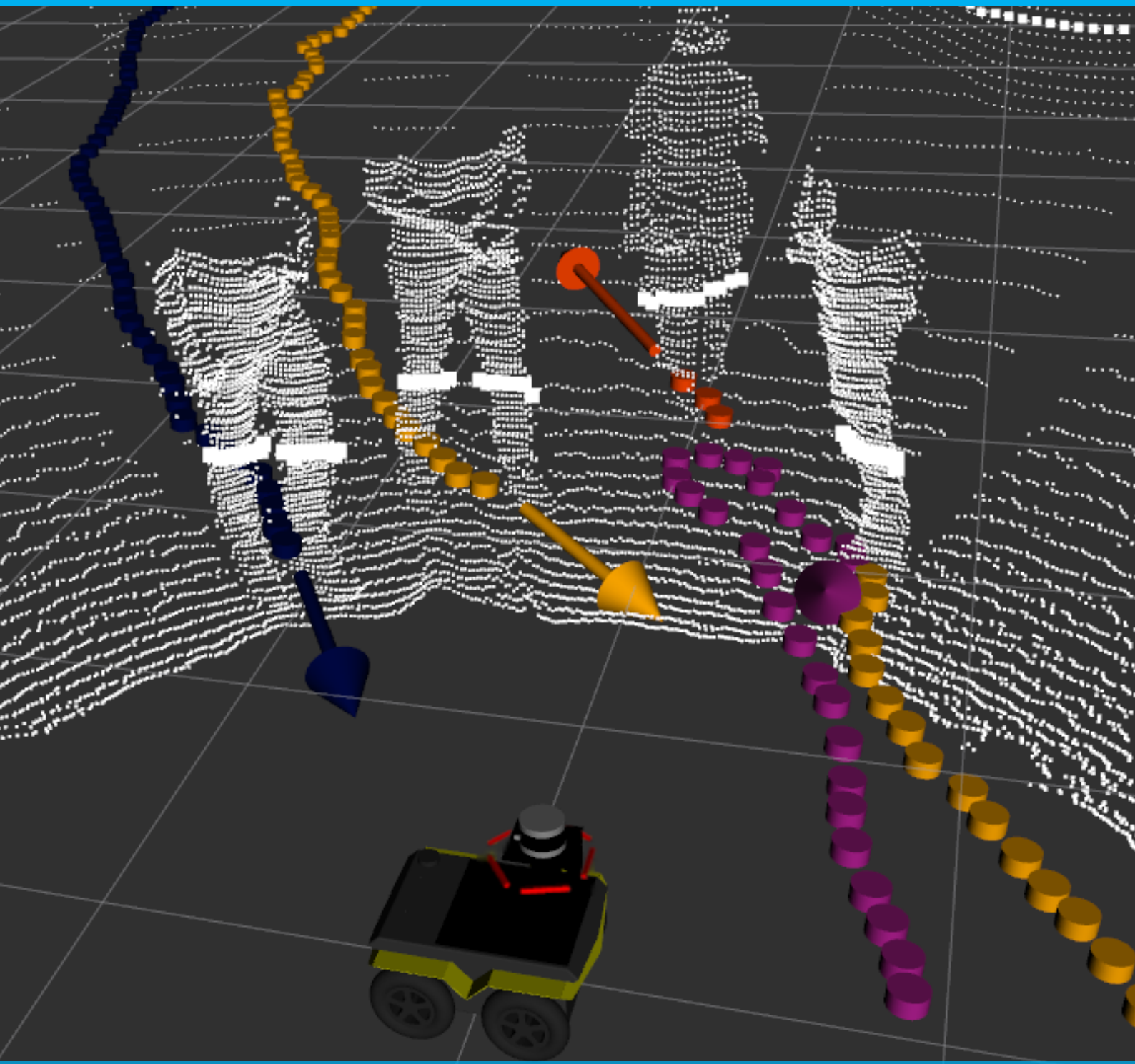
Pedestrian Detection and Tracking for Mobile Robots in Human Environments

N.H.H. Marcelis



TU Delft

Master of Science Thesis

Pedestrian Detection and Tracking for Mobile Robots in Human Environments

by

N.H.H. Marcelis

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday December 15, 2021 at 15:00 PM.

