multisensor setup represents a bicycle helmet with sensors mounted on it. Sensors in the setup:

- Camera Intel RealSense D415 [2];
- LiDAR: Blickfeld Cube 1[3];
- RADAR: Infineon BGT60TR13C[4];
- ToF: CamBoard pico flexx[5];
- CubeOrange with IMU (ICM-20948[6]) and GPS (Here3[7] + Here+ RTK[8] Base).

Multisensor setup is connected to laptop with the following configuration:

Parameter	Value
Model	ASUS TUF 15 FX506HM
Processor	Intel Core i5-11400H
GPU	NVIDIA GeForce RTX 3060
CUDA cores	3840
RAM	16 GB
Computing power	11.4 TFLOPS

Table 2.1: Configuration of the laptop that was used for data acquisition

The setup records data of the data types that are commonly used for autonomous vehicles' environment perception. All data are synchronized among all sensors.

In the Figure 2.19 the sensor setup for multi-modal data acquisition is demonstrated.

[2]https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf

[3]https://www.blickfeld.com/lidar-sensor-products/cube-1/

[4]https://www.infineon.com/dgdl/Infineon-BGT60TR13CDataSheet-DataSheet-v01$_0$0 $- EN.pdf?fileId = 8ac78c8c7d718a49017d94bac88e5d43$

[5]https://3d.pmdtec.com/en/

[6]https://invensense.tdk.com/download-pdf/icm-20948-datasheet/

[7]https://docs.cubepilot.org/user-guides/here-3/here-3-manual

[8]https://ardupilot.org/copter/docs/common-here-plus-gps.html

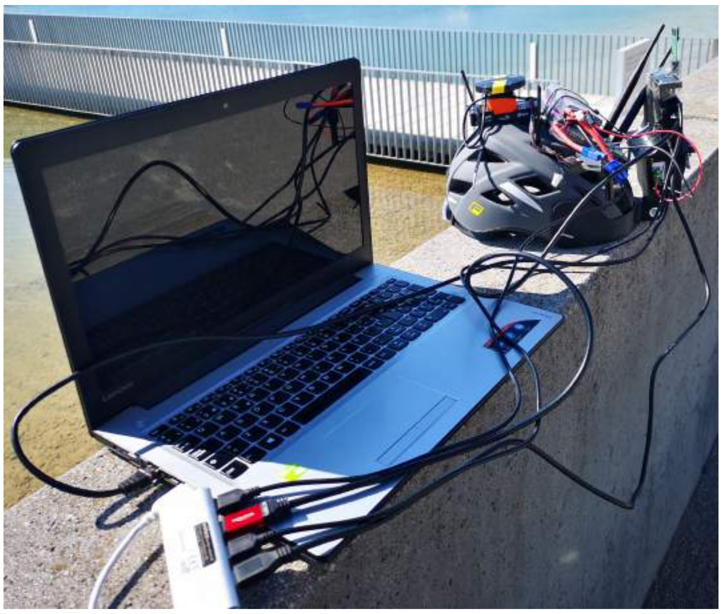


Figure 2.19: Sensor setup for data acquisition

Data acquisition was conducted at Infineon office park "Campeon" - an office complex embedded in a 62-hectare public landscaped park with 6.8 ha of water. Campeon has a good mix of environmental features including buildings, trees, and green vegetation of various sizes, which could also be considered an obstacle for autonomous vehicles. Data acquisition was conducted in compliance with the standards of data privacy - all participants of data acquisition are employees of Infineon Technologies AG who have agreed to participate in the data recording. Due to privacy issues, data acquisition in the city was not possible.

In the current work, only a monocular camera and LiDAR data is used. Data from each sensor are stored in separate folders with synchronous timestamps/ids. The framework for data acquisition and storage developed by researchers of Infineon stores LiDAR data in .csv format. The resulting .csv file contains 3D coordinates of the points and a timestamp. Reflection is not recorded.

Data acquisition results

Object detection data was acquired in 3 recording sessions at an FPS = 10.34 (2340 frames), 10.28 (4000 frames) and 7.87 (4000 frames). Since with the increase of the frame rate the number of points of LiDAR and its resolution decreases significantly, the LiDAR sensor limits the opportunity to increase the recording frequency rate. Nevertheless, an FPS = 7.8 is a sufficient frequency of data acquisition for object detection model training.

In order to diversify the data, the scenes were recorded in several locations of Campeon with different landscapes. 2340 frames were recorded in a dynamic manner (the operator was wearing the helmet with a sensor setup). Dynamic data acquisition provides higher data variability, however, due to the high sensitivity of the sensors to slight changes of the position in space, for example, due to a change in the position of the head of the operator, the remaining 8000 frames were recorded in a static manner, in 8 locations (1000 frames in each location). During static data acquisition, sensor setup was installed on a flat horizontal surface, pedestrians and cyclists were moving in the field of view of sensors at different distances from it.

In current work five classes of objects are used:

- pedestrian,
- tree,
- building,
- fence,
- cyclist.

Each class of object is represented by various entities. The resulting dataset is unlabeled. Supplementing the dataset with labels provides wide opportunities for research of the object detection and tracking models based on data from a monocular camera, stereo camera, LiDAR, RADAR, and models based on fused data from several sensors.

3 Methods

3.1 LiDAR-Camera Calibration

3.1.1 Calibration tools

The following tools were used to perform the sensor calibration:

- MatLab Singe Camera Calibrator App was used to find the intrinsic parameters of the camera,
- Matlab Camera-LiDAR Calibrator App was used to find extrinsic parameters between the camera and LiDAR.

For the aims of sensor calibration planar checkerboard with a chess pattern was created. The chessboard has a size of 8x7 cells, a cell size – 78 mm, and no padding. Data was recorded according to Matlab guidelines for calibration data acquisition and contains the images on which both the borders of the chessboard and the pattern are clearly visible.

The dataset for single-camera calibration contains images of a board with different angles of rotation without restrictions regarding the plane of rotation.

Dataset for LiDAR-camera calibration contains images of a board with different angles of rotation along the x-z plane in LiDAR coordinate space. Dataset for LiDAR-camera calibration was created with images from 6 positions relative to the sensor:

- near sensors (center, right, left),
- away from sensors (center, right, left).

The resulting dataset for calibration contained 40 sequences of 200-500 frames, 12000 frames in total, out of which 6 sessions are intended for single camera calibration and the rest of the data were designed for camera-LiDAR calibration. Such amount of data was recorded since the calibration algorithm that was utilized, is very sensitive to the slightest deviations from the expected planes of rotation indicated in the guideline. In addition, during the calibration process, there were difficulties with the time synchronization of the sensors.

3.1.2 Calibration pipeline

Generally, the extrinsic calibration of a LiDAR and a camera needs a preliminary calculation of the intrinsic parameters of the camera. The extrinsic calibration problem of a LiDAR and a camera is to estimate the relative rotation and translation between the two sensors. On the Figure 3.1 below the calibration pipeline is demonstrated.

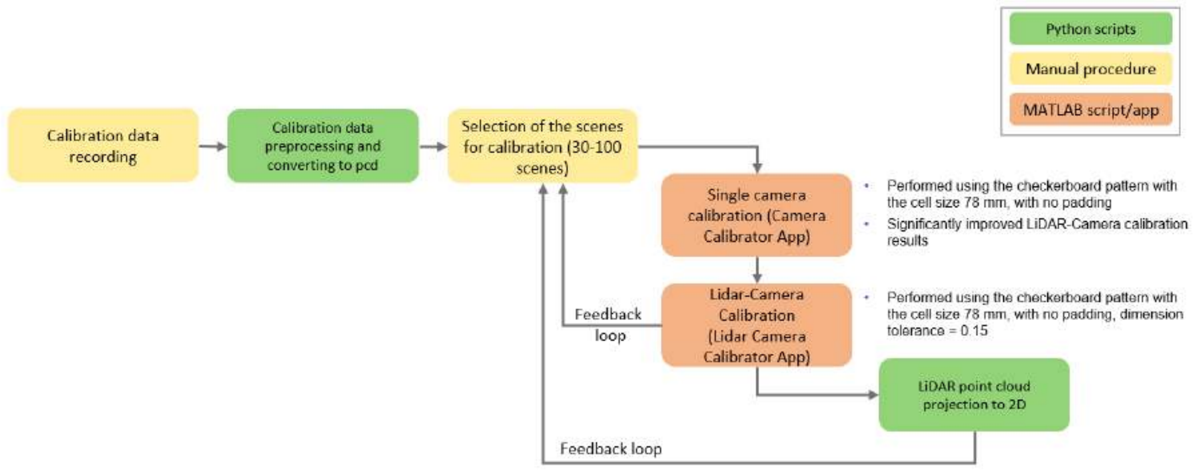

Python scripts
Manual procedure
MATLAB script/app
Calibration data recording
Calibration data preprocessing and converting to pcd
Selection of the scenes for calibration (30-100 scenes)
Single camera calibration (Camera Calibrator App)
Performed using the checkerboard pattern with the cell size 78 mm, with no padding
Significantly improved LiDAR-Camera calibration results
Lidar-Camera Calibration (Lidar Camera Calibrator App)
Performed using the checkerboard pattern with the cell size 78 mm, with no padding, dimension tolerance = 0.15
Feedback loop
LiDAR point cloud projection to 2D
Feedback loop

Figure 3.1: Camera-LiDAR calibration pipeline

The first step of calibration is data acquisition and preprocessing since LiDAR data are recorded as a .csv file that contains entire data from the acquisition session and individual frames should be extracted and converted to the .pcd format that is compatible with Matlab.

In order to use the Matlab application for calibration, a certain number of scenes that meet certain criteria should be selected. One of these criteria is an angle of rotation of the checkerboard that should not exceed 45 degrees. In order to visually assess whether the data meets the criteria and select suitable scenes, projection of the 3D LiDAR data onto the 2D image based on transformation from Cartesian to spherical coordinates was prepared with the use of Python. For this purpose, the approach to depth maps extraction suggested by J.Mendez et al. [34] was used. Coordinates of 3D points from the point cloud were projected on the image plane based on the following formulas:

$$u_i = tan^{-1}(\frac{y_i}{x_i}) \tag{3.1}$$

$$d_i = \sqrt{x_i^2 + y_i^2 + z_i^2} \tag{3.2}$$

$$v_i = cos^{-1}(\frac{z_i}{d_i}) \tag{3.3}$$

$$color = \sqrt{x_i^2 + y_i^2} \tag{3.4}$$

where $[u_i, v_i]$ - image coordinates along X and Y axes, $[x_i, y_i, z_i]$ - 3D coordinates of the point of point cloud, i = [1,...,N], where N is a number of points in Point cloud. The color was calculated based on the distance to the objects alone X-Y plane. In the Figure 3.2 example of such projection is demonstrated. Despite the LiDAR and camera data are not aligned, such projection gives an understanding of the angle of the checkerboard and allows to select data for calibration. Besides it shows that calibration is necessary in order to implement sensor fusion.

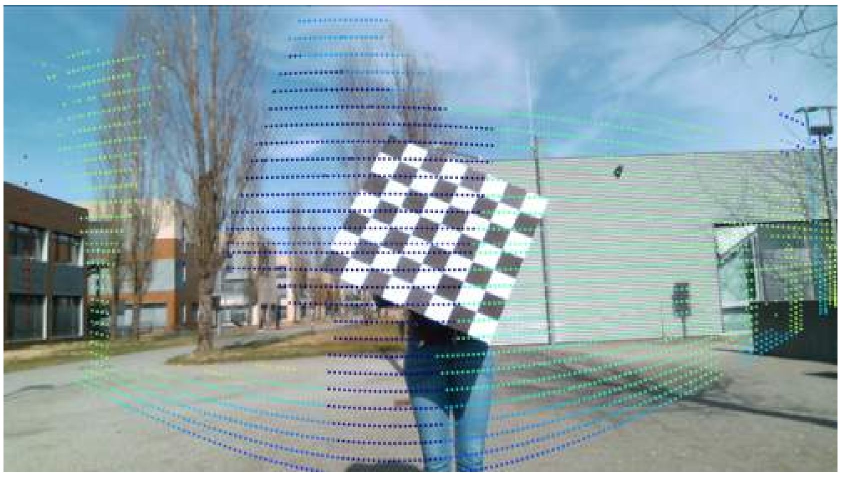


Figure 3.2: Example of LiDAR to image projection for visual estimation of the data

Subsequent steps of the pipeline include single camera and LiDAR-Camera calibration, and preparation of projections based on the intrinsic and extrinsic parameters obtained during calibration. During the calibration process, estimation of parameters is carried out on the basis of errors (see subsection 3.1.3, subsection 3.1.4). The last step of the pipeline is the visual evaluation of parameters using the LiDAR to image projection that is obtained using calibration parameters. Calibration data recorded from 6 positions relative to sensors (near the sensor and far away from the sensor; to the right, in the center, and to the left) were used for evaluation. The procedure was repeated several times in order to improve results since the overall approach of this work requires perfectly aligned data from Camera and LiDAR.

3.1.3 Single camera calibration

The aim of camera calibration is an estimation of camera parameters using images of a special calibration pattern. According to the pinhole camera model, there are two groups of parameters: extrinsic parameters (rotation and translation) that describe the projection of 3D world coordinates into 3D camera coordinates, and intrinsic parameters that describe the mapping from 3D camera coordinates into a 2D image. In the current work, calibration is performed in order to find a correspondence between data from different sensors. Therefore, extrinsic parameters of the camera are not needed and only the extraction of intrinsic parameters of the camera is considered in this section. Extrinsic calibration is performed in order to describe the mapping of LiDAR data into 3D camera coordinates and considered in the subsection 3.1.4.

Single camera calibration was performed using MatLab Singe Camera Calibrator App. The calibration workflow is demonstrated in the Figure 3.3[1]:

[1]https://de.mathworks.com/help/vision/ug/using-the-single-camera-calibrator-app.html

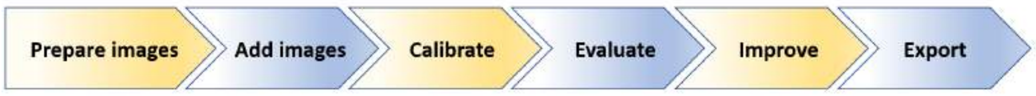


Figure 3.3: Single camera calibration steps based on official documentation

For the current work checkerboard calibration pattern was created. Images of the calibration pattern were obtained according to the guidelines provided in .

After loading the images, the application automatically detects the points of the pattern. Calibration is performed after parameters adjustment, including the selection of the camera model (pinhole camera or fish-eye). The user of the application also defines if the skew coefficient and tangential distortion should be calculated. Calibration is performed automatically based on the approach proposed in [40]. The Calibrator app visualizes a pattern keypoint detected in a calibration image and a corresponding world point projected into the same image (See Figure 3.4 below).

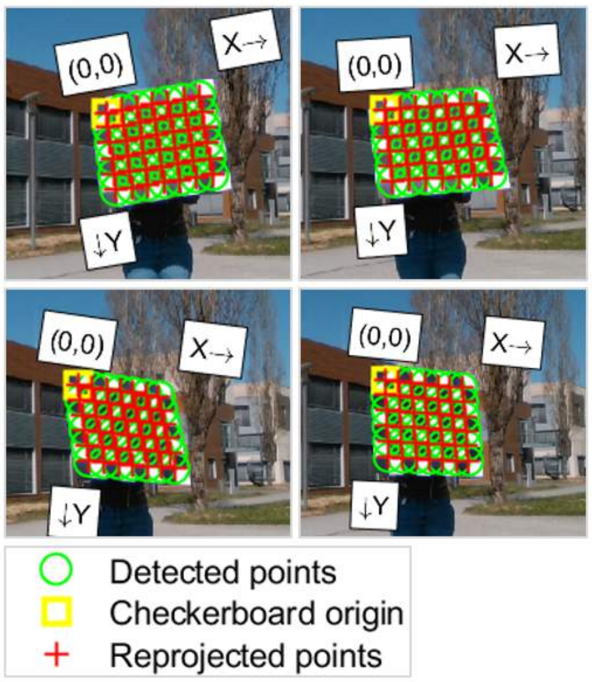


Figure 3.4: Single camera calibration detections

Intrinsic parameters as well as reprojection error for each image are obtained as a result of the calibration. Actual values of intrinsic parameters and reprojection errors are considered in the subsection 4.1.1.

Intrinsic calibration is an iterative procedure. Despite there are measures such as reprojection error that allow to estimate the accuracy of parameters (see subsection 4.1.1), the final estimation of results is possible only after extrinsic calibration, since both intrinsic and extrinsic calibration parameters define how accurate is resulting projection. More than 12 single-camera calibration sessions were performed in order to find satisfactory intrinsic parameters.

3.1.4 LiDAR-Camera calibration

This section describes the LiDAR-Camera calibration that was performed to find the extrinsic parameters that establish the correspondences between 3D LiDAR coordinate space and 3D camera coordinate space. The transformation from 3D camera coordinate space to 2D image space is defined by intrinsic parameters of the camera that were obtained at the previous step and described in the subsection 3.1.3. It is also assumed that the intrinsic parameters of the LiDAR sensor are calibrated by the manufacturer in advance.

The extrinsic LiDAR-Camera calibration is performed using a checkerboard calibration pattern based on the approach that is described in [41]. The workflow for extrinsic calibration is demonstrated on the Figure 3.5 and consists of several steps. These steps include:

- Acquisition of synchronous LiDAR and camera data that capture checkerboard pattern.
- Extraction of the 3-D information of the checkerboard from both the camera and LiDAR sensor (this step is automated in the LiDAR-Camera Calibrator App).
- Obtaining the rigid transformation matrix using the checkerboard corners and planes. The rigid transformation matrix consists of the rotation matrix R and translation vector t (this step is automated in LiDAR-Camera Calibrator App).
- Evaluation of the calibration accuracy based on the translation, rotation, and reprojection errors calculation (calculation of the errors is automated in LiDAR-Camera Calibrator App).

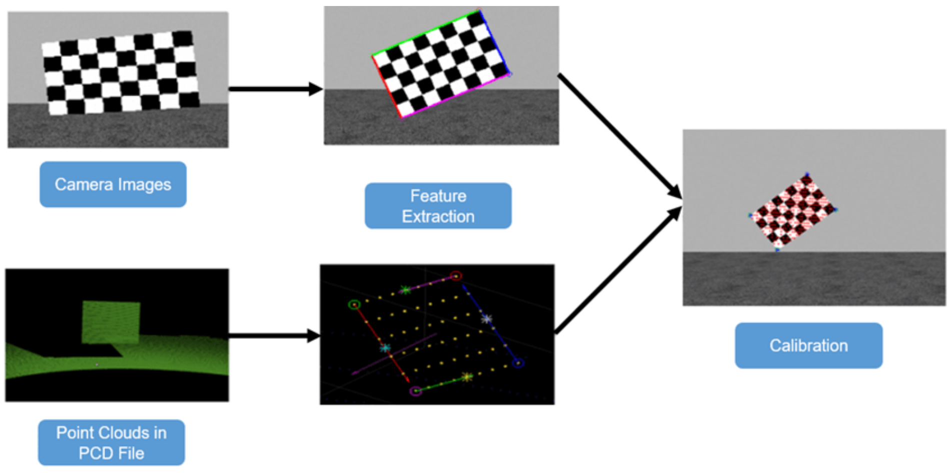


Figure 3.5: LiDAR-Camera calibration workflow

The approach that is implemented in LiDAR-Camera Calibrator App is based on the ideas proposed in [41] by Zhou et al. and is described in the subsection 2.3.2.

The LiDAR-Camera calibration procedure was repeated more than 50 times on different datasets, combinations and amounts of images as well as with different intrinsic parameters of the camera that were extracted in advance due to the high sensitivity of the algorithm to the input data. LiDAR-Camera calibration results are described in the section 4.1.

3.2 Object detection via transfer learning

3.2.1 Transfer learning approach

Deep learning became popular due to the ability of the neural network to approximate any random nonlinear function with minimal error and generalize over a large amount of data. A vast amount of data is necessary in order to conduct supervised training of the neural network. Even though there are many domain-specific datasets, the creation of a high-quality dataset for each domain where deep learning could be applied is tough. Thereby, transfer learning became one of the cornerstone techniques that enable the adaptation of neural networks that were pretrained on the data from another domain.

Transfer learning definition, provided in [44] by F.Zhuang et al.:

> A domain D is composed of two parts, i.e. a feature space χ and a marginal distribution P(X). D = {χ, P(X)}, where X is an instance set and is defined as X = {x | $x_i \in \chi, i = 1, ..., n\}$, n - number of instances. In [45] a domain is defined as the scope of application for the algorithm. A task τ consists of a label space Y and a decision function f, i.e. τ = $\{Y, f\}$. The decision function f is an implicit one, which is expected to be learned from the sample data.
>
> Given some/an observation(s) corresponding to $m^S \in N$ source domain(s) and task(s) (i.e. $\{(D_{S_i}, \tau_{S_i})|i = 1, ..., m^S\}$), and some/an observation(s) about $m^T \in N$ target domain(s) and task(s) (i.e.,$\{(D_{T_i}, \tau_{T_i})|i = 1, ..., m^T\}$)), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions $f^{T_j}(j = 1, ...m^T)$ on the target domain(s).

If m^S equals 1, the scenario is called single-source transfer learning, otherwise it is a multi-source transfer learning scenario. m^T represents the number of the transfer learning tasks. In current work, scenario with $m^S = 1$ and $m^T = 1$ is considered.

The goal of transfer learning is to learn a more accurate decision function on the target domain.

In [44] and [46] the transfer learning approaches are categorized into four groups:

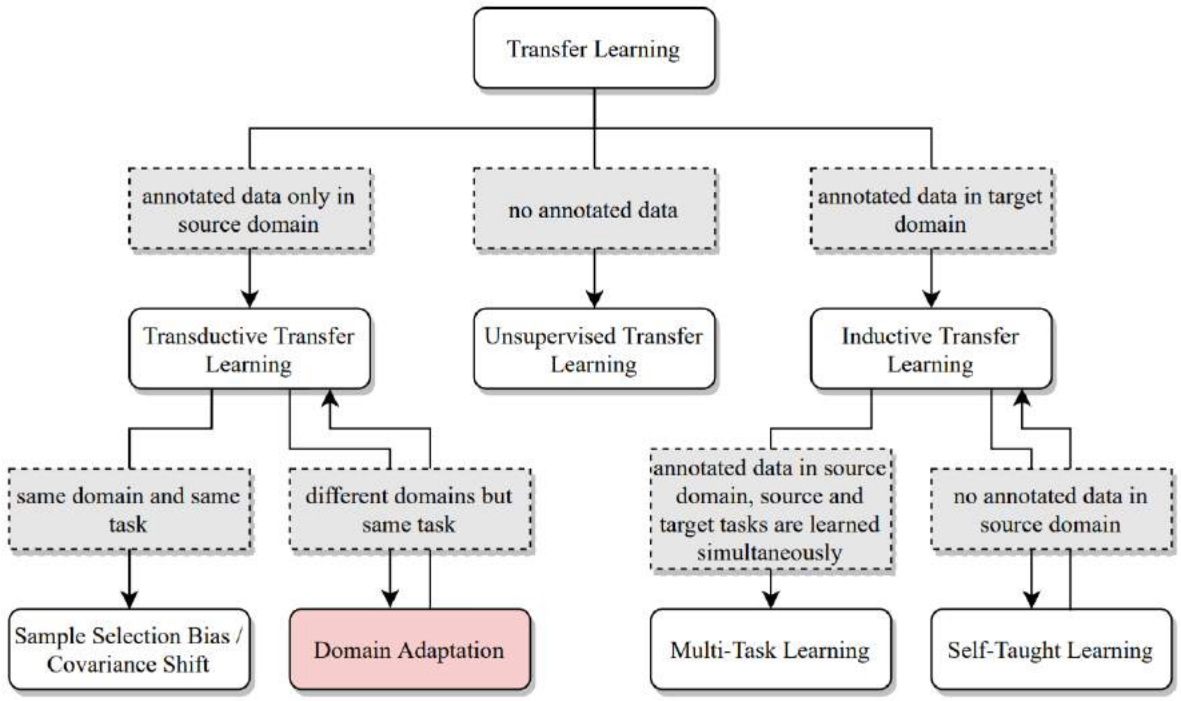
- instance-based,
- feature-based,
- parameter-based, and

- relational-based.

Instance-based transfer learning approaches are mainly based on the instance weighting strategy. Feature-based approaches transform the original features to create a new feature representation. Two subcategories could be distinguished: asymmetric and symmetric feature-based transfer learning. Asymmetric approaches transform the source features to match the target ones. In contrast, symmetric approaches attempt to find a common latent feature space and then transform both the source and the target features into a new feature representation. The parameter-based transfer learning approaches transfer the knowledge at the model/parameter level. Relational-based transfer learning approaches mainly focus on the problems in relational domains. Such approaches transfer the logical relationship or rules learned in the source domain to the target domain. [44]

Another taxonomy of transfer learning approaches is proposed by L.T.Triess et all in [45]. Depending on whether annotated data is available in source or target domain transfer learning methods are divided into 3 groups (See Figure 3.6):

- Transductive Transfer Learning (annotated data only in source domain),
- Unsupervised Transfer Learning (no annotated data),
- Inductive Transfer Learning (annotated data in target domain).


Transfer Learning
annotated data only in source domain
no annotated data
annotated data in target domain
Transductive Transfer Learning
Unsupervised Transfer Learning
Inductive Transfer Learning
same domain and same task
different domains but same task
annotated data in source domain, source and target tasks are learned simultaneously
no annotated data in source domain
Sample Selection Bias / Covariance Shift
Domain Adaptation
Multi-Task Learning
Self-Taught Learning
Figure 3.6: Taxonomy of the transfer learning methods presented in [45] by L.T.Triess et al.

In the current work Transductive Transfer Learning approach is employed in two scenarios.

In the first scenario pretrained on MS COCO object detection model is used in order to label the unlabeled Infineon dataset. In this case, the data may differ in their feature space or follow a different data distribution [45].

During the training of the object detector, the weights of the backbone obtained based on the MS COCO dataset are frozen while the weights in the neck and detection head are updated. Training is conducted in order to enable the model to recognize a specific set of object classes that does not present in MS COCO. The advantage of this approach is that a relatively small amount of data is sufficient for such training.

In the second scenario, Domain Adaptation is utilized when the information about the environment that was extracted from camera image data (labels) is used in order to train an object detection model based on LiDAR depth maps. In this case, training is carried out with updating the weights of all layers of the neural network.

The availability of a highly accurate object detection model for camera data allows to annotate a significant amount of data in a reasonable time while the projection of depth maps on the image plain enables to use labels obtained by the camera-based object detector for training LiDAR depth maps based object detection model.

3.2.2 Object detection performance estimation

Estimation of the performance of object detection models is typically based on the measure called Intersection over Union (IoU). Many object detection model use anchor-based detection methods that assume that a number of anchors (center points of the possible objects) is scattered across the feature map. Bounding boxes of various scales and aspect ratios could be built around the anchors. During training, anchors are assigned to objects (positive anchors) or background (negative anchors) by threshold their Intersection over Unions (IoUs) with the ground-truth bounding boxes.

Intersection over Union (IoU) is a measure that evaluates the overlap between ground truth and predicted bounding boxes and is widely used for the evaluation of 2D and 3D object detection algorithms performance:

$$IOU\ (Intersection\ over\ Union) = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{3.5}$$

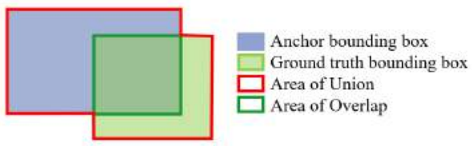


Figure 3.7: Intersection over Union

During inference, the anchors independently predict the object's bounding box where the box with the highest classification score is retained after the Non Maximum Suppression procedure.

Mean Average Precision (mAP) summarizes object detection model performance across all possible object classes. The mAP is calculated as the area under the precision-recall curve (AUC), averaged across all object classes.

In the first step Average Precision (AP) is calculated for each class. In the object detection task the settings for Precision and Recall calculation are as follows:

- True Positive (TP) detections are those detections where IoU with the ground truth (GT) is above the predefined threshold.
- False Positive (FP) detections are those detections where either the IoU with GT is below the predefined threshold or the bounding boxes that have IoU with a GT that has already been detected.
- True Negative (TN) detections are not considered since the scene is expected to contain at least one object.
- False Negative (FN) detections: those ground truth bounding boxes for which no detections were found.

Each predicted bounding box has a confidence value for the given class. The scoring method sorts the predictions by the confidence in descending order and computes the cumulative Precision and Recall for each prediction:

$$Precision = \frac{TP}{(TP + FP)} \tag{3.6}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{3.7}$$

In order to compute the interpolated Precision-Recall curve, the maximum Precision that has a corresponding Recall$'$ $\geq$ Recall is selected for each value of Recall. The Average Precision(AP) is calculated as the average of selected Precisions.

Another performance measure that is used for object detection model evaluation is F1-score. F1-score is the harmonic mean of precision and recall. It is calculated based on precision and recall values for each class via the following formula:

$$F1\_score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.8}$$

3.2.3 Object detection model architecture

YOLOv5 object detection architecture has been utilized in current work. YOLOv5 consists of 4 main parts: input, backbone network, neck, and detection head. The input contains the preprocessing of the image data, such as mosaic data augmentation and auto-learning of anchor bounding boxes. Mosaic data augmentation implies the usage of the 4-image mosaic during training instead of a single image [23] allowing learning objects in different contexts.

The Backbone is a convolutional neural network that extracts features from images. YOLOv5 uses BottleneckCSP as a backbone. CSPNet [24] (Cross Stage Partial) Network architecture is based on DenseNet [47], which was designed to avoid vanishing gradient problem in deep neural networks. In DenseNet, each layer is concatenated with additional inputs from all preceding layers. CSPNet separates the feature map of the base layer into two parts, one part goes through a dense block and a transition layer; the other part is then combined with the transmitted feature map to the next stage.

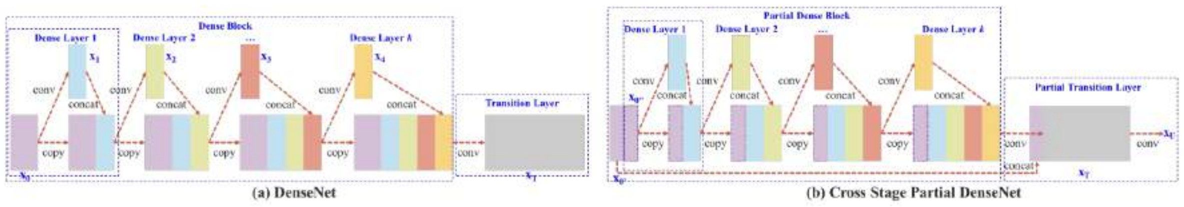


Figure 3.8: DenseNet and CSPNet architecture

YOLOv5 uses SPP [25] (Spatial Pyramid Pooling) structure in the backbone in order to extract features of different scales and generate a multi-scale feature map to improve detection accuracy.

YOLOv5 neck uses PAN [26] (Path Aggregation Network) structure in order to aggregate extracted features.

YOLOv5 architecture is demonstrated in the Figure 3.9, Figure 3.10, and Figure 3.11 below.

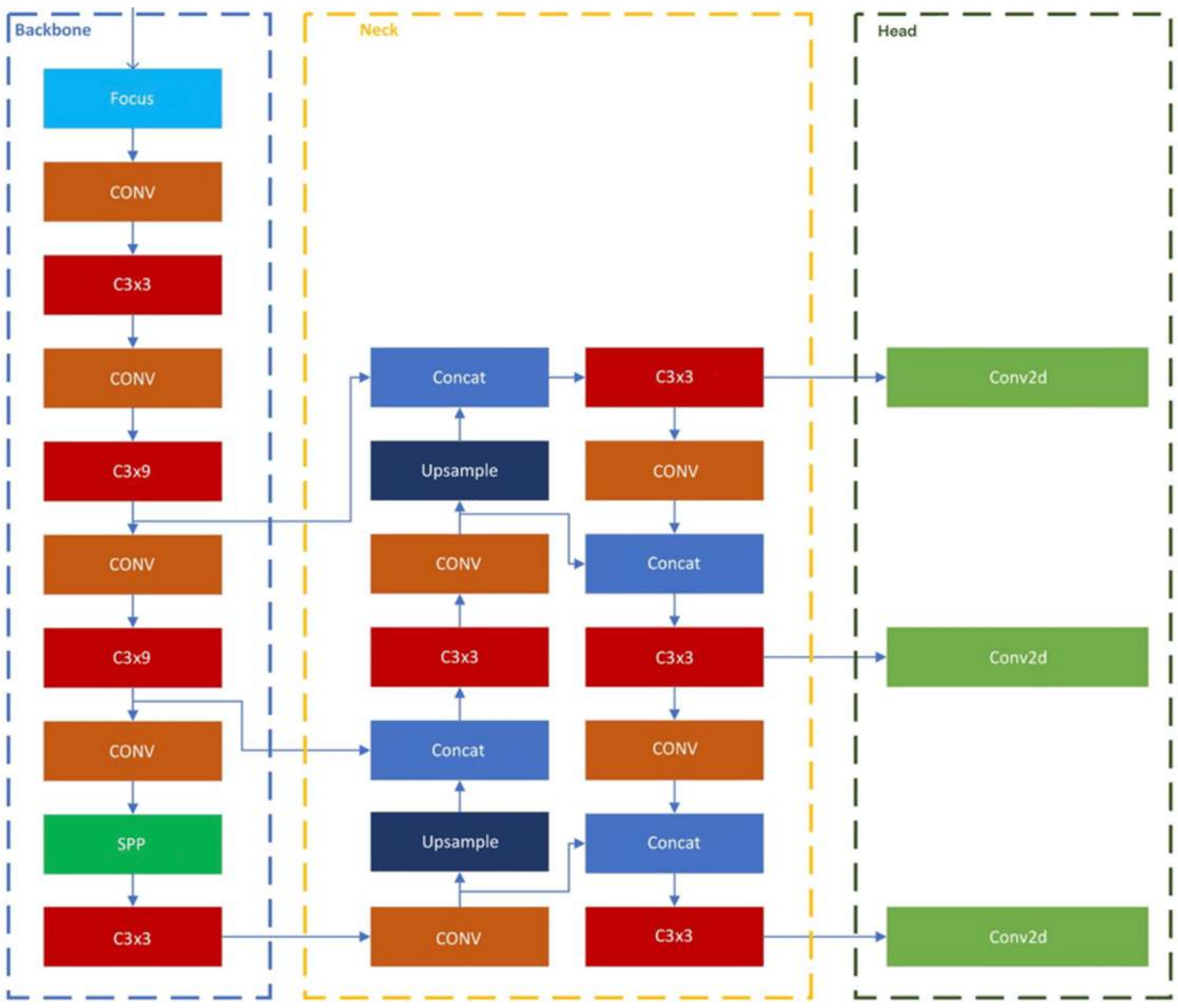


Figure 3.9: YOLOv5 architecture overview based on [48]

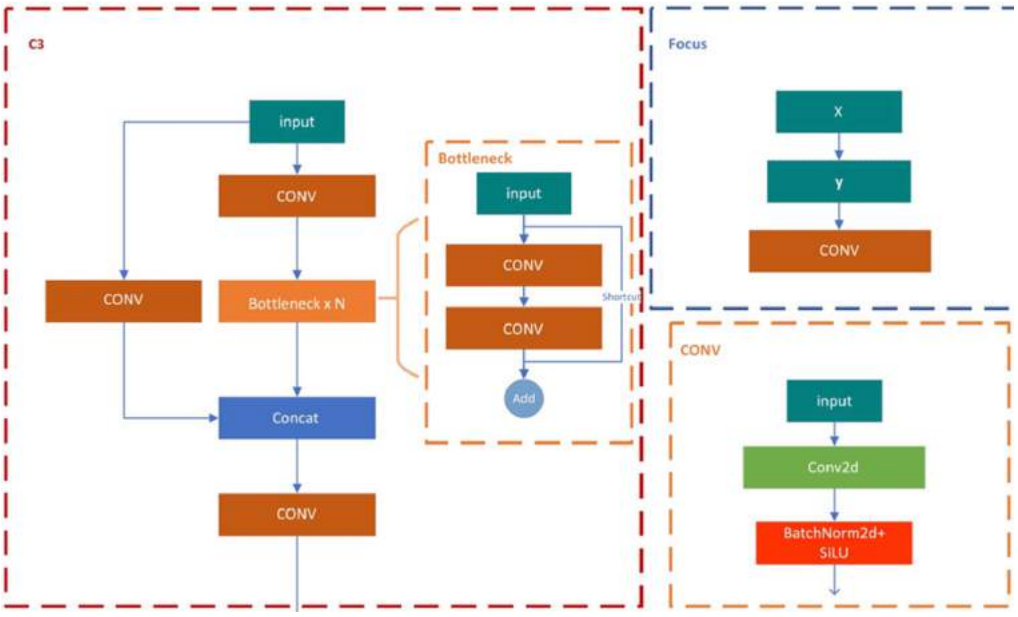


Figure 3.10: YOLOv5 architecture overview based on [48]. Building blocks

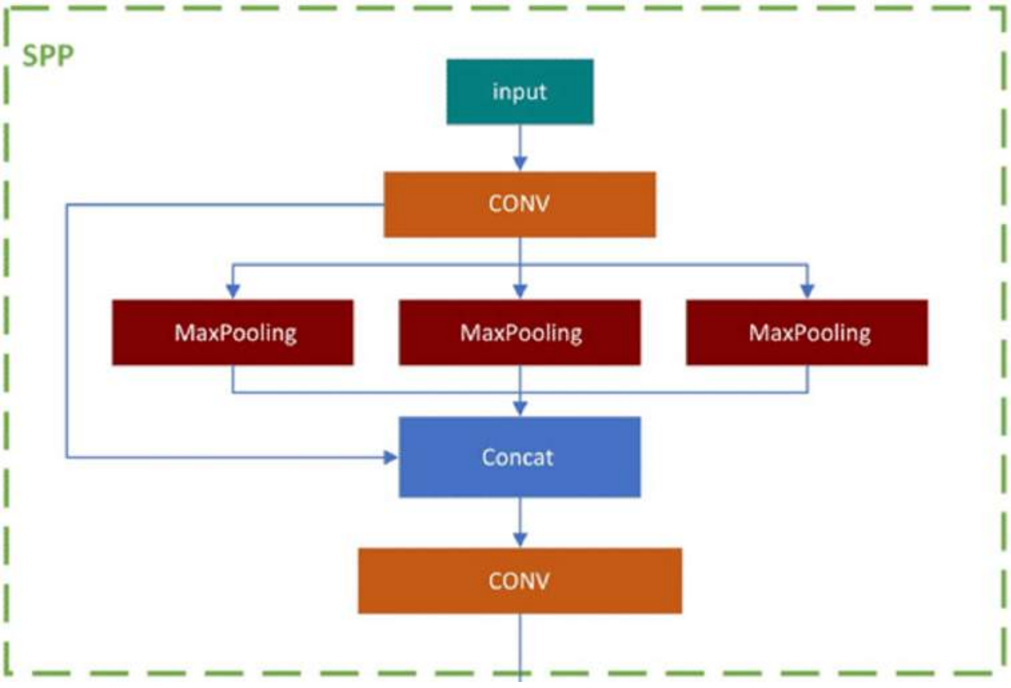
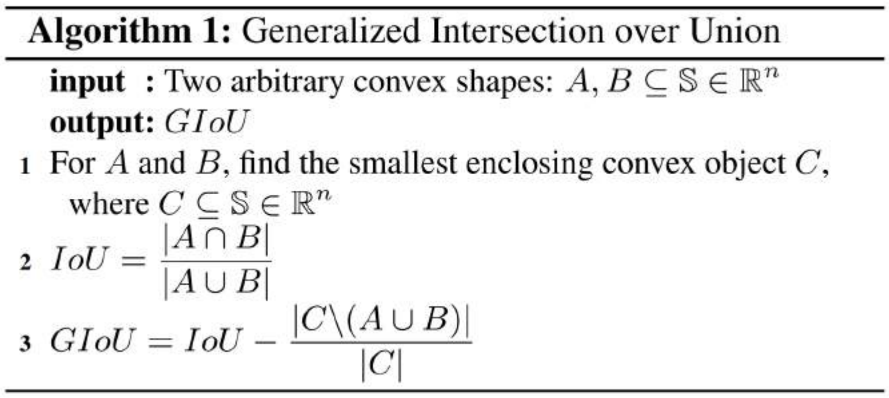


Figure 3.11: YOLOv5 architecture overview based on [48]. SSP

YOLOv5 uses YOLOv3 head with GIoU-loss [49].

IoU is a normalized measure that is invariant to the scale and due to this property, is usually used as a basis for performance measures calculation in order to evaluate object detection algorithm performance. Nevertheless, if two objects do not overlap, the IoU value is zero and does not reflect how far the two shapes are from each other. In this case, if IoU is used as a loss, its gradient will be equal to zero and cannot be optimized. GIoU extends the concept of IoU and covers non-overlapping cases as well [49]. GIoU is calculated according to the algorithm provided in [49]:



Algorithm 1: Generalized Intersection over Union

```
input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$
output: $GIoU$
1 For $A$ and $B$, find the smallest enclosing convex object $C$,
    where $C \subseteq \mathbb{S} \in \mathbb{R}^n$
2 $IoU = \frac{|A \cap B|}{|A \cup B|}$
3 $GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$
```

Figure 3.12: GIoU calculation based on [49]

For two arbitrary convex shapes (volumes) $A, B \subset S \in R_n$, we first find the smallest convex shapes $C \subset S \in R_n$ enclosing both A and B. A ratio between the area occupied by C excluding A and B is calculated and divided by the total area occupied by C. This represents a normalized measure that focuses on the empty area between A and B. GIoU value is calculated as IoU minus the ratio.

GIoU loss preserves the main properties of IoU and addresses the situation when there is no intersection of predicted and ground truth bounding boxes. Figure 3.13 below demonstrates a correlation between GIoU and IoU. In case of intersection of the predicted bounding box and ground truth bounding box, GIoU is equal to IoU, while in the case of no intersection between predicted and ground truth bounding boxes GIoU reflects the distance and could be used for model optimization while IoU is equal to zero.

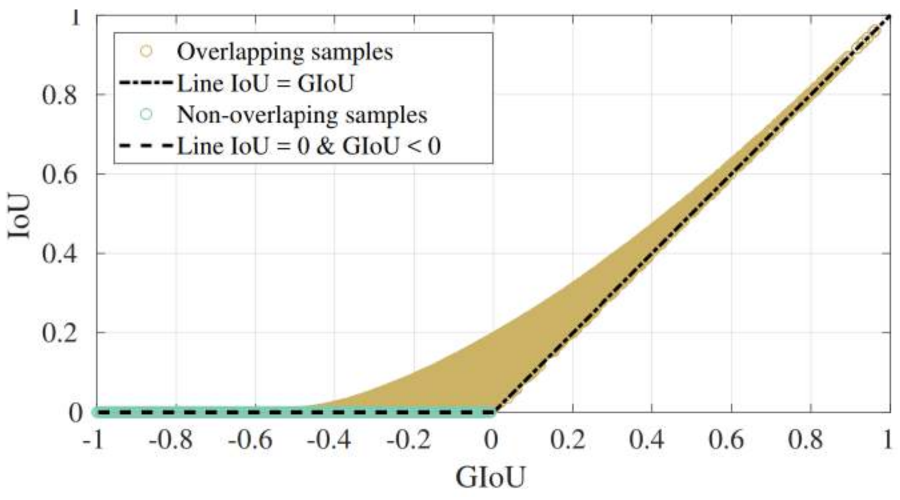


Figure 3.13: Correlation between GIoU and IOU for overlapping and non-overlapping samples based on [49]

The YOLOv5 has several variants of architecture that differ mainly by their parameters size: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), YOLOv5x (extra large) for training with an image size of 640x640 pixels. Additional series of models called P6 (YOLOv5n6, YOLOv5s6, YOLOv5m6, YOLOv5l6, YOLOv5x6) is created for training on the image of higher resolution 1280x1280. The P6 series includes an extra output layer for the detection of larger objects. P6 models benefit the most from training at higher resolution and produce better results. YOLOv5 models vary not only in the number of parameters, but also show different inference speed and accuracy.

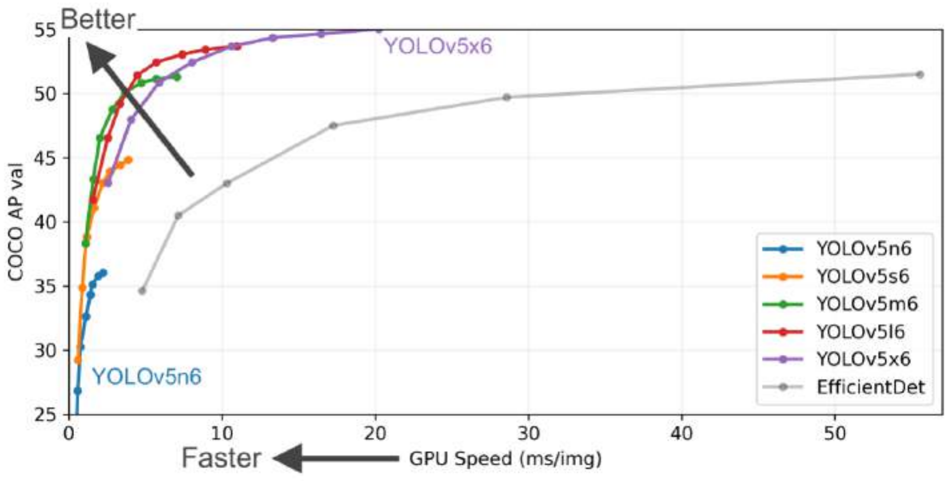


Figure 3.14: YOLOv5 (P6 series) models comparison based on [9]

All experiments in the current work were conducted on the architecture YOLOv5m6 (35.7 million parameters) since it provides a good trade-off between speed and accuracy.

3.3 Label filtering and adaptation for LiDAR based model training

3.3.1 Label filtering approach

Label filtering and adaptation is an important step that is necessary in order to prepare high-quality data for depth maps-based object detection model training.

It is more important for autonomous vehicles to detect and avoid objects that are closer in comparison to far away objects. It is assumed that close objects have a larger size in the image. Therefore, label filtering is aimed at keeping such objects for model training and elimination of small, far, or highly occluded objects at first approximation to this challenge, while in the future the model can be improved for more accurate detection of the small or highly occluded objects as well.

Labels are filtered according to the following considerations:

- **Label filtering based on the field of view,**
  Since the camera and LiDAR field of view vary in the Infineon setup, some objects that are present in image data are not visible for LiDAR. Labels for such objects should be excluded from consideration for LiDAR-based object detection model training.

- **Label filtering based on occlusion rate,**
  Detection of highly occluded objects is a complex task that requires a robust approach and they are excluded for simplification of the development and validation of the depth map-based OD model. In future work it is possible to split the objects into 3 categories of difficulty related to the occlusion rate similar to KITTI dataset [6].

- **Label filtering based on number of points inside the bounding box,**
  Objects that are represented by the small number of points either have a small size and are located close to the sensor, located far from the sensor setup or highly occluded. After experimental exploration, a minimum threshold value for the number of points is set to N = 30 in order to exclude difficult cases for model training. The minimum possible value has been set, which nevertheless allows small or partially visible objects to be left.

- **Bounding box adjustment based on the assumption about the object location.**
  Despite the calibration of the sensors, the data from the camera and LiDAR do not match completely, especially for moving objects. The reason lies in the difference in technologies: while the camera takes a picture almost instantly, scanning the environment by LiDAR takes more time.

  The approach is based on point cloud clustering and the allocation of clusters to the boundaries of objects. In order to find the location of the object, the clustering of the 3D point cloud was carried out, followed by the allocation of the clusters to bounding boxes extracted by camera-based object detector. Then the bounding boxes of the objects were adjusted based on the location of the 2D projection of the cluster.

Label filtering and adaptation algorithm is described in the following section in more detail.

3.3.2 Label filtering algorithm

The label filtering algorithm is presented in the scheme below. The first step is the ground points extraction and removal (See subsection 3.3.3 below). This approach ensures more accurate clustering in the next step as well as a more accurate calculation of the number of points within the bounding boxes that may belong to the object. After ground extraction, coarse filtering of labels is performed (See subsection 3.3.4 below) as well as clustering of non-ground 3D points based on the DBSCAN algorithm. The last step of the algorithm is the allocation of clusters (See subsection 3.3.5 below).

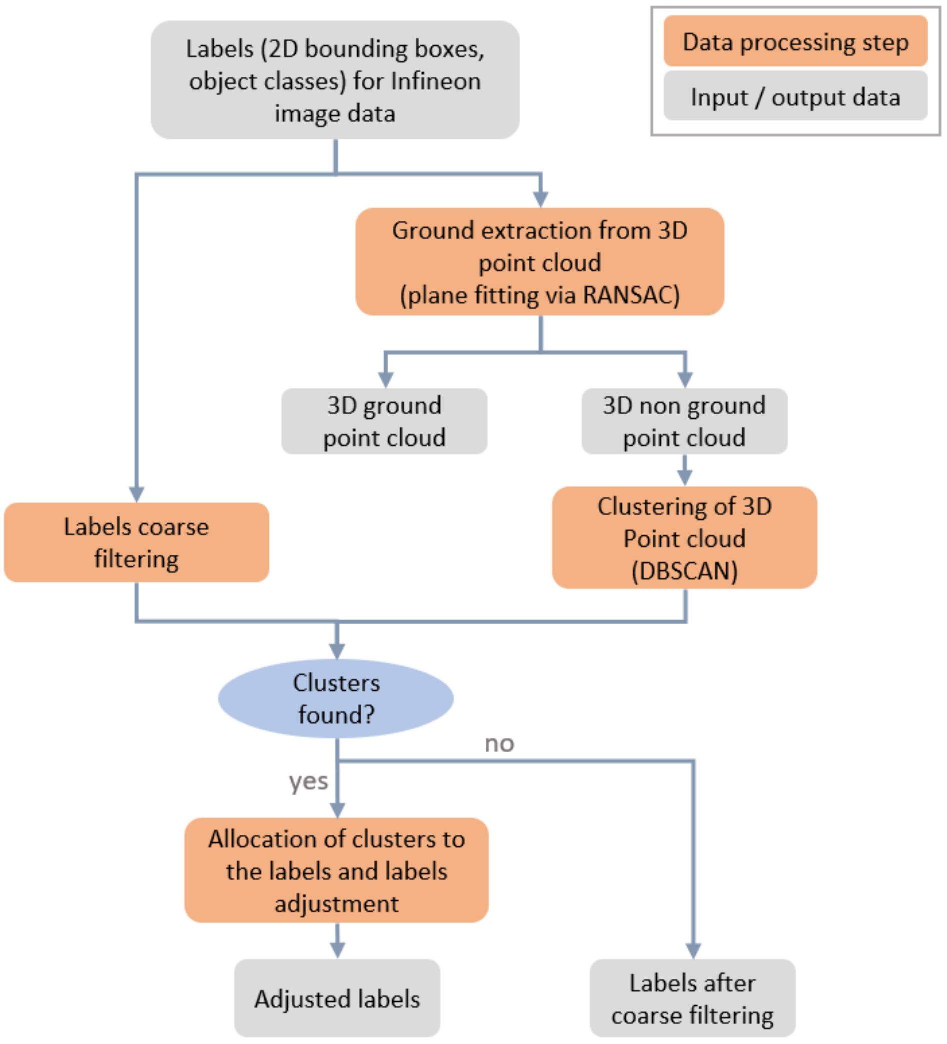

Labels (2D bounding boxes, object classes) for Infineon image data
Data processing step
Input / output data
Ground extraction from 3D point cloud (plane fitting via RANSAC)
3D ground point cloud
3D non ground point cloud
Labels coarse filtering
Clustering of 3D Point cloud (DBSCAN)
Clusters found?
no
yes
Allocation of clusters to the labels and labels adjustment
Adjusted labels
Labels after coarse filtering

Figure 3.15: Labels filtering and adjustment

3.3.3 Ground segmentation

Ground segmentation and removal is an important step that has a great impact on the results of the next steps - point cloud clustering and cluster allocation to labels. It allows to identify and exclude ground points that do not belong to any object. Besides, the density of points in the point cloud is uneven and for objects near the LiDAR sensor density is very high. Ground removal allows to avoid the situation when close objects together with the land on which they are located, are treated by the density based clustering algorithm as a single cluster. The example of such situation is demonstrated in the figure below (See Figure 3.16).