Student number:	4232712	
Project duration:	June 22, 2020 – December 15, 2021	
Thesis committee:	Ir. B. Brito,	TU Delft, daily supervisor
	Dr. J. Alonso Mora,	TU Delft, supervisor
	Dr. Ir. L. Ferranti,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

With the performance of current motion planning methods being highly dependent on the quality of the perception system, robust 3D multi-object detection and tracking are vital for autonomous driving applications. Despite all the advancements in 2D and 3D object detectors, robust tracking of pedestrians in dense scenarios is still a challenging subject for small Automated Guided Vehicles (AGVs). Most research in the field of object detection and tracking focuses on autonomous cars, neglecting the design challenges that come with small AGVs.

This thesis presents a real-time multi-modal multi-pedestrian detection and tracking pipeline for small mobile robots. The framework integrates five RGB-D cameras and a LiDAR sensor to achieve real-time pedestrian detection and tracking. The system relies on state-of-the-art 2D and 3D object detectors, a sensor fusion and filtering scheme, and a 3D object tracker. Moreover, to improve detection and tracking performance, we have collected a pedestrian dataset tailored for small AGVs. We use this dataset to train the 3D pedestrian detector and evaluate the performance of the pedestrian detectors and tracker. Evaluation of the proposed framework demonstrated the ability to robustly detect and track multiple pedestrians up to a distance of 10 meters. We open-sourced our framework at [1].

[1] https://github.com/bbrito/amr_navigation

Acknowledgements

Before you lies the thesis work ”Pedestrian Detection and Tracking for Mobile Robots in Human Environments”. The goal of this thesis is to design and implement a real-time pedestrian detection and tracking framework tailored for a small mobile robotic platform. The thesis is performed at the Cognitive Robotics Department of the Delft University of Technology.

Throughout my life, I have always been curious to find out how things work. The challenge of solving practical problems inherent to the field of engineering excites me. During my robotics minor, this passion converged to the field of mechatronics. When I applied for this master thesis, I knew that this project would embrace this passion. Although the exceptional event of a global pandemic introduced some challenges, I enjoyed doing research and development at the Cognitive Robotics Department.

I would like to thank my supervisors, Ir. Bruno Brito and Dr. Javier Alonso-Mora, for their excellent supervision and guidance during this process. Their constructive feedback was always inspiring and helped me push my work to a higher level. Furthermore, I would like to thank Chadi Salmi and Sant Brinkman for all their help and friendliness, which have made this a pleasant journey. Finally, I would like to thank my family and friends for their unconditional support throughout this somewhat strange but exciting time in my life.

N.H.H. Marcelis
Delft, December 2021

Nomenclature

AC	Alternating Current
AGV	Automated Guided Vehicle
AP	Average Precision
CPU	Central Processing Unit
FN	False Negative
FOV	Field of View
FP	False Positive
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
LiDAR	Light Detection And Ranging
MOT	Multi Object Tracking
RADAR	Radio Detection and Ranging
RGB-D	Red Green Blue And Depth
RoI	Region of Interest
RPN	Region Proposal Network
Sonar	Sound Navigation and Ranging
TN	True Negative
TP	True Positive

Contents

1 Introduction

Since before the twenty-first century, research facilities and companies have been competing to develop the first fully autonomous vehicle that is robust and safe enough to be deployed in real-world environments. Advances in the field of machine learning and the introduction of several large-scale 3D annotated datasets, such as the KITTI [36], NuScenes [13], and Waymo [99] datasets, renewed the interest of researchers and helped accelerate the development of autonomous systems. Although many car manufacturing companies have invested tremendous efforts into the development of autonomous driving systems, as of today, still none of these systems is eligible to qualify as fully autonomous (SAE automation level 5 [88]).