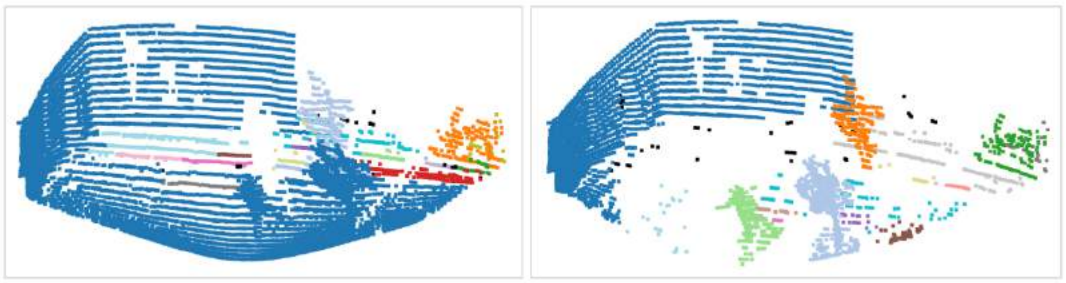


Figure 3.16: Example of point cloud clustering without and with preliminary ground segmentation.

On the left picture on the Figure 3.16 clustering was conducted without preliminary ground segmentation. As a result, the density-based clustering algorithm classified most of the points as the same cluster. Moreover, some ground points formed several clusters that further could be incorrectly used for allocation to object labels.

On the right picture on Figure 3.16 ground segmentation and removal were conducted before clustering. As a result, objects such as a person, trees, etc. were identified as separate clusters by the algorithm. However, the picture shows that the points belonging to the ground are still present in the scene and form several separate clusters. Unfortunately, it was not possible to eliminate such points completely in all the frames.

In the current implementation, RANSAC is utilized for ground segmentation since this algorithm is often used for plane fitting on point cloud data and has shown itself as a reliable approach.

Ground plane fitting with RANSAC

RANSAC [50] (RANdom Sampling and Consensus) is an iterative approach for estimation parameters of a mathematical model from data that contains outliers via repeated random sub-sampling. RANSAC assumes that data contains both inliers and outliers, the voting scheme is utilized by RANSAC in order to find the optimal model parameters.
RANSAC is widely used for ground plane fitting based on point cloud data. The algorithm included the following steps that are repeated k times, where k - number of iterations:

Step 1. Sampling of the set of n points from data uniformly and at random, where n - minimum number of points, that is required for a model of interest. For plane fitting a minimum number of 3 points are necessary.

Step 2. Model fitting to the set of n points. For the case of plane fitting, plane equation parameters are calculated. The plane equation in the Cartesian form:

$$ax + by + cz + d = 0 \tag{3.9}$$

Constants a, b, c, and d could be derived from following equations, if n=3 points are sampled from the dataset:

$$a = ((y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_2)) \tag{3.10}$$

$$b = ((z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_2)) \tag{3.11}$$

$$c = ((x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_2)) \tag{3.12}$$

$$d = -(ax + by + cz) \tag{3.13}$$

Step 3. Calculation of the deviation of all data points outside the sample of n points from the fitted model. For the plane fitting case, a Cartesian distance is calculated. The distance from the point to the fitted model is tested against threshold t. If the distance is smaller than t, the point is considered an inlier, otherwise, the point is considered an outlier.

For the case of plane fitting, the distance from the point with coordinates (x_i, y_i, z_i) to the plane is calculated in the following way:

$$Distance = \frac{ax_i + by_i + cz_i + d}{\sqrt{a^2 + b^2 + c^2}} \tag{3.14}$$

Step 4. If there the number of inliers for the model is larger than the minimum number of inliers to consider the model as a good fit, then the model is added to the collection of good fits.

After these steps are completed the best model is chosen from the collection of good fits based on the fitting error as a criterion.

The examples of the ground segmentation via RANSAC (k=2000, n=3, t = 0.15) are demonstrated in the Figure 4.21, page 78.

Iterative approach to ground plane fitting

RANSAC is a non-deterministic algorithm since its result depends on random samples. It could exhibit different behaviors on different runs even for the same input point cloud. Therefore, it is necessary to control the results obtained by the algorithm.

In order to check if the fitted plane via RANSAC is (or could be) a ground plane, the angle α between the fitted plane and the X-Y plane in LiDAR coordinates has been checked.

The angle between two planes is calculated according to the following formulas in the Cartesian form:

Given that plane equations in Cartesian form $a_1x + b_1y + c_1z + d_1 = 0$ and $a_2x + b_2y + c_2z + d_2 = 0$

$$\cos\alpha = \frac{a_1a_2 + b_1b_2 + c_1c_2}{\sqrt{a_1^2 + b_1^2 + c_1^2}\sqrt{a_2^2 + b_2^2 + c_2^2}} \tag{3.15}$$

Considering that the second plane is the X-Y plane and its equation is $0x + 0y + 1 * z = 0$, the formula could be simplified in the following way:

$$\cos\alpha = \frac{c_1}{\sqrt{a_1^2 + b_1^2 + c_1^2}} \tag{3.16}$$

Angle between two planes:

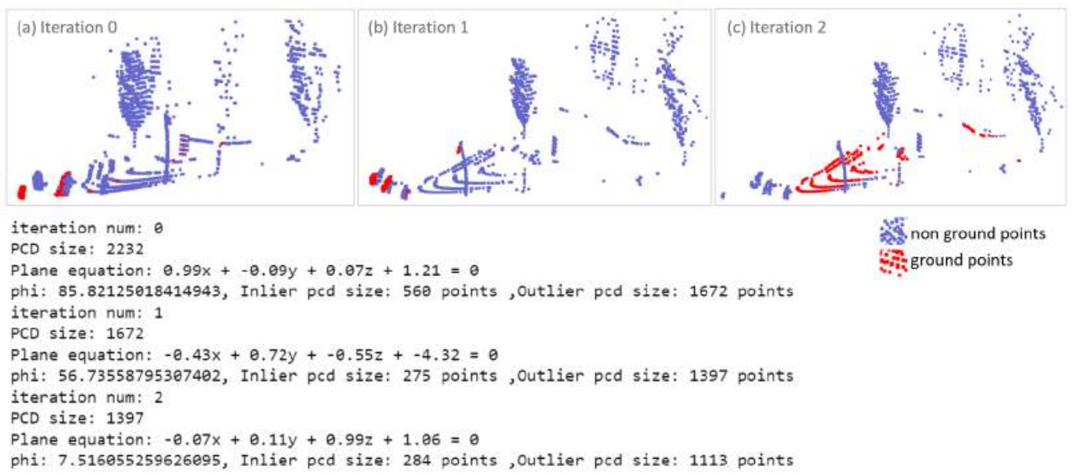
$$\alpha = \cos^{-1}(\cos\alpha) \tag{3.17}$$

The reference value of $30°$ has been chosen based on the analysis of the collected data: since the data was collected in a static and dynamic manner, the inclination angle between the ground plane and the X-Y plane varies. In most cases, the value of the angle α is in the range $[5°; 10°]$. However, there are also exceptional cases, when the angle exceeded the specified values. It happened when the operator wearing a helmet with sensors setup tilted the head. In order to cover all cases, including exceptional cases the reference value of $30°$ was used.

If for the found plane $\alpha \geq 30°$, the inlier points are excluded from further consideration and the RANSAC plane fitting is repeated on points that are identified as outliers.

The procedure is repeated until a plane with α less than $30°$ is found, but no more than 10 times. Experiments have shown that as a rule the ground plane is found in 1-3 iterations, in exceptional cases, up to 7-8 iterations are required. Therefore, the number of iterations equal to 10 covers almost all cases. The search is also terminated if the number of remaining outlier points is less than 500. Considering that the average point cloud consists of 5000 points and the ground plane includes a significant number of points that often exceeds 1000 points, it is unlikely that in the remaining 500 points ground plane could be found. At the same time the chance that, following the formal criteria, the algorithm will find a plane that will satisfy the criteria, but will not be the ground, increases.

An example of an iterative ground segmentation result is shown in Figure 3.17. For the given point cloud first and second iteration resulted in the fitted planes with $\alpha_1 = 85.82°$ and $\alpha_2 = 56.73°$ respectively. Ground plane was found after 3 iterations of RANSAC with $\alpha_3 = 7.51°$.



```
iteration num: 0
PCD size: 2232
Plane equation: 0.99x + -0.09y + 0.07z + 1.21 = 0
phi: 85.82125018414943, Inlier pcd size: 560 points ,Outlier pcd size: 1672 points
iteration num: 1
PCD size: 1672
Plane equation: -0.43x + 0.72y + -0.55z + -4.32 = 0
phi: 56.73558795307402, Inlier pcd size: 275 points ,Outlier pcd size: 1397 points
iteration num: 2
PCD size: 1397
Plane equation: -0.07x + 0.11y + 0.99z + 1.06 = 0
phi: 7.516055259626095, Inlier pcd size: 284 points ,Outlier pcd size: 1113 points
```

Figure 3.17: Iterative approach to ground segmentation

3.3.4 Labels coarse filtering

Label coarse filtering is basic filtering based on the fact that in the current sensor setup camera has a wider field of view than LiDAR and thus some objects that were detected based on image camera data do not present in LiDAR point cloud and thus are irrelevant for LiDAR based object detector. Besides, some objects are highly occluded or located far from the sensor setup and represented by just a few points in the point cloud. Detection of such objects is a complex task, they are excluded from the training of the current model for simplification.

Labels are considered in the order from the nearest object to the furthest one based on the assumption that the closer the object is located to the sensor setup, the higher will be the value y_{max} - lower border of the bounding box on the image, considering that the (0,0) is a upper left corner of the image. Labels are considered consequently. The first step of the coarse filtering is a check of the number of projected points inside the bounding box. If the number is lower than the minimum number of points, then the label is eliminated from consideration.

The next step is an adjustment of the bounding box to the LiDAR field of view and a check of the non-occluded area. The non-occluded area is calculated with respect to the bounding boxes of the objects in front of the current bounding box (bounding boxes with higher y_{max}).

$$\text{non occluded area} = \text{number of non occluded pixels} \tag{3.18}$$

$$\text{non occluded rate} = \frac{\text{non occluded area}}{(x_{max} - x_{min})(y_{max} - y_{min})} \tag{3.19}$$

where $x_{min}, x_{max}, y_{min}, y_{max}$ - minimum and maximum coordinate of the bounding box along x and y axes on the image plane respectively. In case the bounding box is occluded to a very high extent or there are points only in the occluded part of the bounding box, the bounding box is eliminated. Label coarse filtering algorithm is shown in the scheme below (See Figure 3.18).

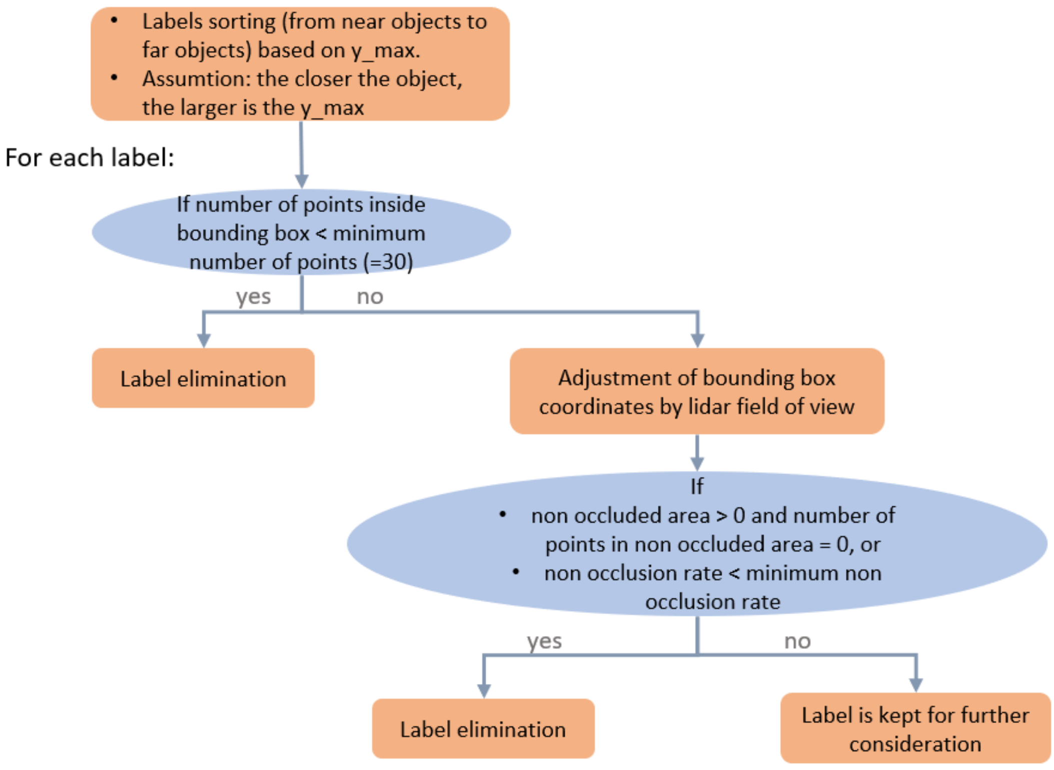

Labels sorting (from near objects to far objects) based on y_max.
Assumtion: the closer the object, the larger is the y_max
For each label:
If number of points inside bounding box < minimum number of points (=30)
yes
no
Label elimination
Adjustment of bounding box coordinates by lidar field of view
If
non occluded area > 0 and number of points in non occluded area = 0, or
non occlusion rate < minimum non occlusion rate
yes
no
Label elimination
Label is kept for further consideration

Figure 3.18: Labels coarse filtering

3.3.5 3D point cloud clustering and cluster allocation

Clustering of 3D point cloud with DBSCAN

DBSCAN (Density-based spatial clustering of applications with noise) - density-based clustering non-parametric algorithm. Given a set of points in the space, DBSCAN groups together points with many nearby neighbor points, marking as outliers the points that lie alone in low-density regions.

Hyperparameters:

- *eps* - the radius of the neighborhood,
- min_points - minimum number of points required to form a cluster.

DBSCAN is a recursive algorithm that starts at a random point and analyzes the number of points in the neighborhood *eps* of the current point and each point in the neighborhood and classifies points into 3 categories:

- core points: a point p_i is a core point if at least min_points points are within distance *eps* of it (including p_i).
- reachable points: a point q_i is a reachable point if it is located within a distance *eps* to the core point.

- outliers: all points that are not reachable from any other point are outliers or noise points.

Core points and reachable points form a cluster. Hence each cluster contains at least one core point; non-core points are a part of a cluster but have the number of points in *eps* that is smaller than min_points and are called "edge" points.

The main advantage of this algorithm over other widely used clustering algorithms is that it does not require specifying the number of resulting clusters. Hyperparameters are set based on the structure of the underlying data. This approach ensures accurate results of the algorithm on homogeneous data. However, if the point cloud contains groups of points with different structure (density of points), the algorithm may result in inaccurate clustering (See section subsubsection 3.3.5).

Iterative clustering on point cloud

Any clustering algorithm is unsupervised and its results depend on the data structure and applied hyperparameters values. In the case of the DBSCAN algorithm, such hyperparameters are epsilon (the radius of a neighborhood) and a minimum number of points in epsilon.

The algorithm uses the values of these hyperparameters to the entire point cloud, however, the distance between neighbor points for near and far objects could greatly vary. Thus, using the same values of epsilon and a minimum number of points for the entire point cloud might give incorrect results: with the same value of epsilon and the minimum number of points in epsilon, with low epsilon, distant objects can be classified as outliers, and with high value - objects located near the sensor could be combined into one cluster.

In order to solve this problem, clustering with DBSCAN is performed in 2 iterations:

Step 1. Clustering of the entire point cloud with the hyperparameters values eps = 0.5, min_points = 10 that allows to cluster near objects in separate clusters.

Step 2. Clustering of the outlier points with the hyperparameters values eps = 1.5, min_points = 10 that allows to cluster distant objects more accurately.

In the Figure 3.19 results of step 1 and step 2 are demonstrated. After DBSCAN clustering (step 1) several large distant objects, such as trees and a building, fell into the outlier group (marked in black color). After step 2 the problem is solved - such objects are clustered correctly.

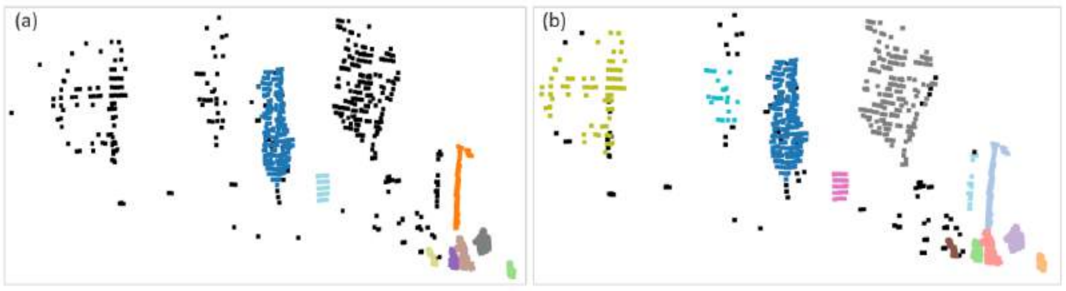


Figure 3.19: Iterative clustering on point cloud: (a) Step 1, (b) Step 2.

Point cloud clusters allocation to labels and adjustment of labels

Cluster allocation to labels and final adjustment of labels is the most important step in the filtering algorithm since the final quality of the labels depends on the correctness of the allocation. Since at this stage there is limited information about the clusters based on the point cloud and about the location of the objects based on the camera data, the correct allocation of clusters to labels for all possible cases (different object classes, varying scales of objects, areas of the scene with a highly dense objects) is a complex task.

Despite the clustering is carried out in 3D space, the allocation of clusters to 2D labels is conducted in 2D image space based on the 2D projections of clusters.

At the allocation step labels are considered sequentially in the order from the nearest object to the furthest one based on the assumption that the closer the object is located to sensor setup, the higher will be the value y_{max} - lower border of bounding box on the image, considering that the (0,0) is a upper left corner of the image.

The algorithm of the clusters to labels allocation is described below.

Given that N - total number of labels after camera-based object detection:

For $label_i$ **in** [$label_1$,...,$label_N$]:

1. Extend bounding box by a fraction (extension by 40%).

   Note: At the next step of the algorithm, a set of clusters is selected, which are considered candidates for cluster-label allocation. And one of the selection criteria is that the center of the cluster must be located inside the bounding box. An increase in bounding box size allows to find a correspondence between the objects in the image and the projection of the LiDAR 3D points in case of a lack of camera and LiDAR scene synchronization. The value of 40% has been chosen after a series of experiments in order to cover the most complex cases in the dataset.

2. Find all clusters that
   - have a center inside the extended bounding box,
   - have points density rate > minimum points density rate.

   Note: Points density rate = number of cluster points/(bounding box height * bounding box width). The minimum point density rate is set to 0.0007 based on the results of experiments. Such value allows to keep relatively small clusters in the case when due to different field of view, the object in LiDAR is represented by a small number of points. For example, there are many cases in the data when there is a pedestrian in the camera images, however, only his head is visible in LiDAR data and thus the relative point density might have a value that slightly exceeds the threshold of 0.0007). At the same time, such a threshold helps to prevent the allocation of the cluster of the small object, like a pedestrian or tree, to the label of the building when this small object is located in front of the building and the center of its point cloud projection is the closest to the center of the bounding box of the building.

3. Allocate the "closest" cluster based on the Euclidean distance from the bounding box center to the cluster center based in 2D projection.
   - Cluster center is calculated according to the following formula:

$$\left(\frac{x\_min + (x\_max\breve{}x\_min)}{2}; \frac{y\_min + (y\_max\breve{}y\_min)}{2}\right) \tag{3.20}$$

   where x_min, x_max – minimum and maximum coordinates of cluster point in 2D projection along X-axis, y_min, y_max – minimum and maximum coordinates of cluster point in 2D projection along Y-axis.

4. Adjust bounding box size and location according to the size and location of the selected cluster.
5. Extend bounding box size by a fraction (extension by 10%).

   Note: Small extension by 10% is made in order to more accurately capture the entire object in case of inaccuracies during clustering.
6. Exclude allocated cluster from further consideration for remaining labels.

3.4 LiDAR 2D depth maps extraction

A depth map is an image representation that contains information regarding the distance to the surfaces of objects around. Depth map representations reduce the dimensionality of the LiDAR data in order to generate 2D images. 2D depth map generated from 3D point cloud contains the same information, but at the same time ensures the low memory consumption and latency of the model during processing in comparison to other approaches, for example, voxel-based approaches that require computationally heavy 3D convolutions [34]. In current work, accurate and precise projection of the LiDAR depth map onto the image is necessary in order to be able to use the labels obtained via camera based object detector.

For this purpose Single Camera and LiDAR-Camera calibration was carried out (See section 3.1, section 4.1). Extrinsic parameters between LiDAR and the camera allow to make transformation from LiDAR 3D coordinates to Camera 3D coordinates, while intrinsic parameters, obtained on a single camera calibration step allows to perform the subsequent projection on the image plane. 2D coordinates on the image of the 3D point are obtained based on the following formula:

$$[u_i, v_i] = M_{int} * M_{ext} * [x_i, y_i, z_i]^T \tag{3.21}$$

where $[u_i, v_i]$ - image coordinates alone X and Y axes, $[x_i, y_i, z_i]^T$ - 3D coordinates of the point of point cloud, i = [1,...,N], where N is a number of points in Point cloud, M_{int} - intrinsic parameters of camera, M_{ext} - extrinsic parameters of camera and LiDAR. Camera images obtained during data acquisition (See section 2.4) have a size [848,480]. Therefore points, that have coordinates outside the range [0:847;0:479] are eliminated from further consideration.

For each point depth information is extracted. Since the potential application of current work - object detection for obstacle avoidance and path planning in drones, Infineon was interested in the calculation of the distance to the objects alone X-Y plane. Therefore, the following formula was used for depth information extraction:

$$d_i = \sqrt{x_i^2 + y_i^2} \tag{3.22}$$

where x_i, y_i - coordinates of the point in 3D LiDAR space along X and Y axes. The resulting depth map is stored as an RGB image in ".png" format, where depth information is encoded as a color, as well as a NumPy array.

Several examples of depth maps and corresponding projections on the images are demonstrated in the Figure 3.20 below.

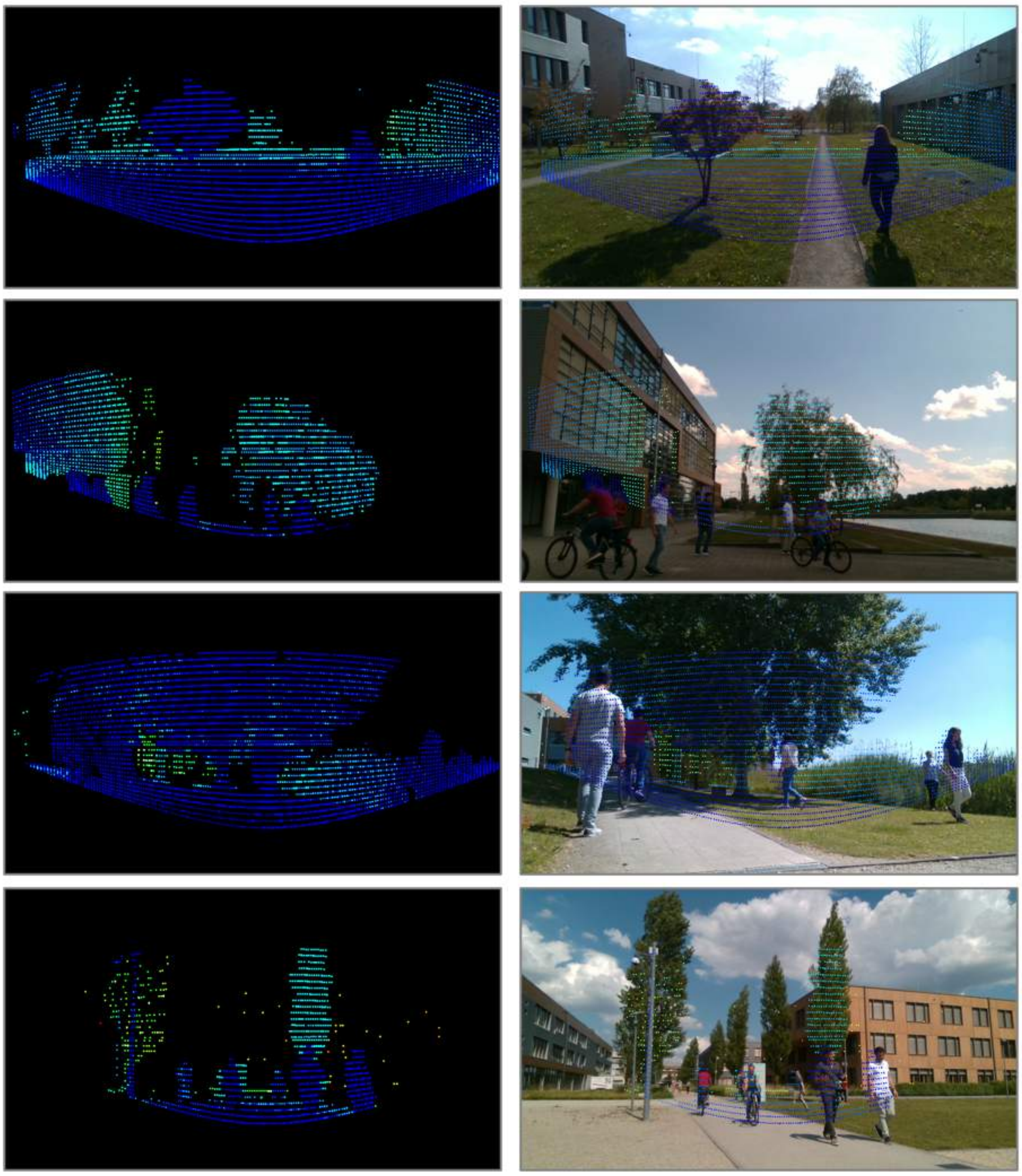


Figure 3.20: Examples of the 2D depth maps obtained based on 3D point cloud

3.5 Fusion of camera and LiDAR labels

In current work, the late fusion strategy is utilized. Late fusion is the most popular sensor fusion strategy since it requires the least amount of computational resources, and is relatively simple. Late fusion strategy allows to combine the results of completely different object detection models for each of the modalities. The late fusion strategy in current work has been chosen based on the requirement of Infineon Technology AG to develop object detection models that could operate based data from on each modality independently. A high-level scheme of the fusion approach that is utilized in current work is demonstrated in the Figure 1.3. Data of each modality is processed separately by object detection models that are trained on 2D camera data and on 2D depth maps. The resulting labels are fused at the last step of the object detection pipeline.

The fusion pipeline is demonstrated in the Figure 3.21 below.

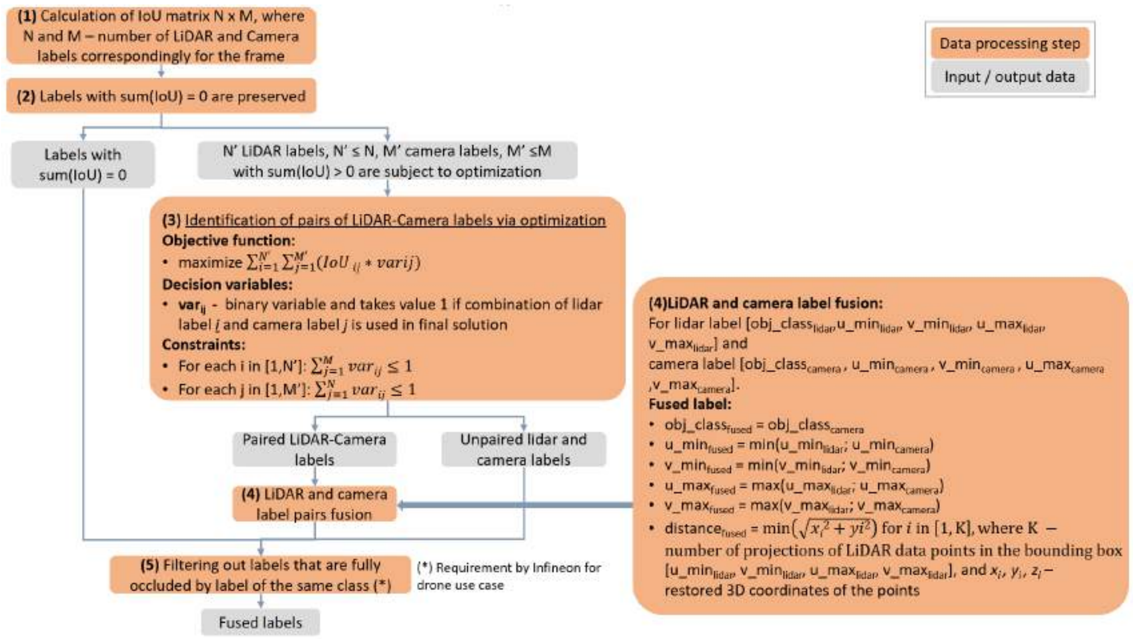

(1) Calculation of IoU matrix N x M, where N and M – number of LiDAR and Camera labels correspondingly for the frame
(2) Labels with sum(IoU) = 0 are preserved
Labels with sum(IoU) = 0
N' LiDAR labels, N' ≤ N, M' camera labels, M' ≤M with sum(IoU) > 0 are subject to optimization
(3) Identification of pairs of LiDAR-Camera labels via optimization
Objective function:
• maximize $\sum_{i=1}^{N'} \sum_{j=1}^{M'} (IoU_{ij} * varij)$
Decision variables:
• var_{ij} - binary variable and takes value 1 if combination of lidar label i and camera label j is used in final solution
Constraints:
• For each i in [1,N']: $\sum_{j=1}^{M} var_{ij} \leq 1$
• For each j in [1,M']: $\sum_{j=1}^{N} var_{ij} \leq 1$
Paired LiDAR-Camera labels
Unpaired lidar and camera labels
(4) LiDAR and camera label pairs fusion
(5) Filtering out labels that are fully occluded by label of the same class (*)
(*) Requirement by Infineon for drone use case
Fused labels
Data processing step
Input / output data
(4)LiDAR and camera label fusion:
For lidar label [obj_class_{lidar}, u_min_{lidar}, v_min_{lidar}, u_max_{lidar}, v_max_{lidar}] and
camera label [obj_class_{camera}, u_min_{camera}, v_min_{camera}, u_max_{camera}, v_max_{camera}].
Fused label:
• $obj\_class_{fused} = obj\_class_{camera}$
• $u\_min_{fused} = \min(u\_min_{lidar}; u\_min_{camera})$
• $v\_min_{fused} = \min(v\_min_{lidar}; v\_min_{camera})$
• $u\_max_{fused} = \max(u\_max_{lidar}; u\_max_{camera})$
• $v\_max_{fused} = \max(v\_max_{lidar}; v\_max_{camera})$
• $distance_{fused} = \min\left(\sqrt{x_i^2 + yi^2}\right)$ for i in $[1, K]$, where K – number of projections of LiDAR data points in the bounding box [u_min_{lidar}, v_min_{lidar}, u_max_{lidar}, v_max_{lidar}], and x_i, y_i, z_i – restored 3D coordinates of the points

Figure 3.21: Fusion pipeline

Fusion of resulting labels from the camera-based and LiDAR-based object detection model is performed based on IoU (Intersection over Union, See subsection 3.2.2 for detailed explanation) of the camera and LiDAR labels with the assumption that "best" combination of camera and LiDAR labels will have the highest total IoU. At the **Step (1)** IoU between all camera and LiDAR labels is calculated:

$$\begin{array}{cccccc} & camera\_label_1 & .. & camera\_label_j & .. & camera\_label_M \\ lidar\_label_1 & IoU_{11} & .. & IoU_{1j} & .. & IoU_{1M} \\ .. & .. & .. & .. & .. & .. \\ lidar\_label_i & IoU_{i1} & .. & IoU_{ij} & .. & IoU_{iM} \\ .. & .. & .. & .. & .. & .. \\ lidar\_label_N & IoU_{N1} & .. & IoU_{Nj} & .. & IoU_{NM} \end{array} \tag{3.23}$$

for $i \in [1, N]$, for $j \in [1, M]$, where N - number of LiDAR labels, M - number of camera labels.

At the **Step (2)** sum of IoU for each label is analyzed: labels from each modality that do not have an intersection with any of labels from another modality (sum of IoU = 0) are taken in the final label list without any changes:

- $lidar\_label_i : \sum_{j=1}^{M} IoU_{i,j} = 0$
- $camera\_label_j : \sum_{i=1}^{N} IoU_{i,j} = 0$

LiDAR labels are augmented with a minimum distance that is calculated based on recovered 3D coordinates of the point that have projections located inside the bounding box. The minimum distance is calculated along the X-Y plane according to the following formula:

$$dist_{min} = \sqrt{x_i^2 + y_i^2} \tag{3.24}$$

, where $i \in [1, K]$, K - total number of point projections inside bounding box, x_i, y_i - restored coordinates in 3D space along X and Y axes.

In minimum distance calculation, occlusion is not taken into account. For camera labels that do not have IoU with any LiDAR labels, the minimum distance is set to 0.

LiDAR labels [1,N′], $N' \leq N$ and camera labels [1,M′], $M' \leq M$ are subject to search for "best" combination of pairs of LiDAR-camera labels.

At the **Step (3)** identification of pairs of LiDAR-Camera labels via optimization routine. Objective function:

$$f = maximize(\sum_{i=1}^{N'} \sum_{j=1}^{M'} IoU_{i,j} * var_{i,j}) \tag{3.25}$$

where $var_{i,j}$ - binary variable and takes value 1 if a combination of lidar label i and camera label j is used in the final solution.

Decision variable is $var_{i,j}$ for any combination of i and j.

Since each label from each modality could be used only once to make a pair with a label from the other modality, the constraints for the optimization function are:

$$\forall i \in [1, N'] : \sum_{j=1}^{M'} IoU_{i,j} \leq 1 \tag{3.26}$$

$$\forall j \in [1, M'] : \sum_{i=1}^{N'} IoU_{i,j} \leq 1 \tag{3.27}$$

An optimization procedure is implemented via open source PuLP library [2] (Python). PuLP utilizes the simplex method combined with other algorithms (ex.: branch-and-bound and cut-generation). The Simplex Method is the earliest solution algorithm for solving linear programming problems by hand. It is an efficient strategy for solving a series of systems of linear equations and takes advantage of the geometry of the problem: it visits the vertices of a feasible set and checks each visited vertex for optimality. By using a greedy strategy while jumping from a feasible vertex to the next adjacent vertex, the algorithm terminates at an optimal solution.

As a result, pairs of LiDAR-camera labels with the maximum total IoU are found. Labels for which a pair was not found during the optimization process are stored in the resulting list of fused labels without any changes. For such LiDAR labels, the minimum distance is calculated according to the formula above 3.24 in Step 2. For camera labels that do not have IoU with any LiDAR labels, the minimum distance is set to 0.

At the **Step (4)** fusion of pairs of LiDAR and camera labels is performed. Fusion strategy is described at the figure Figure 3.21: For lidar label [obj_class_{lidar}, u_min_{lidar}, v_min_{lidar}, u_max_{lidar}, v_max_{lidar}] and camera label [obj_class_{camera}, u_min_{camera}, v_min_{camera}, u_max_{camera} ,v_max_{camera}], fused label is formed in the following way:

- $obj\_class_{fused} = obj\_class_{camera}$
- $u\_min_{fused} = min(u\_min_{lidar}; u\_min_{camera})$,
- $u\_min_{fused} = min(u\_min_{lidar}; u\_min_{camera})$,
- $v\_min_{fused} = min(v\_min_{lidar}; v\_min_{camera})$,
- $u\_max_{fused} = max(u\_max_{lidar}; u\_max_{camera})$,
- $v\_max_{fused} = max(v\_max_{lidar}; v\_max_{camera})$.

Each of the resulting fused labels is augmented with the minimum distance calculated according to the formula above 3.24.

The last **Step 5** of the label fusion algorithm fully occluded labels are eliminated if they are fully occluded by the label with the same object class. This step is provided according to the requirement by Infineon for drone use case when the detection of nearby objects is of the most importance in order to avoid obstacles. Detection of fully occluded objects is not of high importance for this use case.

On the figure Figure 3.22 below an example of the labels fusion results is demonstrated:

- (a) contains labels from the camera-based object detection model,

[2]https://coin-or.github.io/pulp/

- (b) contains labels from the LiDAR depth map-based object detection model,
- (c) contains resulting labels after fusion.

Figure (a) shows the accurate detection of object classes in a camera-based object detection model, while in picture (b) LiDAR incorrectly identified classes for two objects - a pedestrian and a cyclist. Picture (c) contains accurate object classes and bounding box detection after the fusion step. For example, the bounding box of the tree in the center of the picture has changed based on information from both the camera and LiDAR and is more accurate in comparison to (a) and (b).

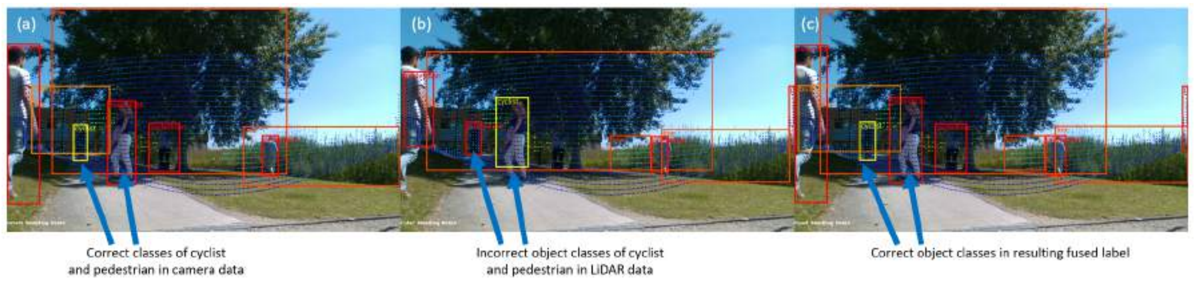

(a)
(b)
(c)
Correct classes of cyclist and pedestrian in camera data
Incorrect object classes of cyclist and pedestrian in LiDAR data
Correct object classes in resulting fused label

Figure 3.22: Fused object class identification

4 Experiments and Evaluation

4.1 Camera-LiDAR Calibration

4.1.1 Single camera calibration

3 datasets for single camera calibration were recorded (each dataset contained 100-200 frames) and more than 12 single camera calibration sessions were conducted with use of Matlab Single Camera Calibrator App. A unique set of 10 to 40 images was used as an input in each session.

The estimation of single camera calibration is performed based on reprojection error. A reprojection error is a distance between a pattern keypoint detected in a calibration image, and a corresponding world point projected into the same image[1]. The mean reprojection error is estimated for each image in the calibration session and then the overall mean is calculated. Therefore, the results obtained in different calibration sessions are not directly comparable, since a different set of images is used as an input for each session. Consequently, only the results of the session that were used in this work are shown. The results of the other sessions are not demonstrated.

The calibration session, the results of which were used further in the current work, was carried out on the basis of 32 images. Mean reprojection error is estimated for each of 32 images, an overall mean error value is equal to 0.13 pixel (See Figure 4.1).

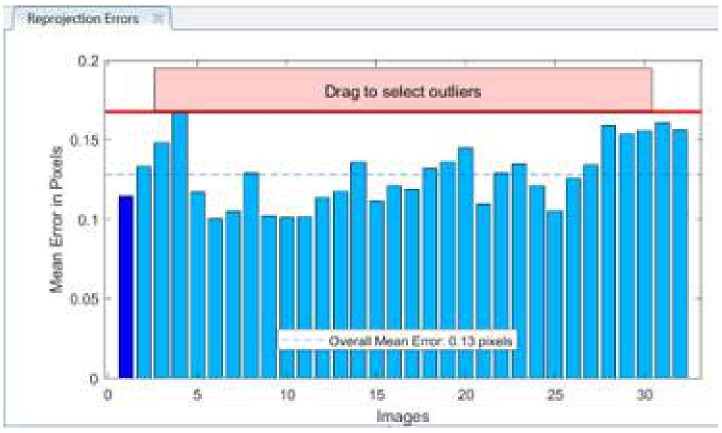


Figure 4.1: Single camera calibration error estimation

[1]'https://de.mathworks.com/help/vision/ug/evaluating-the-accuracy-of-single-camera-calibration.htmlEvaluatingCameraCalibrationExample-4'

Resulting intrinsic matrix, obtained after camera calibration:

$$M_{int} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ u_0 & v_0 & 1 \end{bmatrix} = \begin{bmatrix} 570.7652146 & 0 & 0 \\ 0 & 572.5377849 & 0 \\ 420.2553067 & 259.6667927 & 1 \end{bmatrix} \tag{4.1}$$

where f_x, f_y - horizontal and vertical focal lengths respectively expressed in pixel units, u_0, v_0 - optical center (principal point) of the image.

4.1.2 Camera-LiDAR calibration

37 datasets, 200-500 frames each, were recorded for the purpose of LiDAR-Camera calibration. More than 30 calibration sessions were conducted via the Matlab LiDAR-Camera calibrator App in order to find satisfying extrinsic parameters between LiDAR and the Camera. Similar to single camera calibration, each session was run on a unique set of images. For each session, a different number of images was used - from 20 to 120.

Since for each session the errors are calculated for a specific set of images, the results of the sessions are not directly comparable. Therefore, in addition to the error calculation, a visual analysis of the resulting projection was carried out (described in the current section below). Similar to the previous section on Single Camera Calibration, only the results of the session, which were used in further experiments, are described.

Camera-LiDAR calibration is evaluated based on 3 metrics that are calculated for each image (Description of the metrics is provided in the official documentation of Matlab LiDAR-Camera calibrator App[2]):

- **Translation Error** - The difference between the centroid coordinates of the checkerboard planes in the point clouds and those in the corresponding images, in meters.
- **Rotation Error** -The difference between the normal angles defined by the checkerboard planes in the point clouds and those in the corresponding images, in radians.
- **Reprojection Error** - The difference between the projected (transformed) centroid coordinates of the checkerboard planes from the point clouds and those in the corresponding images, in pixels.

The values of translation, rotation and reprojection error obtained after calibration are shown in the Figure 4.2 below.

[2]'https://de.mathworks.com/help/lidar/ref/estimatelidarcameratransform.htmlmw$_1aea5c0d - 1756 - 480e - ba44 - a55f5a3ee4a5'$

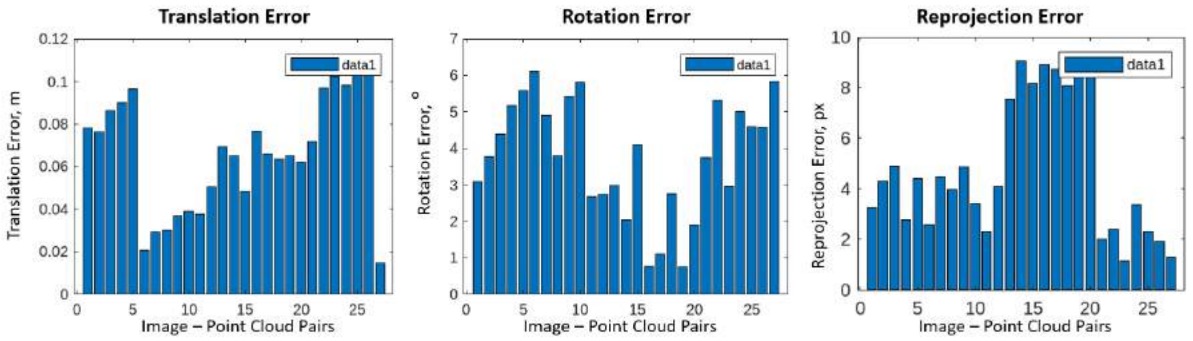


Figure 4.2: Camera-LiDAR calibration error estimation

The calibration was performed based on 27 images. The value of the translation error varies from 0.0146m to 0.1062m with a mean of 0.066m and a standard deviation of 0.027m. The values of rotation error vary from 0.75° to 6.11° with a mean 3.77° and standard deviation is 1.59°. Reprojection error is estimated in pixels and does not exceed 9.2px, while the minimum value is equal to 1.17px, the mean is equal to 4.77px, and the standard deviation is 2.74px.

Resulting extrinsic matrix that was obtained after calibration:

$$M_{ext} = [R|T] = \begin{bmatrix} 0.999686098 & -0.005442685 & -0.024455747 & -0.121739535 \\ 0.02440216 & -0.009703409 & 0.99965513 & 0.167662746 \\ -0.005678113 & -0.999938109 & -0.009567549 & 0.056827091 \end{bmatrix} \tag{4.2}$$

The results of the calibration were assessed visually based on the projection of the checkerboard. The checkerboard has been placed in 6 positions relative to the sensors (close to the sensor, distant from the sensor; on the left side, center, and on the right side of the scene). The example Figure 4.4 shows that the point cloud of the checkerboard and person in the center position is projected onto the image the most accurately, while the projections of the checkerboard and person in the left and right positions are not perfectly aligned with the corresponding objects in the image.

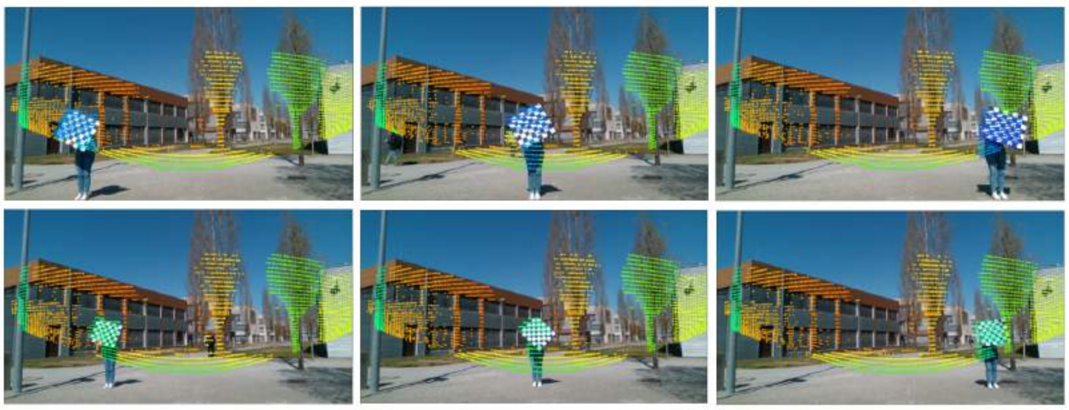


Figure 4.3: Camera-LiDAR calibration results demonstration

The Figure 4.4 below demonstrates the improvements of the projection of the point cloud after calibration in comparison to the projection obtained based on spherical coordinates. The scene contains only static objects (buildings, fences, trees). The image on the right side shows that the projection obtained based on calibration is accurate, and the contours of objects, such as a tree or the windows of a building, match well. Therefore, the obtained results are suitable for further sensor fusion experiments.

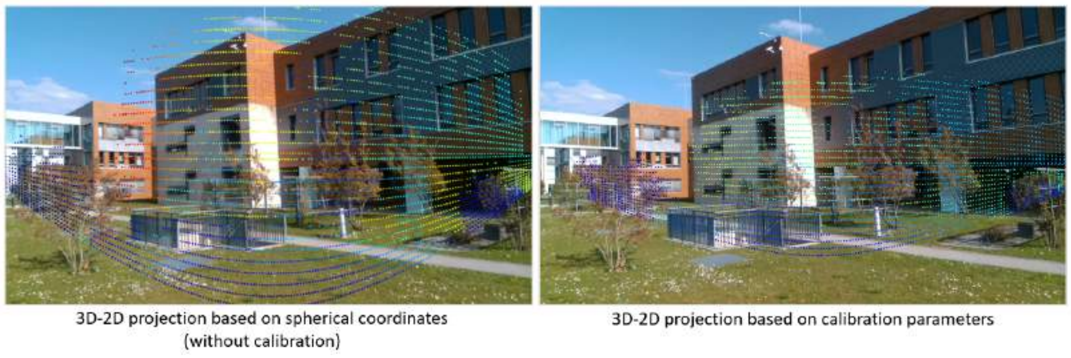

3D-2D projection based on spherical coordinates
(without calibration)
3D-2D projection based on calibration parameters

Figure 4.4: Example of the 3D-2D projection of point cloud without and with calibration

4.2 Camera based object detection

4.2.1 Camera based object detection experiments

Environment

Training of object detection models was conducted at Infineon GPU farm equipped with NVIDIA TESLA P40 GPU with the following specification:

Parameter	Value
MEMORY SIZE (PER BOARD)	24 GB GDDR5
CUDA CORES	3840
Computing power	12 TFLOPS
INTEGER OPERATIONS (INT8)	47 TOPS (Tera-Operations per Second, boost clocks)

Table 4.1: GPU specification

All implementation including all data preprocessing, label filtering, and sensor fusion steps, is completed in Python language (3.9). Official Pytorch implementation of YOLOv5 by Ultralytics[3] was used.

[3]https://github.com/ultralytics/yolov5

Data preparation for training

In order to conduct object detection model training using the transfer learning approach and prepare the model that will be able to detect the custom set of object classes (pedestrian, cyclist, building, tree, and fence), a part of Infineon unlabeled dataset (See section 2.4) was selected and annotated.

Since there are only 5 object classes and this number does not cover the whole variety of objects that can be found in the environment, some assumptions were made: for example, tall grass was annotated as a tree, a cyclist who walks next to a bike has been annotated as a cyclist.

The data frames were selected with steps equal to 10 from a dynamically recorded dataset that contains 2340 frames in total, and with a step equal to 20 - from a statically recorded dataset that contains 8000 in total. As a result, 234 frames were selected from a dynamically recorded dataset and 400 were selected from a statically recorded dataset, 634 frames in total. This sample of data was annotated via labelImg framework[4].

Annotated dataset of 634 images was divided into train-test-validation parts in proportion 60%, 20%, and 20% correspondingly using a Python script. Thus, 380 images were used for training, 127 images were used for test, and 127 - for validation. The structure of the dataset folder is shown in the Figure 4.5.

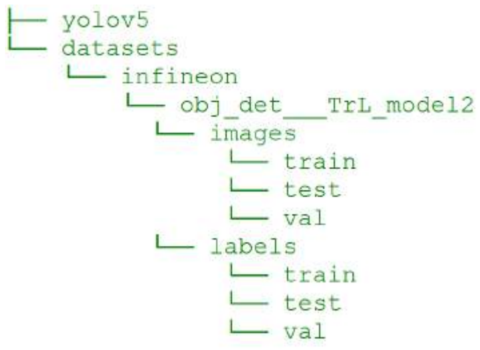
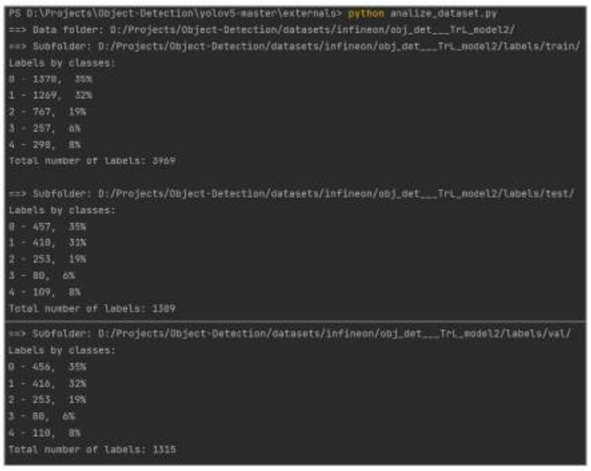

```
├── yolov5
└── datasets
    └── infineon
        └── obj_det___TrL_model2
          └── images
                └── train
                └── test
                └── val
          └── labels
                └── train
                └── test
                └── val
```

Figure 4.5: Structure of the dataset used for camera based object detection model training

[4]https://github.com/heartexlabs/labelImg



```
PS D:\Projects\Object-Detection\yolov5-master\externals> python analize_dataset.py
==> Data folder: D:/Projects/Object-Detection/datasets/infineon/obj_det___TrL_model2/
==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___TrL_model2/labels/train/
Labels by classes:
0 - 1378,  35%
1 - 1269,  32%
2 - 767,  19%
3 - 257,  6%
4 - 298,  8%
Total number of labels: 3969

==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___TrL_model2/labels/test/
Labels by classes:
0 - 457,  35%
1 - 410,  31%
2 - 253,  19%
3 - 80,  6%
4 - 109,  8%
Total number of labels: 1309
==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___TrL_model2/labels/val/
Labels by classes:
0 - 456,  35%
1 - 416,  32%
2 - 253,  19%
3 - 80,  6%
4 - 110,  8%
Total number of labels: 1315
```

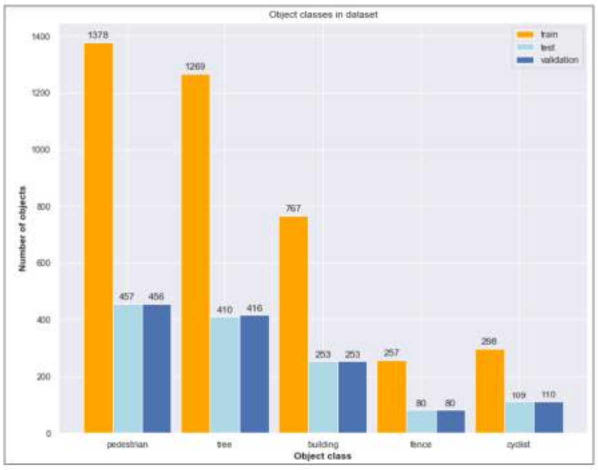


Figure 4.6: Distribution of the object classes in train, test and validation part of the dataset

On the Figure 4.6 above the distribution of the object classes is shown. Since object classes 3 - "fence" and 4 - "cyclist" are underrepresented, the dataset is imbalanced.