With Automated Guided Vehicles (AGVs) making a prominent appearance in places such as factories, hospitals, offices, and airports [22], autonomous navigation demands for a robust perception system. Navigating autonomously in these environments often involves executing maneuvers in close proximity to humans. Although the development of small AGVs benefits considerably from progress made in research focused on autonomous cars, not all advances are directly applicable to AGVs. For instance, the previously discussed large-scale 3D datasets used to train object detectors are all centered on autonomous cars. The data collected in these datasets include urban and highway scenarios captured from sensors on top of a driving car. The difference in perspective between these datasets and sensors on a small AGV is especially notable for objects close to the sensors. Since most of the small AGVs are designed to operate in indoor environments and potentially in close proximity of humans, these datasets are impractical to be used to train object detectors for small AGVs. Another challenge is the difference in available computational power between the two vehicles. Whereas autonomous cars can be equipped with powerful hardware, the hardware on small AGVs is limited by the AGV's size and battery capacity. These limitations introduce some additional challenges to the proposed perception framework.

To plan a safe and efficient trajectory, the robot needs to observe the movement of multiple dynamic objects (e.g., pedestrians) in its vicinity. This perceived motion allows a predictive-based motion planner to predict the possible future paths of pedestrians and guide the robot safely through crowded spaces. Hence, robust 3D multi-object detection and tracking is indispensable for autonomous driving applications as failing to identify and recognize pedestrians might lead to safety-related incidents [49].

1.1. Problem Description

Autonomous vehicle navigation can be visualized by the five components shown in fig. 1.1; *Perception*, *Localization & Mapping*, *Path Planning*, *Decision Making*, and *Vehicle Control* [103]. The *Perception* component uses one or multiple sensors to scan the environment for static and dynamic obstacles continuously. This information is then used by the *Localization & Mapping* component, which is responsible for localizing and tracking the ego-vehicles position in the environment. The *Path Planning* module then computes a list of possible paths considering the static and dynamic obstacles and the ego-vehicles location in the environment. The *Decision Making* component is then responsible for calculating the

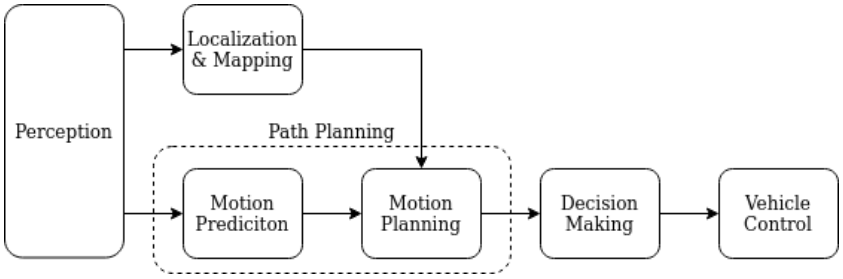


Figure 1.1: Overview of the autonomous navigation process.

optimal route based on the possible paths, current vehicle state, and environment information. The *Vehicle Control* module will then compute the appropriate steering angle and wheel torque to guide the vehicle along the optimal trajectory.

Out of the five components that make up autonomous vehicle navigation, this thesis focuses on the development and integration of a novel *Perception* module. We build upon an already existing AGV integrated with a functioning autonomous navigation framework. However, the framework's performance was unsatisfactory due to lacking a robust perception component. In order to prevent the propagation of errors in the *Perception* module throughout the entire pipeline, the goal of this thesis is to improve the *Perception* module. We will adopt pre-existing methods for the other navigation tasks.