Labels are stored in YOLO format. For each image, a label file with the following name "<image_filename>.txt" is stored. Each row of the label file encodes one object and contains a string of 5 values in the following order [object_class x_center y_center width height]. Object_class is encoded by integer number i, $i = 0, ..., N - 1$, where N - total number of object classes. In the current dataset the following encoding is applied:

- 0 - pedestrian,
- 1 - tree,
- 2 - building,
- 3 - fence,
- 4 - cyclist.

The image size alone X and Y axes is considered equal to 1 and x_center, y_center, width and height of the object are calculated relative to the image size. Example of the label in YOLO format: [2 0.290 0.531 0.256 0.391].

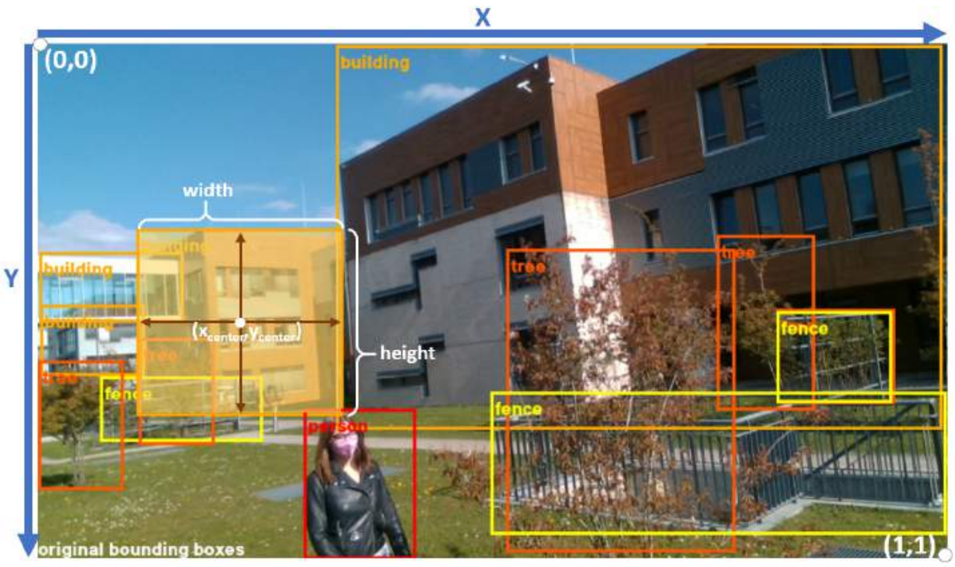


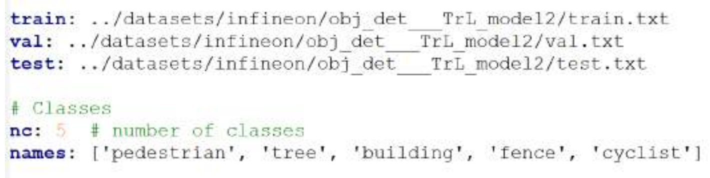
Figure 4.7: Data label in YOLO format

Configuration files

The configurations for the YOLOv5 training are divided into three files with ".yaml" extension:

- data-configurations file (data.yaml)
- model-configurations file (model.yaml)
- hyperparameters-configurations file (hyp.yaml)

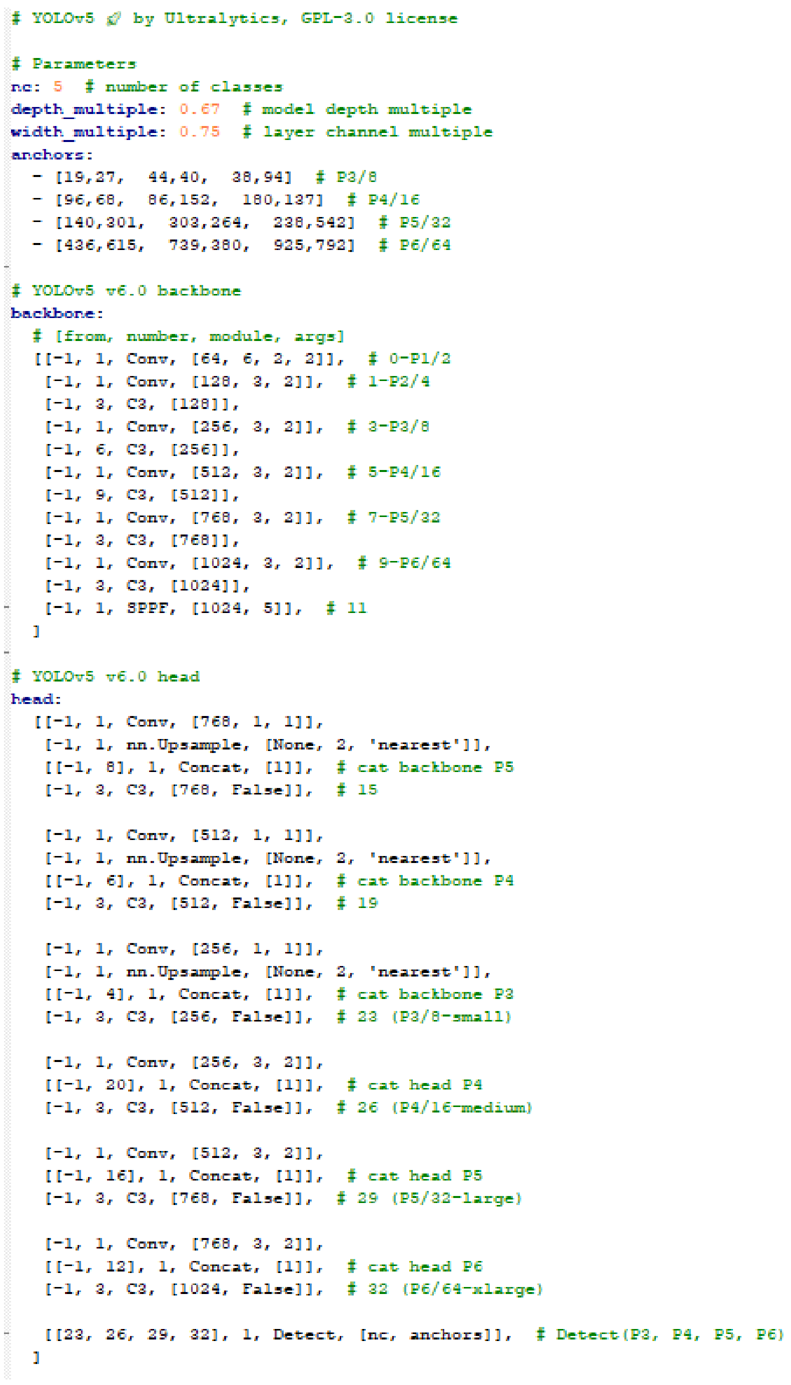
Data-configurations file data.yaml is used by the model in order to access the images during training. Data.yaml contains a summary about the dataset, including paths to train, test, and validation data, number of classes as well as class names (See Figure 4.8 below).



```
train: ../datasets/infineon/obj_det___TrL_model2/train.txt
val: ../datasets/infineon/obj_det___TrL_model2/val.txt
test: ../datasets/infineon/obj_det___TrL_model2/test.txt

# Classes
nc: 5  # number of classes
names: ['pedestrian', 'tree', 'building', 'fence', 'cyclist']
```

Figure 4.8: Data-configurations file for YOLOv5 object detection model training

In the model-configurations file, the architecture of the model is defined. Ultralytics provides the model-configurations for each of models discussed in subsection 3.2.3. During the training, the model-configurations file is read and the model is built based on the architecture specified. This flexible approach provides opportunities to experiment with the architecture and adapt it to specific needs. Model-configurations file for YOLOv5m6 object detection is shown at the Figure 4.9. Default YOLOv5m6 architecture was used during training, and the number of object classes was set to 5. The anchor box parameters can be ignored since auto-learning of anchor box sizes is integrated into the object detection model training pipeline. Parameters "depth_multiple" and "width_multiple" are multipliers that are applied to the number of layers and filters of the network correspondingly in order to balance network depth and width. Higher multiplier values result in higher network parameters number size and accuracy. However, a higher number of model parameters requires more computational resources and training time while the inference speed decreases.

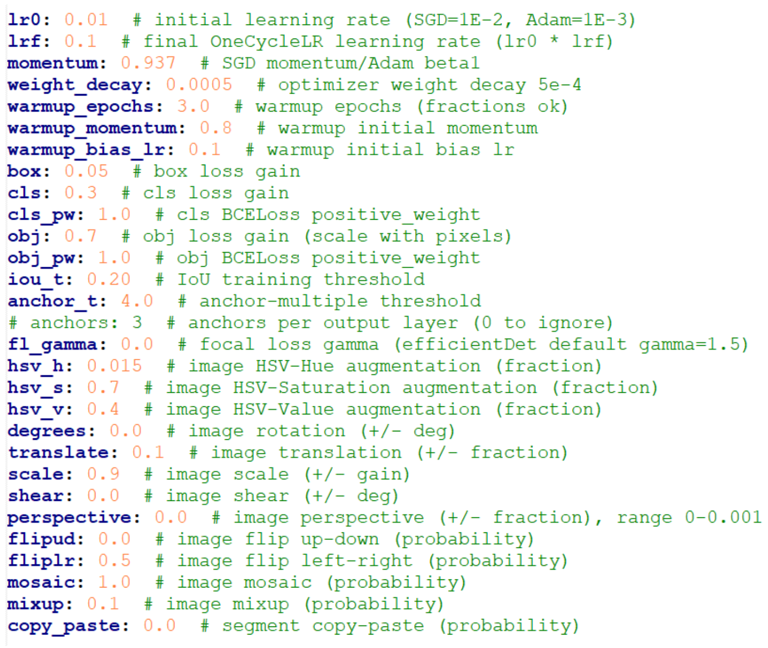


```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 5  # number of classes
depth_multiple: 0.67  # model depth multiple
width_multiple: 0.75  # layer channel multiple
anchors:
  - [19,27,  44,40,  38,94]  # P3/8
  - [96,68,  86,152,  180,137]  # P4/16
  - [140,301,  303,264,  238,542]  # P5/32
  - [436,615,  739,380,  925,792]  # P6/64

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [768, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [768]],
   [-1, 1, Conv, [1024, 3, 2]],  # 9-P6/64
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 11
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [768, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 8], 1, Concat, [1]],  # cat backbone P5
   [-1, 3, C3, [768, False]],  # 15

   [-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 19

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 23 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 20], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 26 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 16], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [768, False]],  # 29 (P5/32-large)

   [-1, 1, Conv, [768, 3, 2]],
   [[-1, 12], 1, Concat, [1]],  # cat head P6
   [-1, 3, C3, [1024, False]],  # 32 (P6/64-xlarge)

   [[23, 26, 29, 32], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5, P6)
  ]
```

Figure 4.9: Model-configurations file for YOLOv5m6 object detection model training

Hyperparameters for the training, including the data augmentation, learning rate, etc. are defined in the hyperparameters-configurations file that includes 28 hyperparameters. Ultralytics provides 3 versions of the hyperparameters-configurations file: hyp.scratch-low.yaml, hyp.scratch-medium.yaml, and hyp.scratch-high.yaml.

In order to set a model baseline, training was performed with hyp.scratch-medium.yaml. Values for each hyperparameter are shown in the Figure 4.10 below.



```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.1  # image mixup (probability)
copy_paste: 0.0  # segment copy-paste (probability)
```

Figure 4.10: Hyperparameters-configurations file for YOLOv5m6 object detection model training with medium data augmentation

Training the object detection model

Model training and convergence are influenced by hyperparameter values, including the following:

- Number of Epochs - the number of times to iterate over the dataset,
- Batch Size - the number of data samples propagated through the network before the parameters are updated,
- Learning Rate - the hyperparameter that determines the step size at each iteration (batch/epoch). Smaller values yield slower learning speed, while larger values may result in unpredictable behavior during training[5].

Initial weights for training were obtained from the checkpoint of the model that was pretrained on MS COCO dataset during 300 epochs. Training using transfer learning approach was conducted with freezing weights of 12 layers of the backbone network:

```
python train.py --batch 16 --epochs 200 --data data/infineon_OD_14-08.yaml
--weights 'yolov5m6.pt' --img 1280
```

[5]'https://pytorch.org/tutorials/beginner/basics/optimization$_t$*utorial.html*'

```
--cfg models/infineon-yolov5m6_OD_14-08.yaml
--hyp data/hyps/hyp.scratch-med.yaml --cache --freeze 12
--project 'runs-inf-08-14' --name 'feature_extraction'
```

A training of several models with batch_size = 8, 16, 32 that result in 47, 23, and 11 gradient updates in one epoch correspondingly, and the number of epochs = 150, 200, 300 was conducted in order to find values of batch_size and number of epochs that provide the best training results. All experiments were conducted with Stochastic Gradient Decent (SGD) optimizer with Nesterov momentum, initial learning rate equal to 0.01, and SGD momentum equal to 0.937. The choice of the optimizer has been made based on the comparison of optimizers (SGD, ADAM, ADAMW) that was conducted by authors of YOLOv5. YOLOv5 has been trained on the VOC dataset over 50 epochs with batch sizes equal to 16 and 64. Authors showed that models which have been trained with SGD optimizer converge faster and reach higher values of mAP after training[6]. A comparison of model performance and losses is provided in the graphs below:

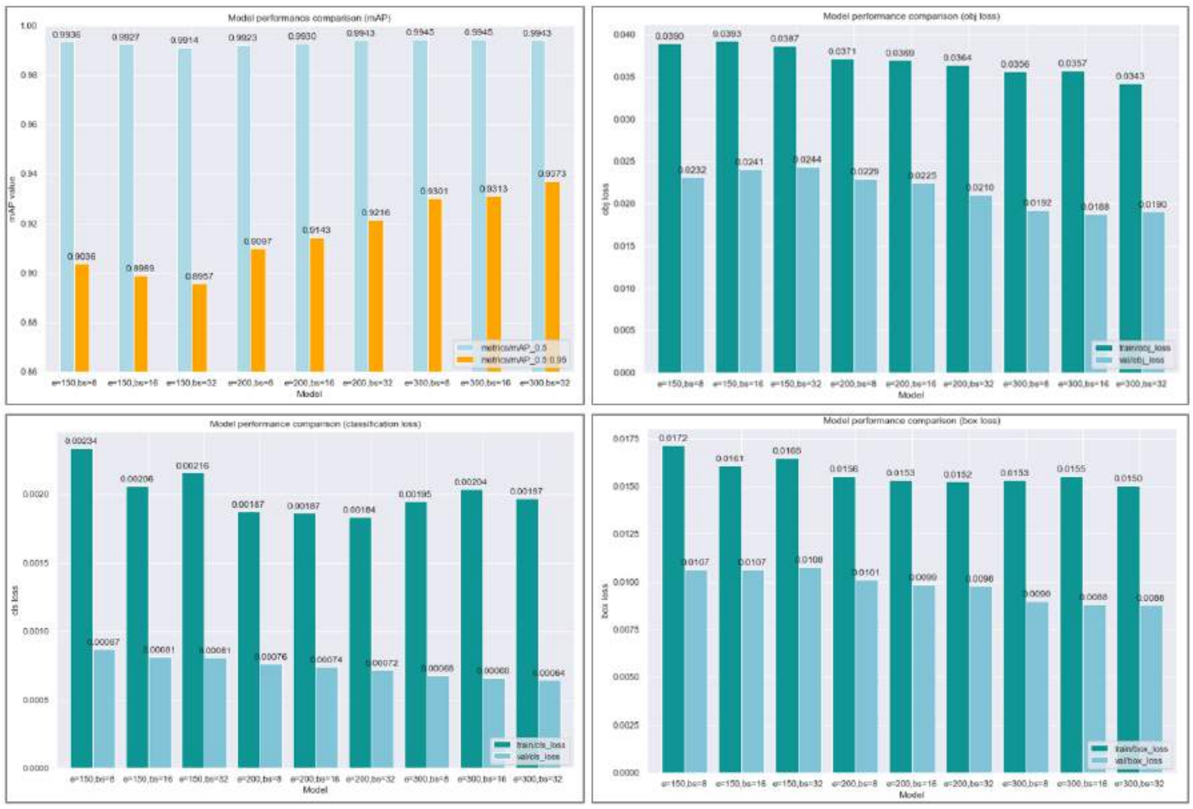
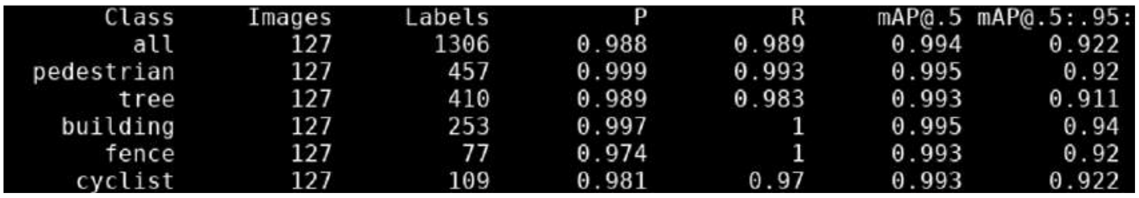


Figure 4.11: Comparison of the object detection YOLOv5m6 model performance trained with batch_size = 8, 16, 32 over epochs = 150, 200, 300.

[6]Results of the comparison are provided in https://user-images.githubusercontent.com/26833433/147961825-8a9d81ba-9374-499f-80a7-60925b4c8328.png

All models showed high performance with a slight difference in values of train and validation losses. After experiments the following parameters have been chosen: batch_size=32, a number of epochs = 200, which has a good trade-off between performance and training time.

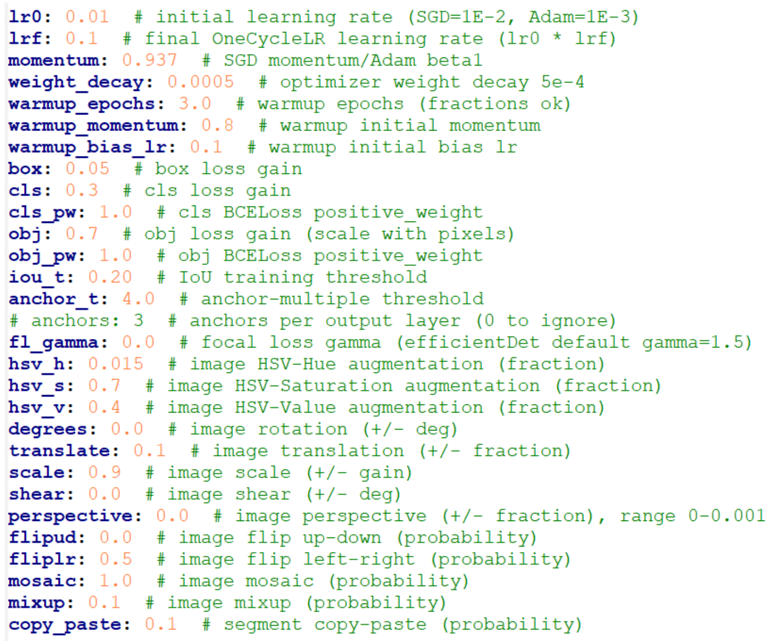
Despite the classes in the dataset are imbalanced and the classes "pedestrian", "tree" and "building" are significantly better represented than the rest, all performance measures achieve the same high values regardless of class (See Figure 4.12).



```
       Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
         all        127       1306      0.988      0.989      0.994      0.922
  pedestrian        127        457      0.999      0.993      0.995       0.92
        tree        127        410      0.989      0.983      0.993      0.911
    building        127        253      0.997          1      0.995       0.94
       fence        127         77      0.974          1      0.993       0.92
     cyclist        127        109      0.981       0.97      0.993      0.922
```

Figure 4.12: Performance measures values by object classes for YOLOv5m6 model trained over 200 epochs with batch_size 3.

Another experiment was performed in order to analyze whether it is possible to improve the model performance using additional data augmentation. Thus the training was conducted based on hyp.scratch-high.yaml hyperparameters-configurations file. In addition to image mix-up augmentation, mosaic, left-right flip, translation, scale, and color adjustment (HSV Hue, Saturation, and Value) that present in hyp.scratch-medium.yaml, segment copy-paste data augmentation technique is introduced in hyp.scratch-high.yaml. Hyperparameters values of the hyperparameters-configurations file (hyp.infineon.scratch-high.yaml) that were used for training, are shown in the figure below:



```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.1  # image mixup (probability)
copy_paste: 0.1  # segment copy-paste (probability)
```

Figure 4.13: Hyperparameters-configurations file for YOLOv5m6 object detection model training with high data augmentation

Comparison of results obtained after training the model with high data augmentation hyperparameters showed no improvement in model performance (See Figure 4.14 below). The model with medium data augmentation has validation box and objectness loss 0.00016 and 0.00121 lower correspondingly than the model with high data augmentation, while performance measures mAP_0.5 and mAP_0.5:0.95 values are higher by 0.00094 and 0.0086 after 200 epochs. Therefore, the model trained with high data augmentation is discarded. The model trained over 200 epochs with medium data augmentation and batch size equal to 32 has been chosen for further experiments.

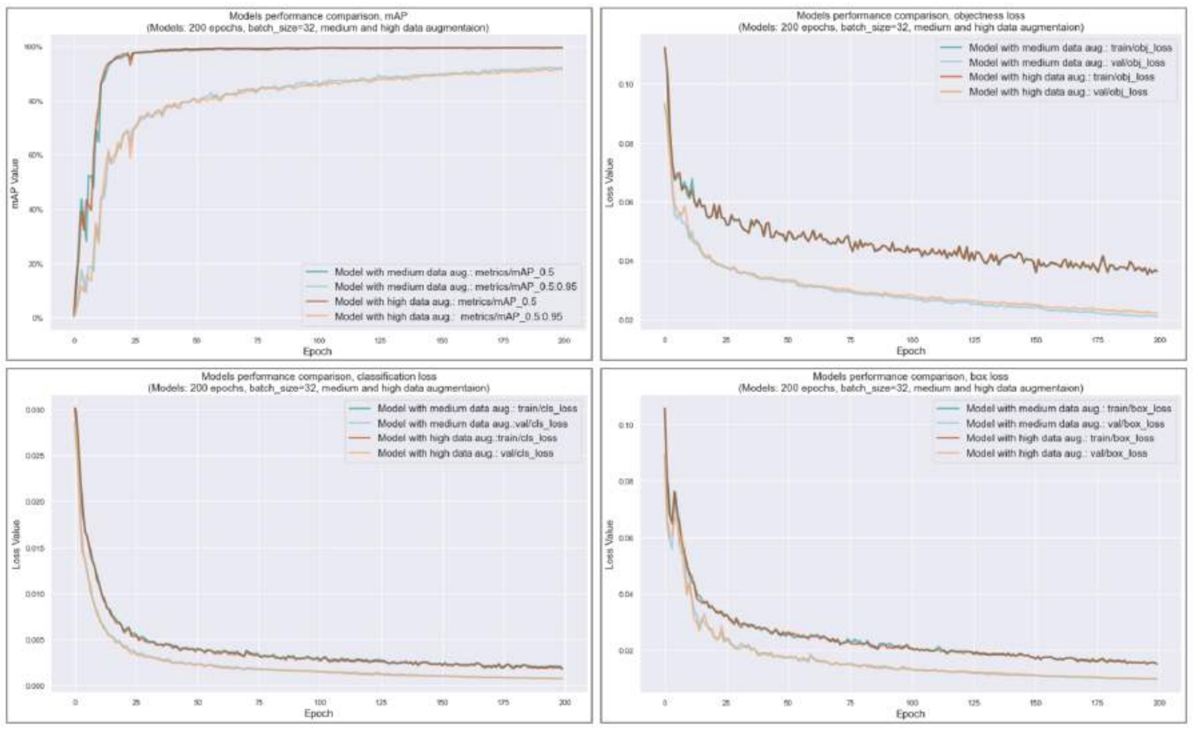


Figure 4.14: Comparison of the object detection YOLOv5m6 model performance with medium and high data augmentation hyperparameters

4.2.2 Object detection model training results

The losses and performance measures values over the entire training process are visualized on the figures below (Figure 4.15, Figure 4.16).

YOLOv5 loss function is composed of three parts:

- box_loss — bounding box regression loss (Mean Squared Error).
- obj_loss — the confidence of object presence is the objectness loss (Binary Cross Entropy with logits).
- cls_loss — the classification loss (Cross-Entropy).

The values of losses and performance metrics mAP@0.5 and mAP@0.5:0.95 that were achieved after training are shown in the table Table 4.2 below:

Parameter	Value
train/box_loss	0.015239
val/box_loss	0.009765
train/obj_loss	0.036410
val/obj_loss	0.021030
train/cls_loss	0.001838
val/cls_loss	0.000717
Precision	0.98803
Recall	0.99616
mAP@0.5	0.99430
mAP@0.5:0.95	0.92158

Table 4.2: Losses and performance measures values achieved at the end of the training of the object detection model over 200 epochs

Figure 4.15 below demonstrates that all three loss values decrease very fast during the first 25 epochs. During epochs 26-200, there is a gradual decrease in losses. Validation losses are consistently lower than training losses during the whole training.

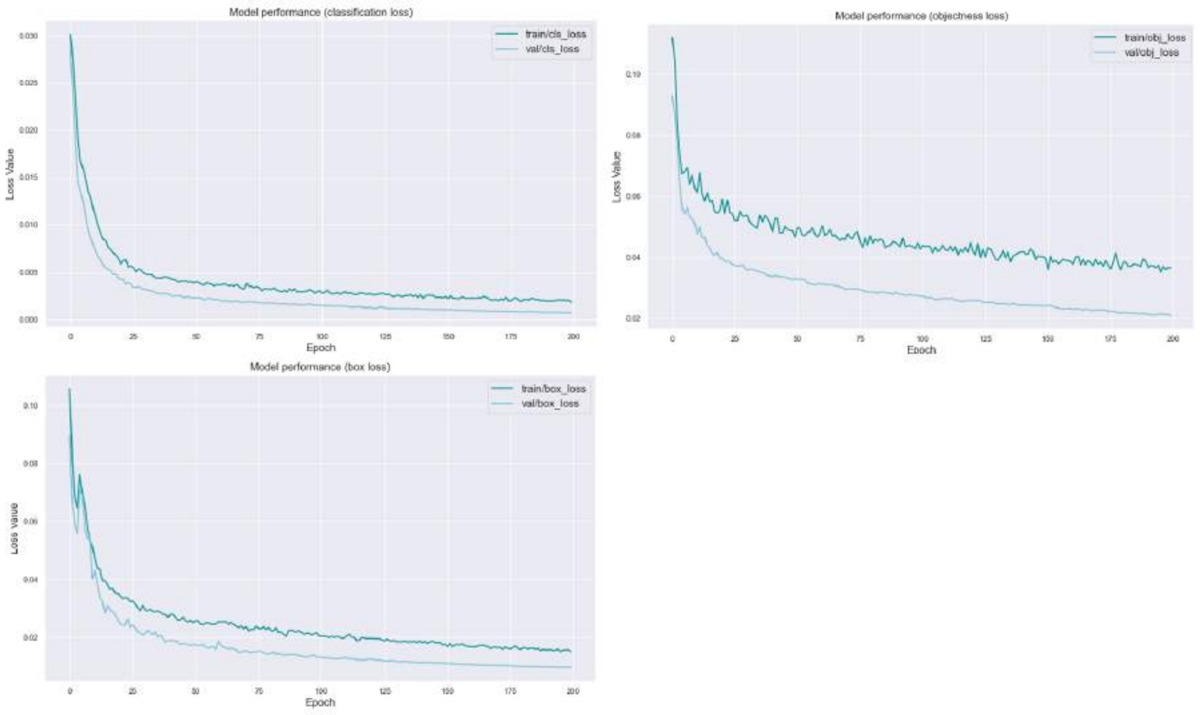


Figure 4.15: Camera based object detection model performance (training and validation losses)

Similar to losses, performance measures of the model increase very fast up to 25_th epoch with a gradual increase till the end of the training. Model reaches the value of 99% mAP@0.5

and 92% mAP@0.5:0.95.

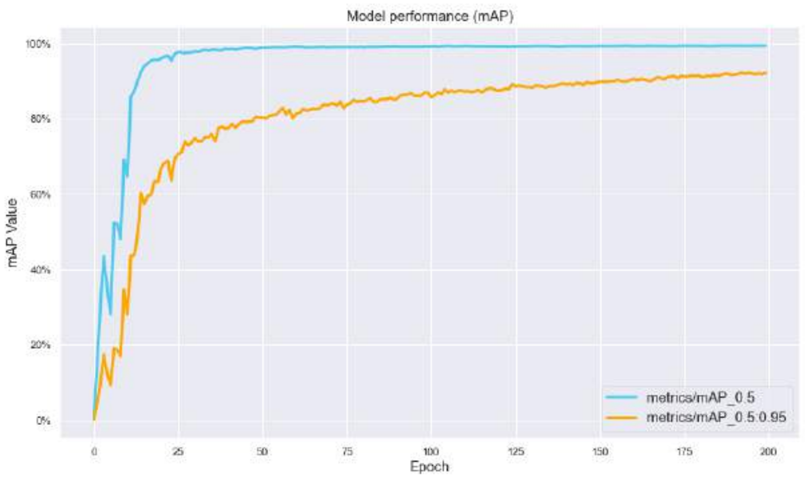


Figure 4.16: Camera based object detection model performance (mAP)

After the training, the best and last weights are saved in files best.pt and last.pt. Both files are identical since the best performance was achieved at the end of the training. The size of each of the files is 70Mb.

4.2.3 Object detection results

In the figures Figure 4.17, Figure 4.18, Figure 4.19, and Figure 4.20 several examples of object detections are shown. All images were recorded in good weather conditions under bright daylight.

Scenes Figure 4.17(a) and Figure 4.17(b) contain building, trees, pedestrian and cyclists. Some of the objects are slightly occluded. Detections of the objects in both images are very accurate.

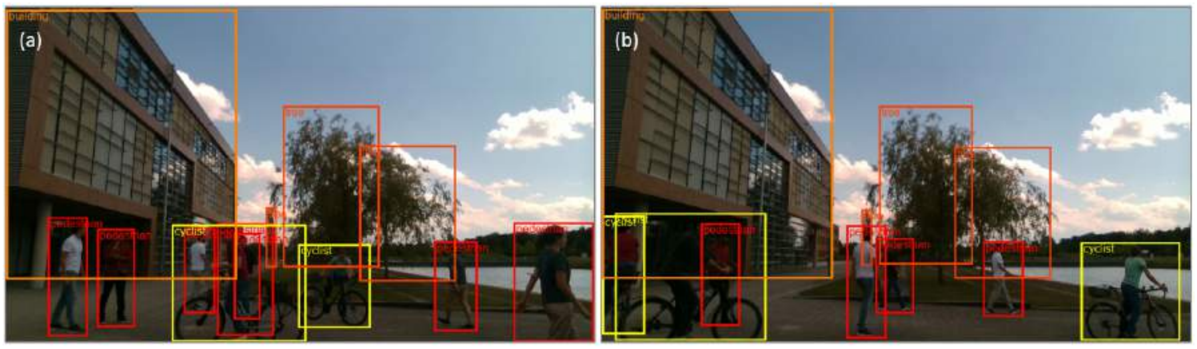


Figure 4.17: Camera based object detection examples (a,b)

Figures Figure 4.18(c) and (d) are a part of the sequence that was recorded on a bridge, along which many people are moving. Therefore these images contain a large amount of highly occluded objects. In image (c) there are two highly occluded pedestrians that were not detected while in image (d) all detections are correct.

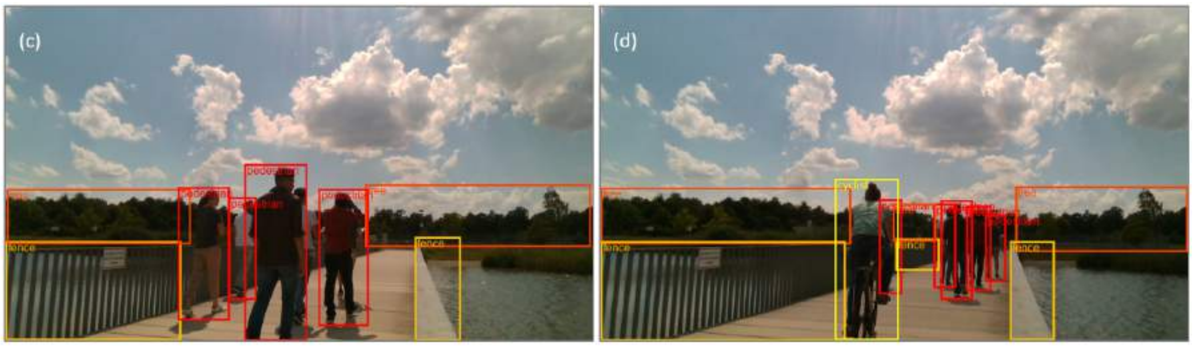


Figure 4.18: Camera based object detection examples (c,d)

Scene Figure 4.19(e) contains only static objects (buildings and trees) that are either close or distant from the camera and thus have different scales on the image. Most of the objects are clearly visible. All detections are very accurate. Scene Figure 4.19(e) contains both static and moving objects. In the middle of the image, there are several distant objects. There is a double detection of the distant cyclist - the object detection model classified it as pedestrian and cyclist.

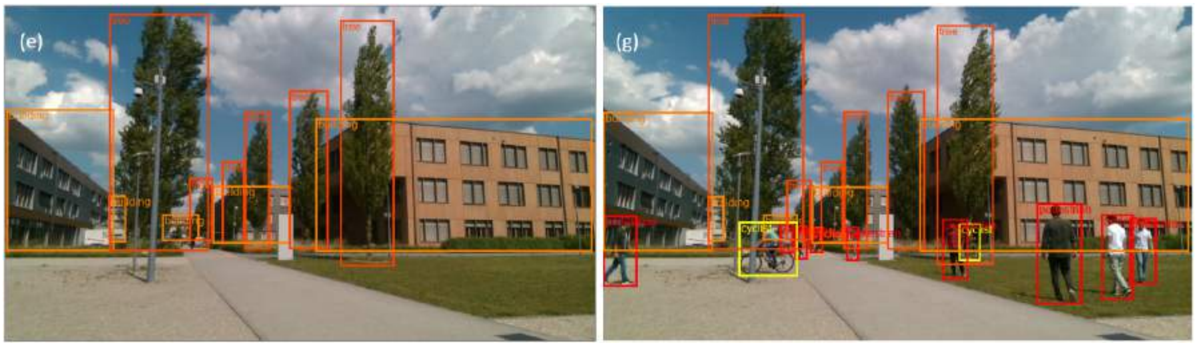


Figure 4.19: Camera based object detection examples (e,g)

In the image Figure 4.19(i) there is a person sitting on the chair. This is not a typical example for the current dataset, however, the detection of this object is correct. At the left side of the Figure 4.19(i) there is a small part of a bicycle wheel that was also detected correctly as a cyclist.

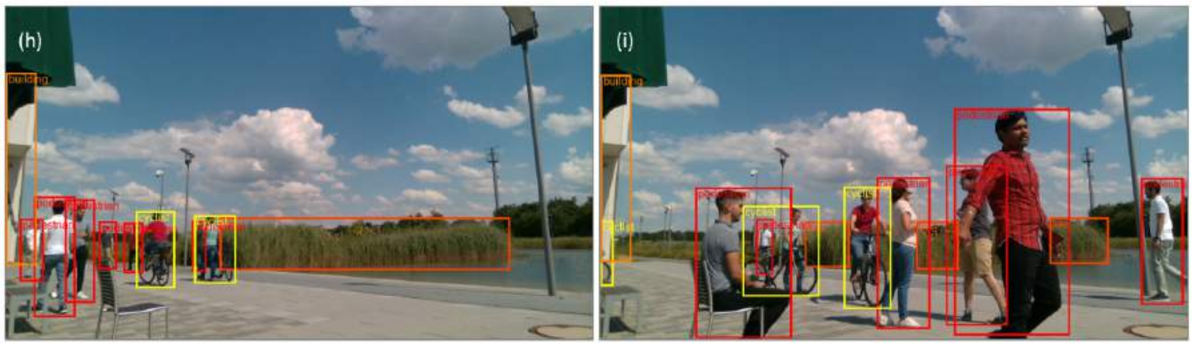


Figure 4.20: Camera based object detection examples (i,h)

4.3 Labels filtering experiments and results

4.3.1 Ground segmentation

Iterative ground segmentation fits the ground plane in a maximum of 10 attempts and returns ground and non-ground point clouds. In case the ground plane was not found in a maximum of 10 attempts, the entire point cloud is considered as non-ground. Several examples of ground segmentation results are shown at Figure 4.21. Inlier points of the fitted ground plane as marked in red. All other points that are considered as non-ground points are shown in purple.

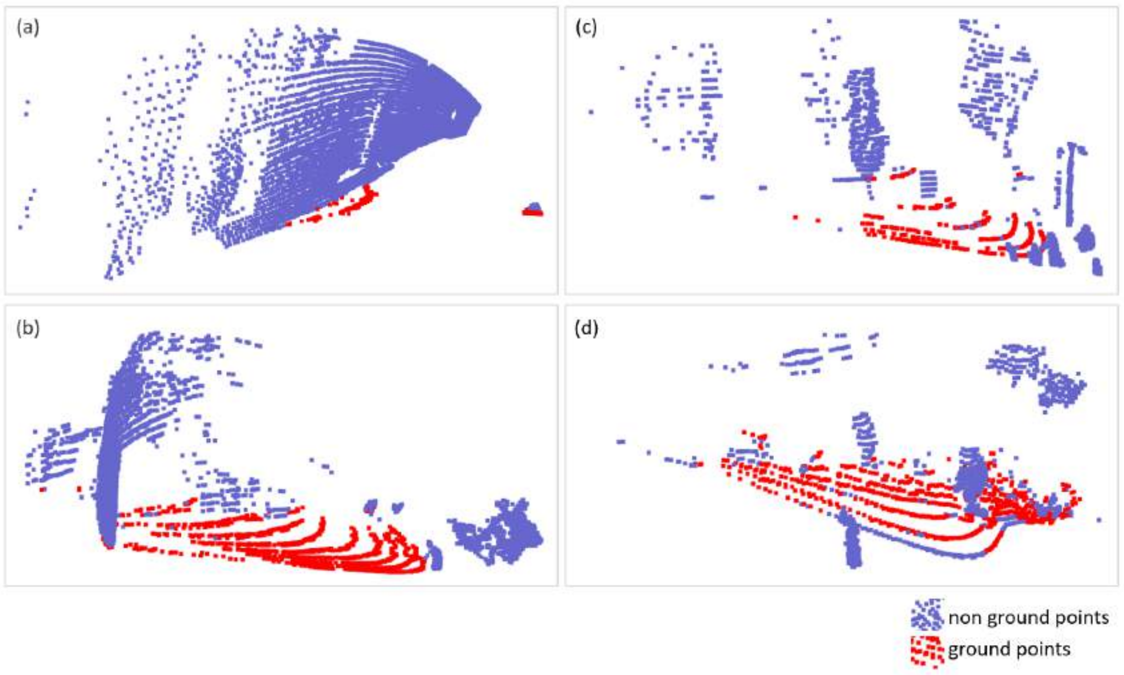


Figure 4.21: Ground segmentation. Examples

In picture (a) a complex case of segmentation is shown. There is a large number of points of a non-ground plane in the point cloud (the wall of the building). With the use of an iterative approach to ground segmentation and the subsequent exclusion of point of fitted planes which have angle α to X-Y plane greater than 30°. As a result, we can see that ground points were correctly identified. Nevertheless, a certain amount of points of another object (person) were also identified as inliers for the fitted ground plane. Pictures (b), (c), and (d) demonstrate a more common situation when the ground plane is represented by a sufficiently large number of points and is fitted by the ground segmentation algorithm in 1-3 iterations. All steps of ground segmentation of the scene (c) have been described earlier in subsubsection 3.3.3 and demonstrated in the Figure 3.17, page 50.

4.3.2 Clustering of 3D point cloud

Results of the clustering of the non ground point are presented on the Figure 4.22 below.

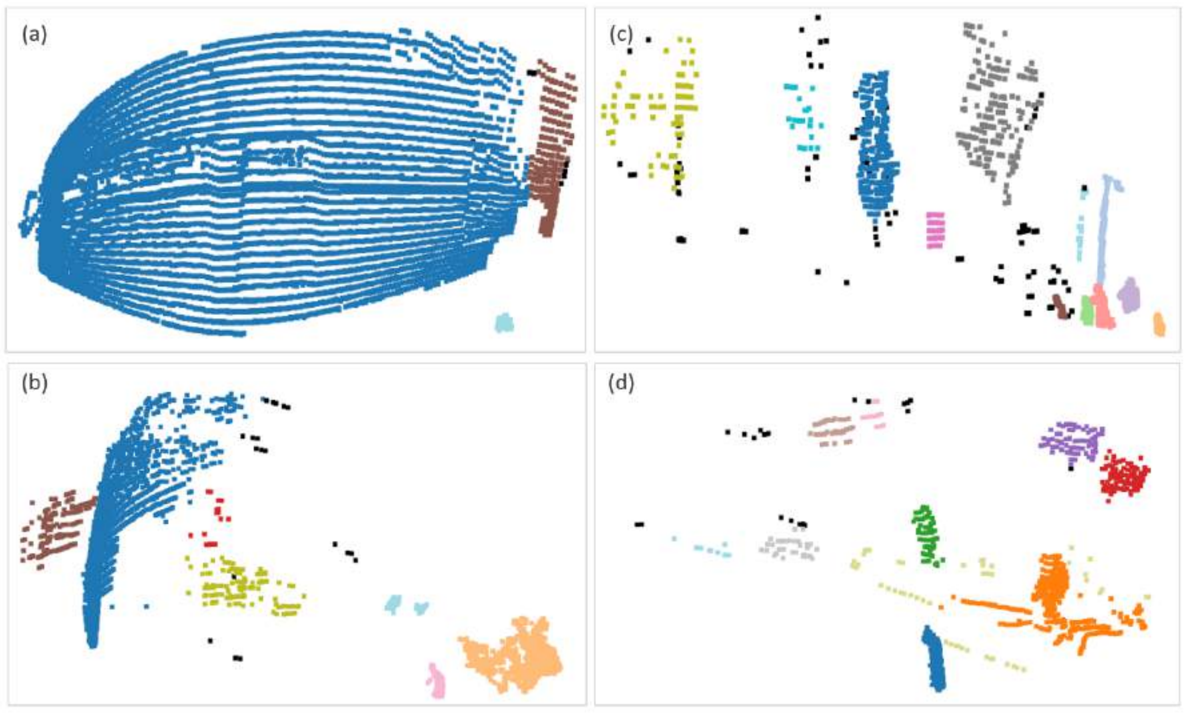


Figure 4.22: Clustering with DBSCAN. Examples

The pictures show that the objects are clustered very accurately. However, in some scenes, there are ground points that remained in the point cloud after ground elimination and are clustered together with the object. This may affect the results of the next steps, such as allocating clusters to labels and bounding box adjustment, as the presence of the ground points could change the size and location of the clusters in the 2D projection. Such a situation is demonstrated in the scene (d) in the Figure 4.22, the cluster marked in orange contains a person and several ground points.

In order to avoid such situations, the distance_threshold (the maximum distance a point can have to an estimated plane to be considered an inlier) could be increased at the ground segmentation step. However, this will also lead to the situation when a number of points belonging to objects near the ground (for example, feet of pedestrians, the lower part of a tree trunk) could be classified as ground and excluded from consideration.

The ground segmentation, as well as clustering, are performed in an unsupervised manner on the entire dataset, and for all scenes and objects in the dataset, a unified set of hyperparameters is used. Therefore, the set of hyperparameters was chosen in order to provide the most accurate results for most cases based on visual evaluation of results. However, there are some exceptions when the selected set of parameters is not optimal for individual objects.

4.3.3 Cluster allocation to labels

The Figure 4.23 below demonstrates the results of the filtering algorithm:

- The top row shows the labels obtained using the YOLOv5 object detector based on camera data,
- The 2^{nd} row shows the results of the ground plane fitting,
- The 3^{rd} row demonstrates the projection of the non-ground point cloud after clustering: outliers and clusters with the number of points less than the minimum threshold value of 30 points were removed, the label of each cluster contains cluster ID and the coordinates of the cluster center,
- The last row shows the resulting labels for the LiDAR point cloud data after all steps of the filtering algorithm: coarse filtering, ground plane fitting and removal, non-ground point cloud clustering, cluster allocation and labels adjustment.

All examples show that after filtering the labels clearly correspond to the LiDAR field of view. Labels that do not contain a minimum number of 30 points have been removed. The remaining labels have been adjusted.

Examples (a), and (b) show that the bounding boxes of the "person" have been shifted significantly and correspond to the position of points in the point cloud related to a person.

In pictures (a), and (c) the bounding boxes of the building on the left side of the scene also have been changed significantly - they have been adjusted based on the position and size of the corresponding clusters in the point cloud. The same situation is observed in picture (b). However, a small cluster was allocated to the bounding box of the building, since its center was closer to the center of the object bounding box.

Due to insufficient correspondence between the camera and LiDAR data, in the areas with a high density of objects, the algorithm could produce inaccurate results, since clusters of other objects may be located closer to the center of the bounding box than the cluster of the desired object. An example of such a situation is shown in picture (c): in the right part of the scene, there is an incorrect allocation of the tree cluster to the bounding box of the building.

Since the resulting labels are designed to be used for training the object detector based on LiDAR depth maps, such cases require manual correction.

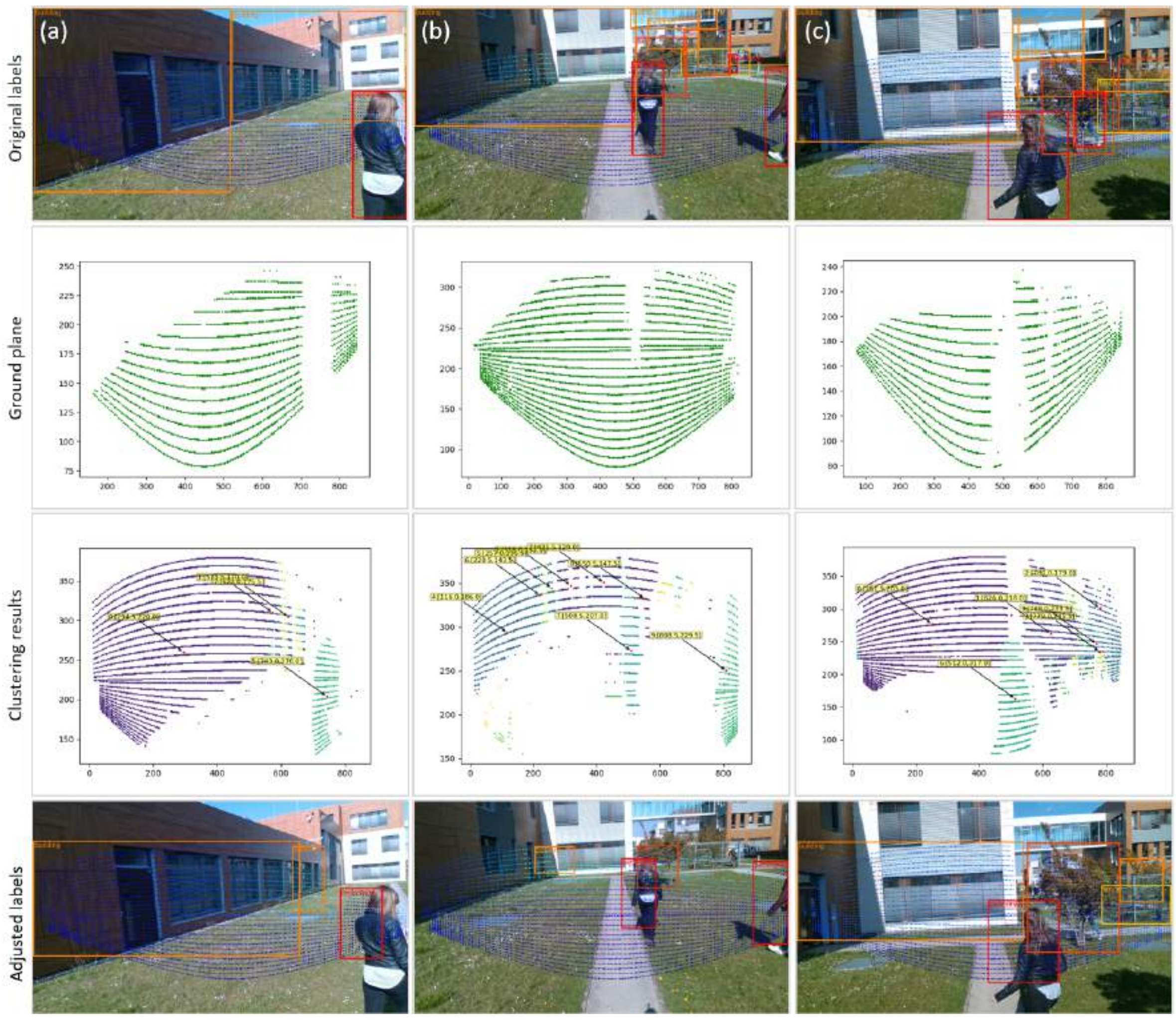


Figure 4.23: Examples of cluster to label allocation and bounding box adjustment

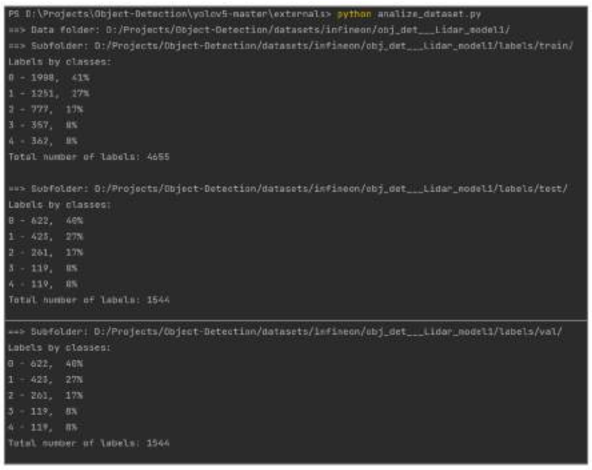
4.4 LiDAR 2D depth maps based object detection experiments and results

4.4.1 LiDAR 2D depth maps based object detection experiments

Environment and Configuration files, including model configuration and model hyper-parameters, are described in the section on camera-based object detection experiments (subsection 4.2.1).