Consider an AGV that must complete several navigation tasks in unknown environments while surrounded by a variable number of pedestrians. For an AGV to navigate safely and efficiently, obtaining an accurate and complete perception of the environment is essential. Sensory data should therefore provide a horizontal 360° field-of-view to allow object detectors to predict the locations of pedestrians in 3D space. The states of the pedestrians should be updated in real-time to ensure collision avoidance. There is no clear definition of 'real-time' in literature. It differs based on the application type; however, this framework assumes real-time detection and tracking is achieved at a minimum frequency of 10 Hz. To reduce computation complexity, we limit the maximum detection range to 10 meters. Because we tailored this framework for small AGV's navigating through crowded scenes, we assume its velocity to be similar to that of a pedestrian (e.g., 1.5 to 2 m/s). With a maximum relative velocity of 4 m/s towards an approaching pedestrian, a maximum depth perception of 10 meters would allow for 2.5 seconds time to collision. Combined with the minimum frequency of 10 Hz, this results in a 25 frames window to detect the pedestrian and calculate its trajectory.

1.1.1. Problem Formulation

For any classification problem, the tradeoff between precision and recall strongly determines the behavior of the system. The precision of an algorithm measures the accuracy of the predictions. It is defined as the ratio of correct predictions to the total number of predictions. The recall, also known as true positive rate or sensitivity, measures how many true positives are detected. It is defined as the ratio of true positives to the total number of positives. We will discuss both metrics in more detail in section 6.1.3.

In essence, the detection problem can be formulated as maximizing the classifier's precision and recall. However, these two metrics are inversely proportional; raising the confidence threshold increases precision and decreases recall. Our implementation prefers a high recall because our primary goal is to navigate the AGV amongst pedestrians safely. Nevertheless, a confidence threshold that is set too low will result in an overflow of false positives, rendering navigation impossible.
We can formulate the tracking problem as a data-association problem. The goal of the tracker is to link detected pedestrians at time step t to detected pedestrians at time step $t-1$. Once two detections are associated, they share the unique track ID belonging to the pedestrian's trajectory. The locations and track IDs of the tracked pedestrians are compared with ground-truth trajectories. In each frame, the goal of this framework is to maximize the number of correct associations.

1.2. Research Objectives

In the next section we will compactly formulate the research objectives that will drive this thesis. The main objective is:

> *To design and implement a robust, real-time, multi-modal, multi-pedestrian detection and tracking framework for small automated guided vehicles.*

This research objective is divided into the following sub-objectives:

- Design and construct a multi-modal sensor suite for the Jackal Clearpath mobile research platform that facilitates a 360° horizontal field-of-view.
- Design and implement a multi-modal multi-pedestrian detection and tracking framework based on state-of-the-art methods that can process the sensory data online at a minimum frequency of 10 Hz.
- Create a training and validation dataset by combining a large-scale online 3D pedestrian dataset and a custom labeled dataset collected with our robot.
- Create a testing dataset by collecting and annotating real-world data from a complex indoor environment to evaluate the performance of the implemented methods.
- Evaluate the performance of the proposed detection and tracking framework using the testing dataset.

1.3. Contributions

This thesis presents a robust real-time 3D pedestrian detection and tracking framework for small AGVs. Two state-of-the-art object detectors are employed to output detections using data from both RGB-D cameras and a LiDAR sensor. The data from the two detectors are fused using a fusion and filtering scheme. Moreover, we utilize a state-of-the-art 3D object tracker to provide accurate 3D depictions of the pedestrians' trajectory. Finally, a custom-designed sensor suite is manufactured and mounted on a mobile robot platform, which serves as our experimental platform.

We use a modified version of the well-established KITTI Object Detection Benchmark [36] to train our 3D object detector. Since this dataset is focused on autonomous driving, we augment its data to match the conditions of our AGV. This augmentation introduces a change in perspective which will, in particular, affect the visual representations of nearby pedestrians. To compensate for this induced distortion, we expand this augmented KITTI dataset with a custom recorded dataset. The OptiTrack motion capture system tracks the mobile robot and multiple pedestrians and enables automatic annotation of the collected data.