Data preparation for training

The dataset for training the LiDAR depth map-based object detection model consists of two parts: 2D depth maps projections, described in section 3.4, and labels. Labels were obtained based on detections, obtained by a camera-based object detection model. Detections were filtered based on the field of view of LiDAR and other factors (See section 3.3 for a detailed description of the approach and algorithm). In order to improve the training data quality, the labels were further adjusted manually using the framework labelImg[7]. According to estimations, about 10% of labels have been adjusted. The resulting dataset for depth maps-based object detection model training contained 1265 images. Dataset was divided into 3 parts: train, test, and validation in proportion 0.6, 0.2, and 0.2 using a Python script. Therefore, the training subset contains 759 depth maps, test and validation subset contains 253 depth maps each. Structure of the dataset folder is shown in the Figure 4.24 below.



```
PS D:\Projects\Object-Detection\yolov5-master\externals> python analize_dataset.py
==> Data folder: D:/Projects/Object-Detection/datasets/infineon/obj_det___Lidar_model1/
==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___Lidar_model1/labels/train/
Labels by classes:
0 - 1908,  41%
1 - 1251,  27%
2 - 777,  17%
3 - 357,  8%
4 - 362,  8%
Total number of labels: 4655

==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___Lidar_model1/labels/test/
Labels by classes:
0 - 622,  40%
1 - 423,  27%
2 - 261,  17%
3 - 119,  8%
4 - 119,  8%
Total number of labels: 1544

==> Subfolder: D:/Projects/Object-Detection/datasets/infineon/obj_det___Lidar_model1/labels/val/
Labels by classes:
0 - 622,  40%
1 - 423,  27%
2 - 261,  17%
3 - 119,  8%
4 - 119,  8%
Total number of labels: 1544
```

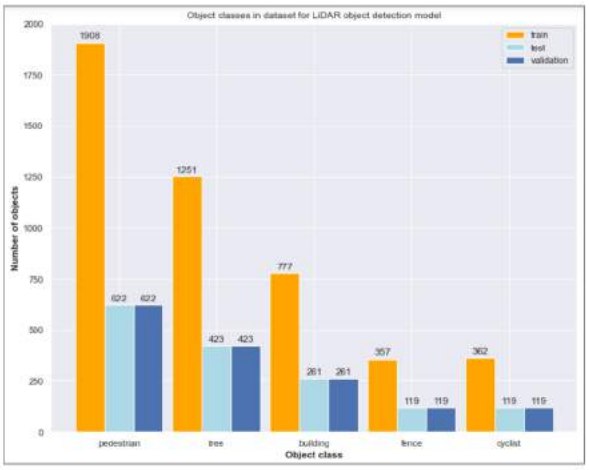


Figure 4.24: Distribution of the object classes in train, test and validation part of the dataset

As well as in training data for the camera-based object detection model, the dataset for LiDAR-based object detection is imbalanced. The distribution of the object classes is shown in the Figure 4.24 above. 8% of objects have a class label 3 - "fence" and 8% of objects have a class label 4 - "cyclist". These classes are underrepresented.

Object detection model training

Depth maps-based object detection model training was conducted with weights pretrained on the MS COCO dataset. Command line to start the training process:

```
python train.py --batch 8 --epochs 150 --data data/infineon_LOD_27-08.yaml
--weights 'yolov5m6.pt' --img 1280 --cfg models/infineon-yolov5m6_OD_14-08.yaml
--hyp data/hyps/hyp.scratch-med.yaml --cache --project 'runs-inf-08-27'
--name 'feature_extraction'
```

[7]https://github.com/heartexlabs/labelImg

The first model has been trained over 150 epochs with a batch size equal to 8. The second model has been trained over 200 epochs with a batch size equal to 16. The Hyperparameters configuration file that was used during the training, is shown in Figure 4.10. All experiments were conducted with a Stochastic Gradient Decent (SGD) optimizer with Nesterov momentum with an initial learning rate equal to 0.01, and SGD momentum β equal to 0.937. A comparison of model performance and losses is provided in the graphs below:

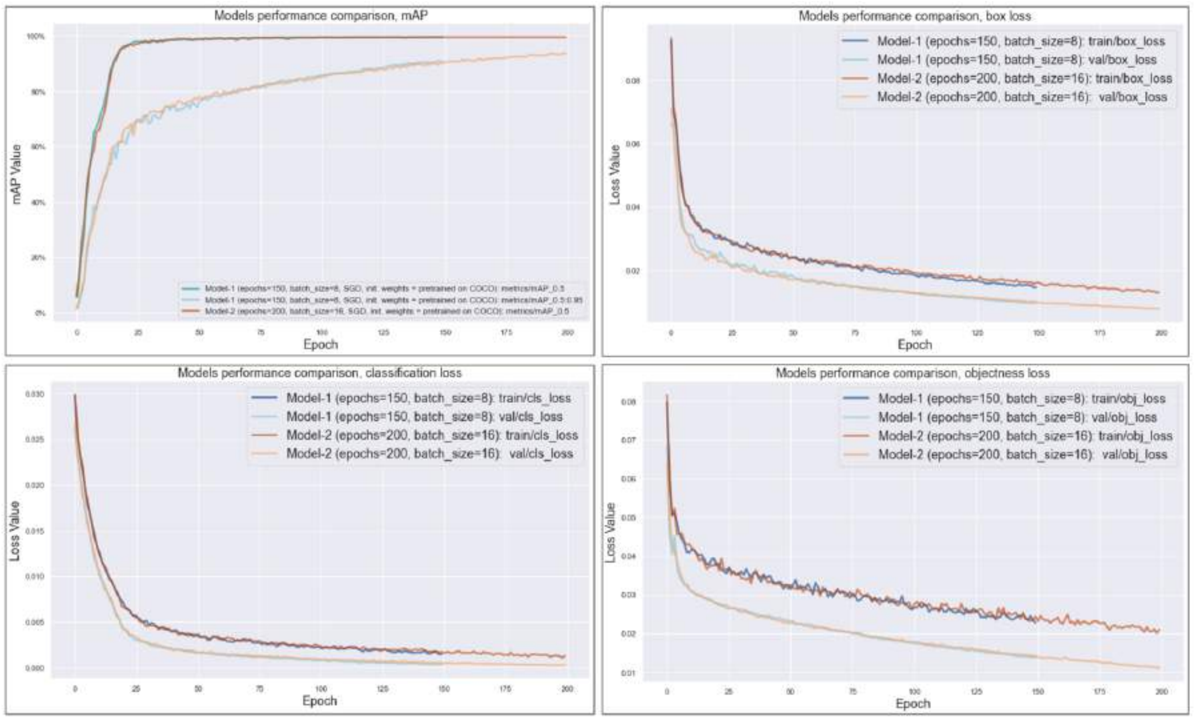


Figure 4.25: Comparison of the depth map based object detection YOLOv5m6 models performance.

Both models showed very high performance. The model that has been trained over 200 epochs (Model-2) showed slightly better performance at the end of training with the value of mAP@0.5 equal to 0.99498 and mAP@0.5:0.95 equal to 0.93667, while the model that has been trained over 150 epochs (Model-1) reached the following values: mAP@0.5 = 0.99496 and mAP@0.5:0.95 = 0.90687. Nevertheless, due to time constraints and the high quality of all models, the experiments on sensor fusion were conducted based on the Model-1 that has been trained first during 150 epochs.

4.4.2 Object detection model training results

The values of losses and performance metrics mAP@0.5 and mAP@0.5:0.95 that were achieved after training are shown in the table Table 4.3 below:

Parameter	Value
train/box_loss	0.014624
val/box_loss	0.00965
train/obj_loss	0.022768
val/obj_loss	0.013905
train/cls_loss	0.001475
val/cls_loss	0.000341
Precision	0.99623
Recall	0.99365
mAP@0.5	0.99496
mAP@0.5:0.95	0.90687

Table 4.3: Losses and performance measures values achieved at the end of the training of the object detection model over 150 epochs

The change of the objectness, classification and box losses and performance measures values over the entire training process is visualized on the figures below (Figure 4.26, Figure 4.27).

Graph Figure 4.26 shows that the most noticeable decrease in losses occurs up to epoch 75, however, after that, a steady decrease in the losses is observed as well. Validation losses are consistently lower than training losses during the whole training. Despite the final values of training and validation classification loss 0.001475 and 0.0003409 after training of LiDAR-based object detection model are lower than corresponding values 0.001838 and 0.000717 of camera-based object detection model. Some misclassifications present in resulting detections by the model for LiDAR 2D depth map data.

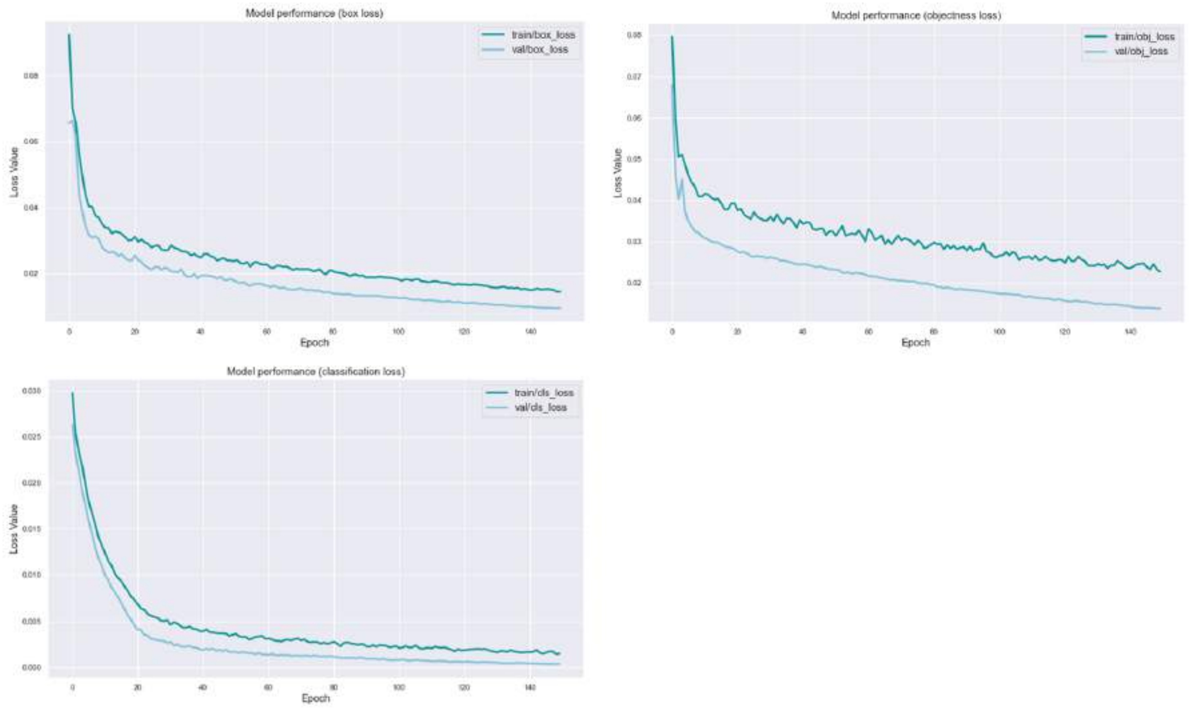


Figure 4.26: LiDAR depth maps based object detection model performance (training and validation losses)

The mAP@0.5 reaches the value of 0.9811 at epoch 31. In the rest of the training, the value of mAP@0.5 continues to increase and reaches 0.99496 at epoch 149. The mAP@0.5:0.95 converges slowlier and gradually increases throughout the training, reaching the value 0.90687 (See Figure 4.27).

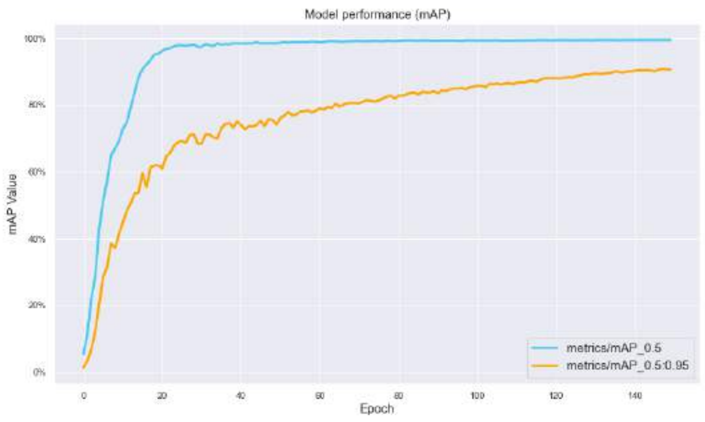


Figure 4.27: LiDAR depth maps based object detection model performance (mAP)

After the training the best and last weights are saved in files best.pt and last.pt. Both files are identical since the best performance was achieved at the end of the training. The size of each of the files is about 70Mb which is similar to the size of the weights of the camera-based object detection model.

4.4.3 Object detection results

Object detection results in daytime scenes

Examples of detections by the model are demonstrated in the Figure 4.28, Figure 4.29, Figure 4.30, and Figure 4.31 below. Detections were performed on the depth maps, nevertheless, the results are demonstrated in the projections of depth maps on the images, since such representation provides better opportunities for visual analysis.

Object detection results of several complex scenes are shown in the Figure 4.28. Scenes (b) and (c) contain the usual cityscape. In scene (e) only the heads of the pedestrians are visible to LiDAR, nevertheless, pedestrians are detected correctly. There is a highly occluded pedestrian in scene (f), however, the object is detected and the correct class is assigned to the object.

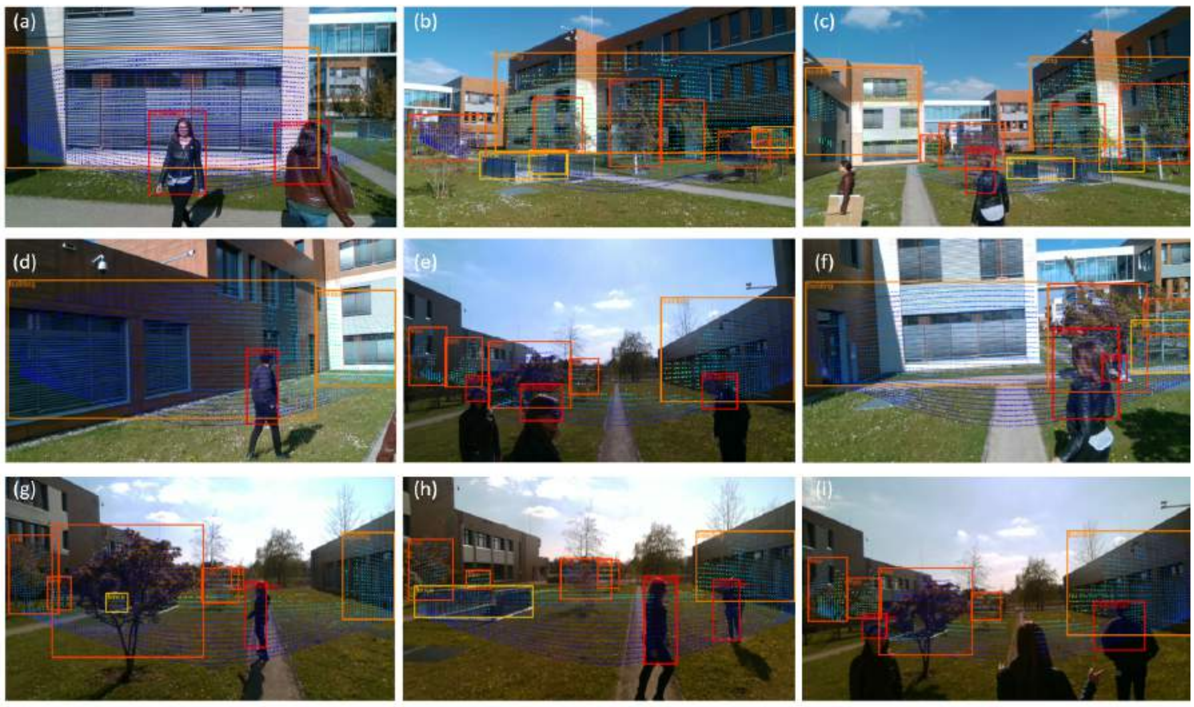


Figure 4.28: Accurate detection of complex scenes (Example 1)

Figure 4.29 contains some more examples of scenes with objects of different scales, for example, scene (i), as well as examples of scenes that contain densely located objects, for example, scene (d). Almost all objects were detected and correctly classified. Occluded pedestrians in scenes (d), and (e) are not detected.

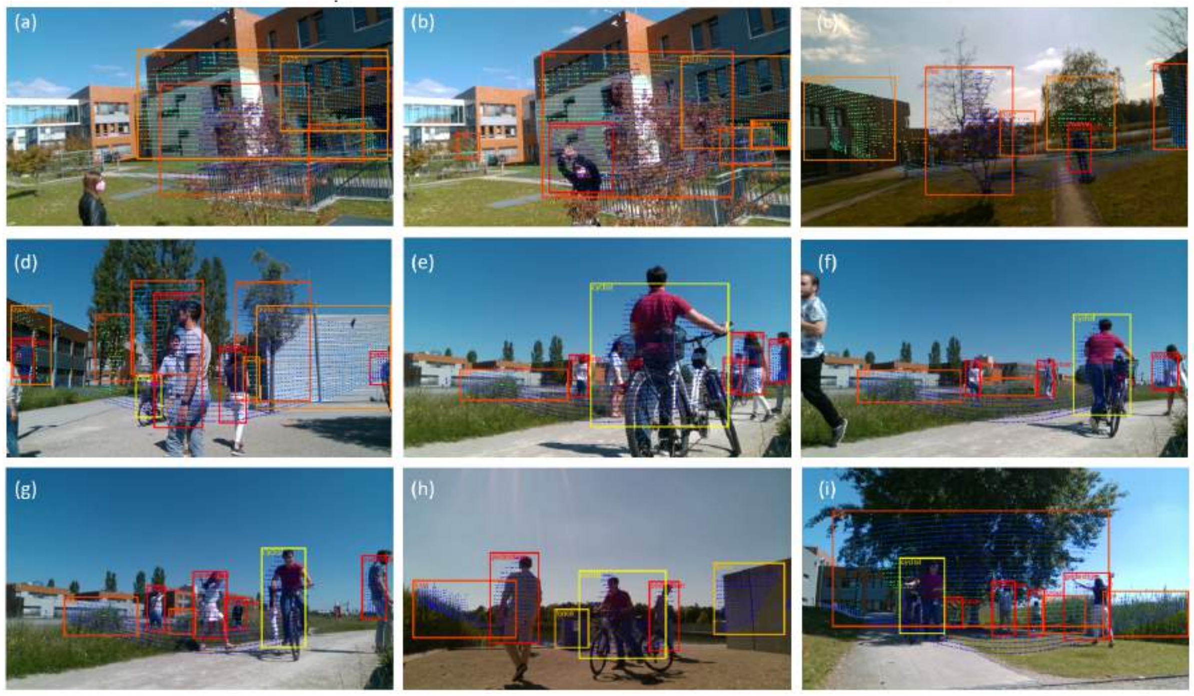


Figure 4.29: Accurate detection of complex scenes (Example 2)

Several examples of scenes that contain highly occluded objects are shown in Figure 4.30. In scenes (a) and (b) there are two pedestrians highly occluded by a cyclist. In scene (a) one of the pedestrians is detected while detection for the second is missing. In scene (b) both pedestrians are detected and classified correctly. In scene (d) there are also 2 pedestrians that are highly occluded by the cyclist. Only one of them is detected.

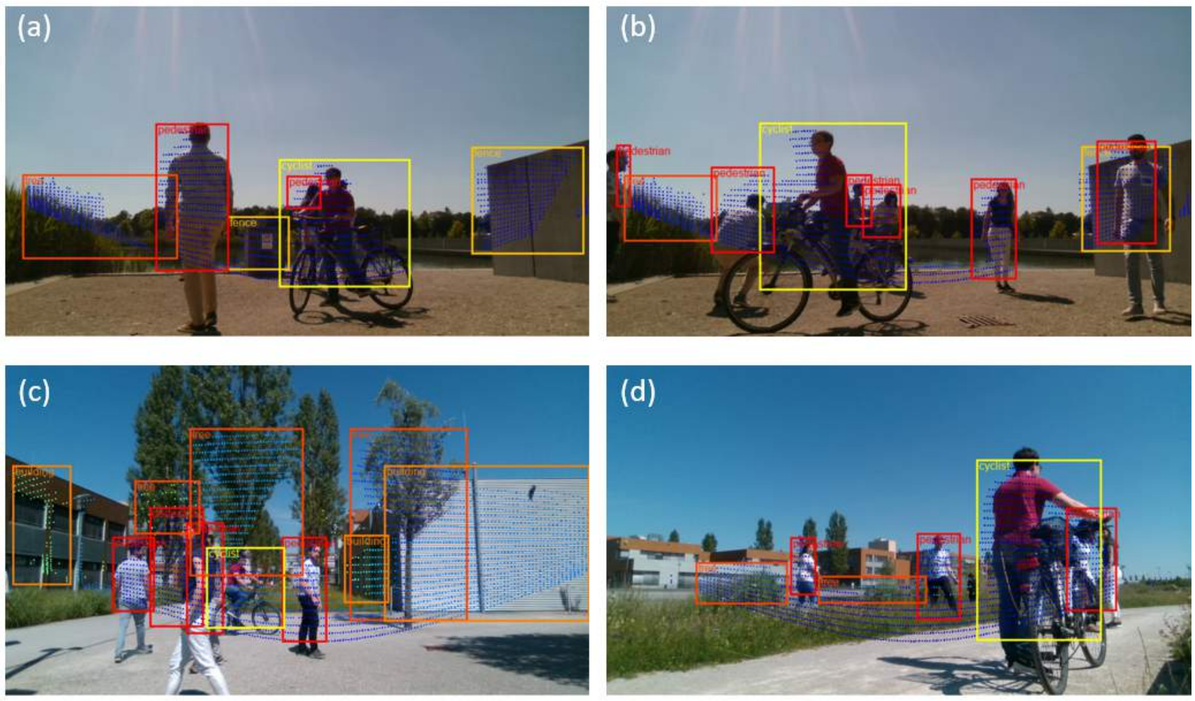


Figure 4.30: Accurate detection of highly occluded objects

In the Figure 4.31 only the upper part of the person is in the LiDAR field of view and the classification of the person varies depending on the position of the hands. In scene (b) the angle of the raised arms corresponds to the angle of the branches of the tree and therefore the person is classified as a tree. In scene (d) pedestrian is classified as a cyclist. The classification is incorrect, however, only the upper body of the person is not enough to determine if it is a cyclist or pedestrian.

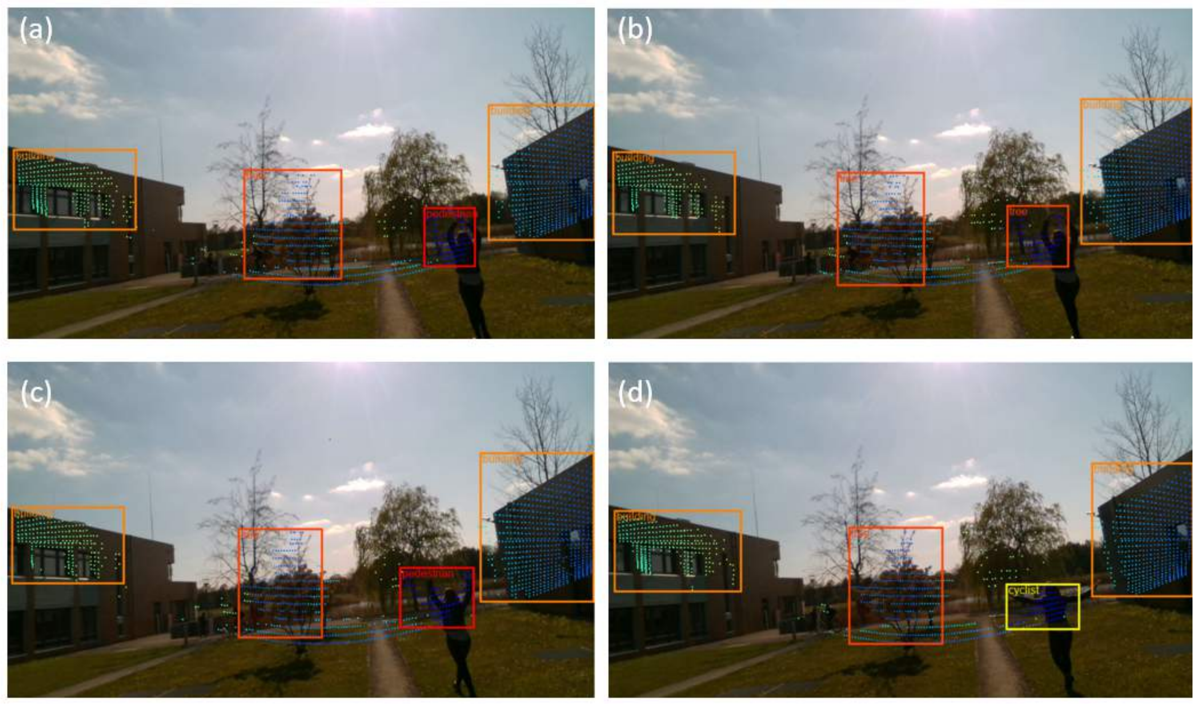


Figure 4.31: Classification of the person based on position of hands (pedestrian, tree, cyclist)

Object detection results in nighttime scenes

Poor illumination is an example of conditions when the camera sensor provides insufficient information for reliable and accurate object detection. LiDAR does not depend on illumination and, therefore, can compensate for such drawback of the camera.

In Figure 4.32, Figure 4.33, examples of detections by camera-based (left picture in each row) and LiDAR-based (right picture in each row) object detection models in nighttime are provided.

Nighttime data has not been included in the training dataset for both camera and LiDAR-based object detection models. Therefore after training on daytime data, the camera-based object detection model is not able to correctly detect and classify objects. As a result, the whole scene is detected as one object and classified as a tree. On the contrary, LiDAR-based object detection model provides quite accurate results.

In the Figure 4.32(a)-(b) all objects are detected and classified correctly. In the Figure 4.32(c) detection of the tree on the left side of the frame is missing, while one of the trees in the center of the frame is classified as a pedestrian.

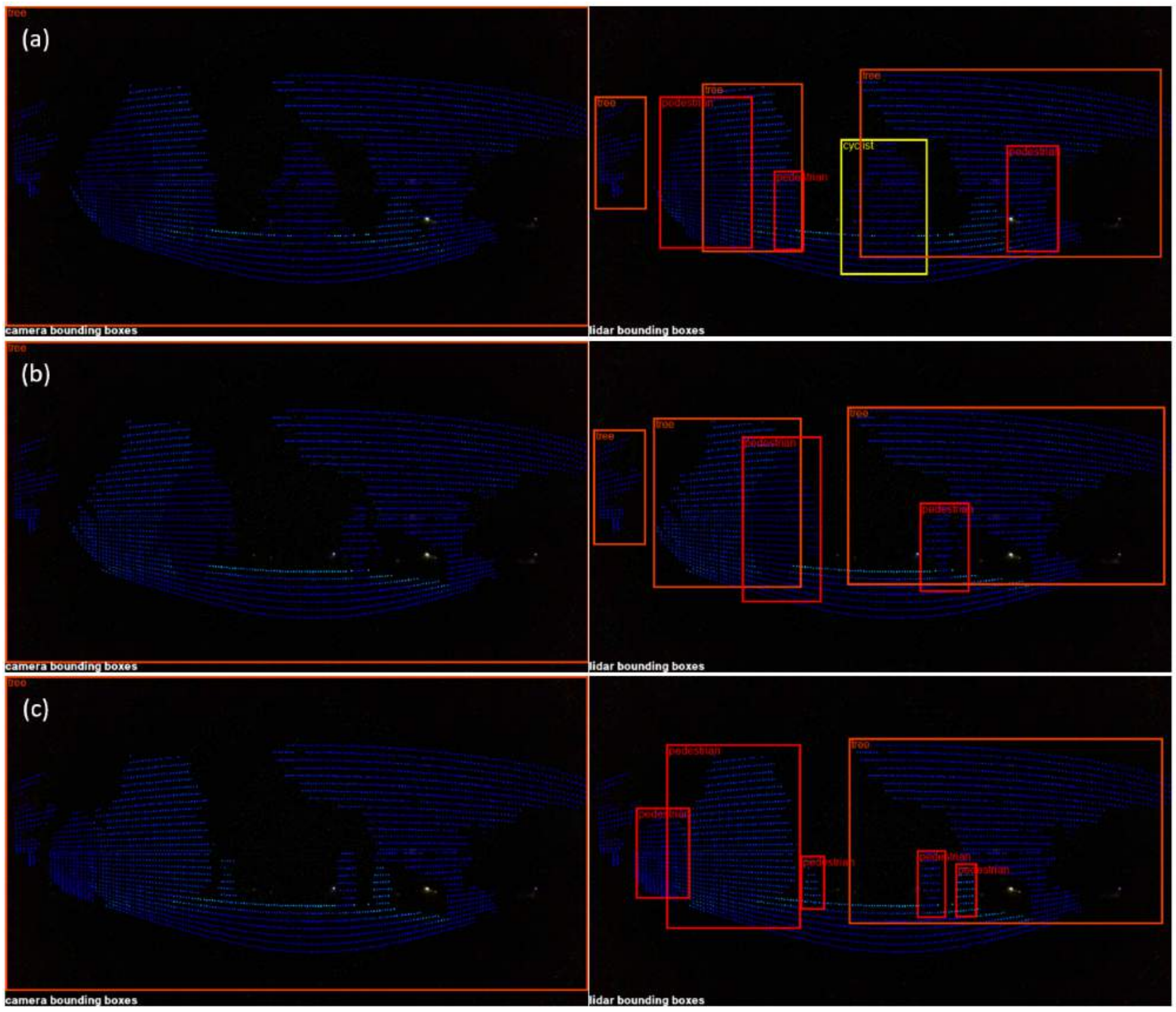


Figure 4.32: Example of object detection results by camera-based and LiDAR-based object detection model in nighttime scenes (part 1).

In the Figure 4.33(d)-(f) trees in the left upper corner are detected as buildings. It happened since 5 out of 8 data acquisition sessions in a static manner contained a building in the left upper corner of the scene, resulting in it also being present in at least 39% of training data. Therefore, the model is overfitted on this object at that position in the image. Figure 4.33(f) contains a double detection of the person as a pedestrian and cyclist. A such mistake might occur due to some misleading examples that were present in training data, for example, a person walking with a bike labeled as a cyclist. Another example is the upper part of the body of the person in the LiDAR field of view that was labeled based on information from the camera sensor as pedestrian or cyclist. However, LiDAR was deprived of the essential information in such a situation.

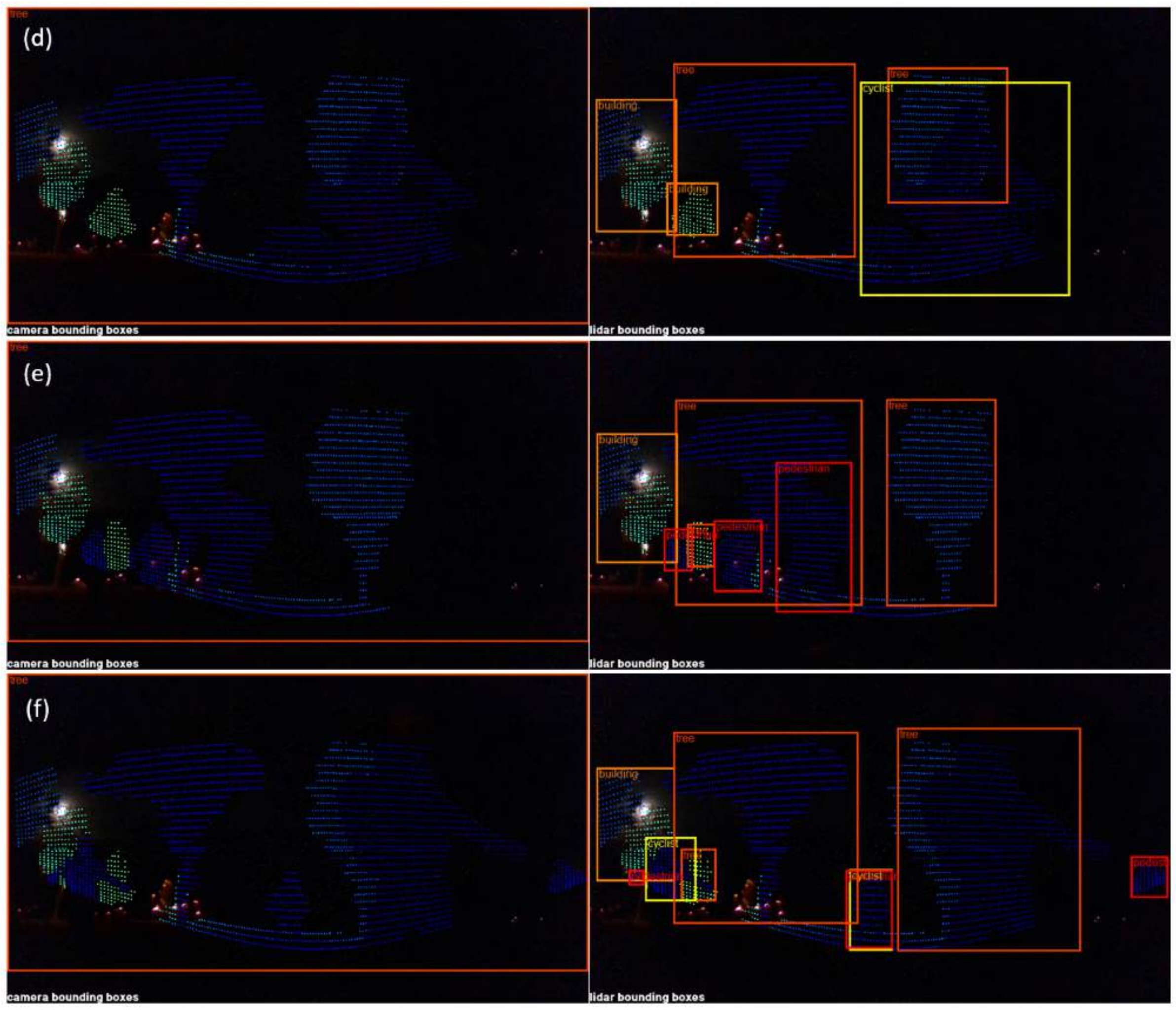


Figure 4.33: Example of object detection results by camera-based and LiDAR-based object detection model in nighttime scenes (part 2).

The LiDAR-based object detection model demonstrated a high quality of detection. Therefore, the environmental perception system benefits greatly from having two multimodal sensors. Moreover, since manual labeling of nighttime camera images is difficult, a LiDAR-based object detection model could be used for labeling such data.

Since LiDAR provides very accurate information about the environment, while ensuring a high level of data privacy, it could bring tremendous benefits and provide technological breakthroughs in many fields of application, such as security, logistics, warehousing, agriculture, robotics, solutions for smart cities, fleet management for car rental companies, military, and many others.

4.5 Fusion of Camera and LiDAR labels

In the figure Figure 4.34 below several examples of label fusion are demonstrated. For each frame (frames 1,2, and 3) there are 3 pictures:

- (a) contains camera-based object detection results,
- (b) contains LiDAR depth map-based object detection results,
- (c) contains fused labels.

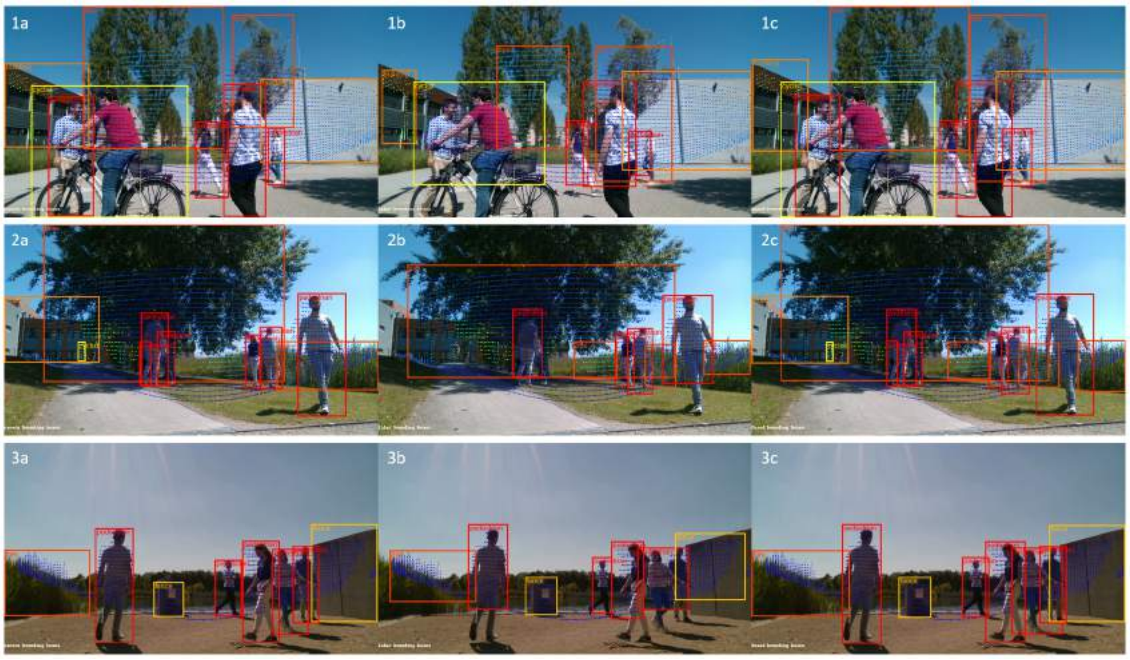


Figure 4.34: Examples of the labels fusion.

The evaluation of the obtained results was carried out visually. The utilized label fusion approach allows to preserve the most important information from both sensors and to correct the insufficiently accurate detection of the object class by the LiDAR-based object detection model. The reasons for the insufficiently accurate detection of the object class are described in the section section 4.4.

Example 1 contains a scene where most of the object classes are present. Both camera and LiDAR object detection models identified all objects in corresponding fields of view (1a, 1b). Object classes are identified correctly. Labels after the fusion step (1c) have extended bounding boxes based on matched camera and LiDAR labels. The resulting bounding boxes go beyond the LiDAR field of view and are wider in comparison to bounding boxes by the camera object detector due to a slight shift of objects between the image and the LiDAR depth map.

Example 2 demonstrates the performance of the object detection models as well as the fusion algorithm based on the scene with highly occluded and distant objects. Camera-based object detector recognizes even highly occluded pedestrians that are hardly visible to the human eye in the middle of the image (2a). These objects are not detected by a LiDAR-based object detector. The resulting fused labels (2c) contain the correct labels for these objects. In the current dataset grass was labeled as a "tree". It is recognized by both camera and LiDAR object detection models. The building is out of the LiDAR field of view and thus is not recognized in (2b). Since the cyclist is far from the sensors and is represented by a small number of LiDAR points, the object was not detected by the depth map-based object detector (2b). Nevertheless, this object was detected and correctly classified by an image-based object detector (2a) and thus has a correct label in the resulting fused labels (2c). The tree in the middle of scene 2 is detected by both camera and LiDAR object detector. The size of bounding boxes in (2a) and (2b) varies. The resulting fused label (2c) shows a more accurate bounding box in comparison to labels provided by uni-modal object detectors.

In scene (3) as in previous examples, objects are detected by uni-modal object detectors in corresponding fields of view. Camera-based object detector showed more accurate results in the detection of occluded objects: the outermost right pedestrian is detected at (3a), while LiDAR-based object detector did not detect this object (3b). The resulting labels after fusion (3c) contain correct information about this object.

The results, that are obtained after fusion and shown in 1c, 2c, and 3c, demonstrate that the chosen fusion algorithm provides accurate and correct results.

5 Conclusion and Discussion

5.1 Conclusion

Currently, there are unitary cases of real-life usage of autonomous vehicles, but mass use has not started yet. The perception of the environment, as well as the rest of the components of autonomous vehicles, is an active area of research and is constantly being improved through new technologies. Environment perception is a cornerstone technology, since other components, such as path planning, decision-making, motion and control are heavily dependent on information about the environment. Therefore, improvements in the area of environment perception can help to overcome the barrier between laboratory research in the field of autonomous vehicles and their application in real life. The lack of diverse data for training and evaluation of the quality of models for environmental perception tasks is a significant limitation for research in this area. Annotation of the data, especially 3D LiDAR data is a non-trivial task. LiDAR is one of the main types of sensors used for environmental perception tasks and only several annotated datasets for narrow domains exist nowadays.

Therefore, the goal of this work was to fill this gap by creating and implementing an approach for autolabeling of LiDAR data as well as to explore capabilities of object detection based on multimodal camera and LiDAR data.

Current work included the following parts:

- Collection of multimodal data with the use of Infineon multisensor setup.
- Data preprocessing, including LiDAR-Camera sensor calibration, depth maps extraction, data labeling, etc.
- Training of the object detection model for camera image data via transfer learning approach.
- Development and implementation of the labels filtering approach based on 3D point cloud clustering in order to obtain labels for LiDAR depth maps based object detection model training.
- Training of the object detection model for LiDAR 2D depth maps.
- Development and implementation of the sensor fusion approach based on late fusion strategy.

The field of deep learning started to rapidly develop just several years ago. Nevertheless, due to the high interest in this topic and the unlimited potential of application, technologies in this field have demonstrated impressive growth. That allowed to achieve impressive results in many areas of application, including autonomous vehicles. However, further development of technologies is necessary, both in terms of the performance of perception models, as well as in terms of hardware, since computing power is a significant limitation for the use of complex AI-based models.

5.2 Limitations of the approach

A set of object classes was limited to 5 types of objects (pedestrian, tree, building, fence, cyclist) in order to test the approach. Since the real environment is much richer, a model for real-world application should be trained on a large set of object classes, and each of them should be represented by sufficiently diverse examples to ensure that the model gives reliable results in a real situation.

Due to data privacy-related issues, the dataset was collected only around the office of the company Infineon Technologies AG with a small group of people, who agreed to participate in the data acquisition. Since the office area has several buildings of the same type, with the same number of floors and with a very similar design, and the same people present in all the scenes, the data do not have enough variability for the training of the object detection model for real-life application.

LiDAR sensor showed itself very sensitive to the position in the space - even with a slight tilt of the head by the operator wearing a helmet with sensor setup, the resulting locations of the objects in the point cloud did not match the data from the camera. Since this is crucial for the current task, it was decided to record data in a static manner - the helmet was placed on a flat surface during data acquisition. As a result, the static objects, such as trees, buildings, and fences in each session, consisting of 600-1000 scenes, are exactly the same. Such an approach also significantly reduces data variability and could lead to overfitting.

Since the sensors in the sensor setup are not protected from rain, the data was recorded only in good weather conditions. In order to create a reliable system that could be applied in real conditions, it is also necessary to train the object detection model on the appropriate data, since for example, precipitation, such as rain and snow can introduce a significant amount of noise in both camera and LiDAR data, while during the dark time of the day objects could be poorly distinguishable for the camera.

In the sensor setup that was used in the current work, sensors were not sufficiently synchronous. Moreover, the LiDAR sensor is very sensitive to position in space. For real-life application, these issues should be resolved: the sensors should be fully synchronous and must be resistant to changes in the position in space. Otherwise, poor quality data may cause the object detection model to work incorrectly, which could create a dangerous situation in a real-life situation.

5.3 Further development of the work

Work on this topic could be continued and developed in several directions:

- Research on model architectures for object detection and semantic segmentation based on 3D data in order to receive 3d bounding boxes as a model output.
  In the current work the object detection model is created based on 2D depth maps in order to identify 2D bounding boxes and classes of the objects. However, the depth information that the 3D point cloud inherently contains, could be utilized for object detection model training in order for the resulting model to be able to identify the 3d bounding boxes of the objects.

- Enrichment with data from other sensors, for example, with RADAR or/and ToF data. Since the RADAR sensor is installed on the Infineon experimental sensor setup, it is possible to obtain data on the movement direction of the objects, and thus the heading angle of the oriented 3D bounding boxes. This data could be used in object detection model training in order to get oriented 3D bounding boxes.

- Research on other fusion strategies, early and middle fusion, as well as research of architectures based on features extracted from multimodal data. Sensor fusion has great potential for application in the field of autonomous vehicles in order to ensure the required level of safety for the real-life use of autonomous vehicles. Research in this area is currently being actively developed since combining data from several modalities gives a richer representation of the environment.

- Object tracking systems development.
  Object detector processes each frame independently and identifies numerous objects in the particular frame. On the contrary, an object tracker uses the entire sequence of frames to track a particular moving object across this sequence.

- Deeper research on the topic of LiDAR-Camera calibration in order to get more accurate results and eliminate the label filtering step that is necessary in the current work while Camera and LiDAR data do not fully correspond.

List of Figures

List of Tables

Bibliography

[1] D.J.Yeong, G.Velasco-Hernandez, J.Barry, and J.Walsh. *Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review*. Sensors 2021, 21, 2140, 2021.

[2] G.Velasco-Hernandez, D.J.Yeong, J.Barry, and J.Walsh. *Autonomous Driving Architectures, Perception and Data Fusion: A Review*. In Proceedings of the 2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP 2020), 2020.

[3] J.Deng, W.Dong, R.Socher, L.-J.Li, K.Li, and L. Fei-Fei. *ImageNet: A Large-Scale Hierarchical Image Database*. 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848, 2009.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*.

[5] T.-Y.Lin, M.Maire, S.Belongie, L.Bourdev, R.Girshick, J.Hays, P.Perona, D.Ramanan, C.L.Zitnick, and P.Dollar. *Microsoft COCO: Common Objects in Context*. 1405.0312, 2014.

[6] A.Geiger, P.Lenz, and R.Urtasun. *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*. Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[7] P.Sun, H.Kretzschmar, X.Dotiwalla, A.Chouard, V.Patnaik, P.Tsui, J.Guo, Y.Zhou, Y.Chai, B.Caine, V.Vasudevan, W.Han, J.Ngiam, H.Zhao, A.Timofeev, S.Ettinger, M.Krivokon, A.Gao, A.Joshi, S.Zhao, S.Cheng, Y.Zhang, J.Shlens, Z.Chen, and D.Anguelov. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. arXiv: 1912.04838, 2019.

[8] H.Caesar, V.Bankiti, A.H.Lang, S.Vora, V.E.Liong, Q.Xu, A.Krishnan, Y.Pan, G.Baldan, and O.Beijbom. *nuScenes: A multimodal dataset for autonomous driving*. Conference on Computer Vision and Pattern Recognition (CVPR) 2020, 2020.

[9] Ultralytics. *Ultralytics/yolov5: Yolov5 in PyTorch* & *ONNX* & *CoreML* & *TFLite*. GitHub. $https://github.com/ultralytics/yolov5$, 2020.

[10] P.Viola and M.J.Jones. *Robust Real-Time Face Detection*. International Journal of Computer Vision volume 57, pages 137–154 (2004), 2004.

[11] N.Dalal and B.Triggs. *Histograms of oriented gradients for human detection*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005.

[12] A.Krizhevsky, I.Sutskever, and G.Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012.

[13] R.Girshick, J.Donahue, T.Darrell, and J.Malik. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. arXiv:1311.2524, 2014.

[14] J.Fayyad, M.A.Jaradat, D.Gruyer, and H.Najjaran. *Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review*. Sensors 2020, 20, 4220, 2021.

[15] R.Girshick. *Fast R-CNN*. arXiv:1504.08083, 2015.

[16] S.Ren, K.He, R.Girshick, and J.Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. arXiv:1506.01497, 2016.

[17] W. Liu, D.Anguelov, D.Erhan, C.Szegedy, S.Reed, C.-Y.Fu, and A.C.Berg. *SSD: Single Shot MultiBox Detector*. European conference on computer vision. Springer, p.21-37, 2016.

[18] T.-Y.Lin, P.Goyal, R.Girshick, K.He, and P.Dollar. *Focal Loss for Dense Object Detection*. arXiv:1708.02002, 2018.

[19] J. Redmon, S.Divvala, R.Girshick, and A.Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. arXiv:1506.02640, 2016.

[20] J.Redmon and A.Farhadi. *YOLO9000: Better, Faster, Stronger*. arXiv:1612.08242, 2016.

[21] J.Redmon and A.Farhadi. *Yolov3: An incremental improvement*. arXiv:1804.02767, 2018.

[22] K.He, X.Zhang, S.Ren, and J.Sun. *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385, 2015.

[23] A.Bochkovskiy, C.Wang, and H.M.Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934, 2020.

[24] C.-Y.Wang, H.-Y.M.Liao, Y.-H.Wu, P.-Y.Chen, J.-W.Hsieh, and I.-H. Yeh. *CSPNet: A new backbone that can enhance learning capability of CNN*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop), 2020.

[25] K.He, X.Zhang, S.Ren, and J.Sun. *Spatial pyramid pooling in deep convolutional networks for visual recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 37(9):1904–1916, 2015.

[26] S.Liu, L.Qi, H.Qin, J.Shi, and J. Jia. *Path aggregation network for instance segmentation*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 8759–8768, 2018.

[27] A.Balasubramaniam and S.Pasricha. *Object Detection in Autonomous Vehicles: Status and Open Challenges*. arXiv:2201.07706, 2022.

[28] Y.Zhou and O.Tuzel. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. arXiv:1711.06396, 2017.

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

[30] C.R.Qi, H.Su, K.Mo, and L.J.Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. arXiv:1612.00593, 2017.

[31] S.Shi, X.Wang, and H.Li. *PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*. arXiv::1812.04244, 2019.

[32] Z.Yang, Y.Sun, S.Liu, and J.Jia. *3DSSD: Point-based 3D single stage object detector*. Conference on Computer Vision and Pattern Recognition, IEEE, 2020, pp. 11037–11045, 2020.

[33] W.Shi and R.Rajkumar. *Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud*. arXiv:2003.01251, 2020.

[34] J.Mendez, M.Molina, N.Rodriguez, M.P.Cuellar, and D.P.Morales. *Camera-LiDAR Multi-Level Sensor Fusion for Target Detection at the Network Edge*. Sensors 2021, 21, 3992. https://doi.org/10.3390/s21123992, 2021.

[35] L.Li and M.Heizmann. *2.5D-VoteNet: Depth Map based 3D Object Detection for Real-Time Applications*. Conference: British Machine Vision Conference (BMVC) 2021, 2021.

[36] D.L.Hall and J.Llinas. *An introduction to multisensor data fusion*. IEEE, editor, Proceedings of the IEEE, volume 85, pages 6–23, 1997.

[37] D.Bellot, A.Boyer, and F.Charpillet. *A new definition of qualified gain in a data fusion process: application to telemedicine*. Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002 (IEEE Cat.No.02EX5997), vol. 2, 2002, pp. 865–872 vol.2, 2002.

[38] V.A.Sindagi, Y.Zhou, and O.Tuzel. *MVX-Net: Multimodal VoxelNet for 3D Object Detection*. arXiv:1904.01649, 2019.

[39] J.Kim, J.Kim, and J.Cho. *An advanced object classification strategy using YOLO through camera and LiDAR sensor fusion*. In Proceedings of the 2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS), Gold Coast, Australia, 16–18, 2019.

[40] Z.Zhang. *A Flexible New Technique for Camera Calibration*. IEEE TRANSACTIONS ON PATTERN ANALYSIS and MACHINE INTELLIGENCE, VOL. 22, NO. 11, NOVEMBER 2000, 2000.

[41] L.Zhou, Z.Li, and M.Kaess. *Automatic Extrinsic Calibration of a Camera and a 3D LiDAR using Line and Plane Correspondences*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[42] G.Zhaoa, J.Hua, S.Youb, and C.-C.J.Kuo. *CalibDNN: Multimodal Sensor Calibration for Perception Using Deep Neural Networks*. arXiv:2103.14793, 2021.

[43] B.Zhang and R.T.Rajan. *Multi-FEAT: Multi-Feature Edge AlignmentT for Targetless Camera-LiDAR Calibration*. arXiv:2207.07228, 2022.

[44] F.Zhuang, Z.Qi, K.Duan, D.Xi, Y.Zhu, H.Zhu, H.Xiong, and Q.He. *A Comprehensive Survey on Transfer Learning*. arXiv: 1911.02685, 2020.

[45] L.T.Triess, M.Dreissig, C. Rist, and J.M.Zollner. *A Survey on Deep Domain Adaptation for LiDAR Perception*. 2021 IEEE Intelligent Vehicles Symposium Workshops (IV) Workshops, pp. 350-357, 2021.

[46] S.J.Pan and Q.Yang. *A survey on transfer learning*. IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, 2010.

[47] G.Huang, Z.Liu, L. van der Maaten, and K.Q.Weinberger. *Densely Connected Convolutional Networks*. arXiv: 1608.06993, 2017.

[48] V.Maslej-Krešňáková, A.Kundrát, S.Mackovjak, P.Butka, S.Jaščur, I.Kolmašová, and O.Santolík. *Automatic Detection of Atmospherics and Tweek Atmospherics in Radio Spectrograms Based on a Deep Learning Approach*. Earth and Space Science, 8, e2021EA002007, 2021.

[49] H.Rezatofighi, N.Tsoi, J.Gwak, A.Sadeghian, I.Reid, and S.Savarese. *Generalized intersection over union: A metric and a loss for bounding box regression*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 658–666, 2019, 2019.

[50] H. Cantzler. *Random Sample Consensus (RANSAC)*. Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh, 1981.