To summarize, our contributions are:

1. We present a robust real-time 3D pedestrian detection and tracking framework for small AGVs building on state-of-the-art methods.
2. We designed and manufactured a custom sensor suite for the Jackal mobile robot platform. The AGV will be available for future research in the fields of object detection and motion planning.
3. We release a pedestrian dataset tailored for small AGVs, which incorporates augmented data from the KITTI dataset combined with acquired data from our robot to increase close-range detection performance.
4. We show that this framework achieves real-time pedestrian detection and tracking in complex indoor scenarios. We tested our framework with a motion planner architecture and are able to achieve autonomous navigation in human environments.

1.4. Preliminaries

Throughout this report, we distinguish between 2D and 3D bounding boxes. A 2D bounding box is an axis-aligned rectangle in the image plane tightly fitting around an object. It is defined as (u, v, w, h), in which (u, v) is the top-left corner and (w, h) are the width and height of the bounding box. c indicates in which of the five cameras the bounding boxes are detected. A 3D bounding box is an oriented cuboid tightly fitting around an object in Cartesian space. It consists of 7 parameter: $(x, y, z, w, h, l, \theta)$. (x, y, z) corresponds to the location of the center point in 3D space. (w, h, l) are the width, height, and length of the 3D bounding box, and θ represents the rotation of the bounding box around the Cartesian Z-axis. The state vector of a tracked object consists of a 3D bounding box and its velocities (v_x, v_y, v_z). Finally, we denote the confidence scores of 2D and 3D detections with C.

1.5. Thesis Outline

This report is structured as follows. In chapter 2, we discuss related works concerning pedestrian detection and tracking frameworks. Chapter 3 presents an extensive overview of the design choices and construction of our AGV's sensor suite. In chapter 4, we elaborate on the augmentation and collection of the training, validation, and testing datasets used in our framework. The design and implementation of the pedestrian detection and tracking modules are covered in chapter 5 and tested against real-world experiments in chapter 6. We provide a thorough discussion on our conducted research in chapter 7. Finally, chapter 8 reflects on our research objectives, and chapter 9 concludes this thesis by providing recommendations for future research.

2

Related Works

Over the last few years, impressive advances have been made in the field of object detection and tracking. The shift towards convolution neural networks enabled new methods to achieve higher accuracy and robustness and thus outperforming nearly all previously existing methods. The tracking of dynamic objects requires accurate detections based on reliable sensory data. This chapter will provide an overview of state-of-the-art image-based (2D) and point cloud-based (3D) object detection methods, 2D and 3D object tracking methods, and several established perception frameworks.

2.1. Datasets

Large-scale datasets are an essential component to train and validate object detectors. 2D image datasets such as the MS COCO [63], and PASCAL VOC [31] have driven the research of many 2D object detectors towards incredible performances. Although the MS COCO dataset was first introduced in 2014, it is still the most popular image dataset. With the introduction of multi-modal datasets such as the KITTI [36] and later on the NuScenes [13] and Waymo [100] datasets, the research field of 3D object detection has seen major improvements over the last few years. Although KITTI is still used as the primary benchmark to compare various 3D object detection methods, the larger NuScenes and Waymo datasets are expected to become the leading datasets. For tracking in 2D, the MOTChallenge [56] is commonly used to validate 2D object tracking methods, whereas for tracking in 3D, the KITTI MOT dataset and, to a lesser extent, the NuScenes and Waymo datasets are being used. The target domain of these datasets is autonomous driving, and the data is captured from sensors on top of driving cars. The framework we propose will be used in a small AGV, making the change in perspective especially challenging for close-range objects. 3D datasets tailored for small AGVs do exist [71][87], but these are highly focused on a specific sensory setup or contain insufficient data to train 3D object detectors.

Zou et al. [131] conducted an inquiry of popular object detection datasets, and the results are listed in table 2.1. Each of these datasets is accompanied by its own evaluation server. Table 2.1 lists the year the dataset was released, the dimensionality of the data (e.g., 2D for images and 3D for point clouds), the number of individual frames in the dataset, a polar question if the frames are in sequential order, the number of annotated pedestrians and the popularity of the dataset in terms of the number of citations.

2.2. 2D Object Detector

2D object detection is an important computer vision task that involves detecting visual objects of a particular class (such as pedestrians and cars) in images. As one of the fundamental problems of computer vision, 2D object detection forms the basis of many other computer vision tasks, such as semantic segmentation and instance segmentation. Whereas traditional 2D object detectors only predict the presence of an object inside a tightly fitted rectangular bounding box, image segmentation methods provide pixel-level information on the location of an object. Image segmentation can be formulated as a classification problem of pixels. It involves partitioning images into multiple segments of predefined object categories.

Dataset	Year	Dimension	#Frames	Sequential	#Pedestrians	#Citations
KITTI [36]	2012	2D/3D	7,481	Yes & No*	9,400	5,425
MS COCO [63]	2015	2D	~ 200,000	No	~ 250,000	11,781
CityPersons [119]	2017	2D	5,000	No	35,016	271
EuroCity [12]	2018	2D	47,300	No	238,200	51
NuScenes [14]	2019	2D/3D	40,000	Yes	~ 222,000	315
JRDB [71]	2019	2D/3D	~ 60,000	Yes	~ 1,800,000	5
Waymo [99]	2020	2D/3D	198,000	Yes	~ 2,800,000	52

Table 2.1: Number of labeled pedestrians in some popular detection datasets [131]
Note: number of cites retrieved at Oct 13th 2020
* It depends on which of the KITTI datasets

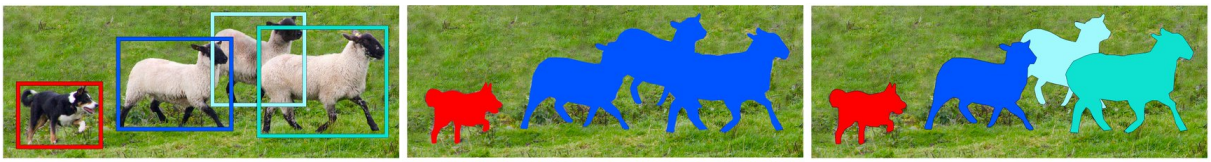


Figure 2.1: Visualization of the data provided by traditional object detectors, semantic segmentation, and instance segmentation methods [46].

2.2.1. Traditional Methods

Traditional image-based 2D object detectors can be divided into two categories: one-stage detectors and two-stage detectors. Generally, two-stage detectors such as Faster R-CNN [85] are known to have high localization and object recognition accuracy. The first of the two stages is a Region of Interest (RoI) pooling layer, sometimes referred to as a Region Proposal Network (RPN). This layer proposes candidate bounding boxes. The second stage uses these candidate bounding boxes and performs classification and bounding-box regression tasks. In contrast to two-stage detectors, one-stage detectors such as YOLOv4 [10], or SSD [67] predict bounding boxes directly from the images without a region proposal layer. By omitting region proposal network, one-stage detectors are significantly faster than two-stage detectors and preferred in real-time applications [45].

2.2.2. Image Segmentation Methods

Image segmentation methods can be divided into two categories: semantic segmentation and instance segmentation. Semantic segmentation labels each pixel of the image and classifies them to a set of object categories. Instance segmentation extends the semantic segmentation algorithm by detecting objects of interest and delineating them. This additional feature allows discerning objects of the same category and is crucial for object detection and tracking. Nevertheless, instance segmentation is highly computationally demanding, and only due to the recent technological advancements is it possible to perform these methods in real-time for a single camera. Although real-time instance segmentation methods such as [19][11] have the potential to replace the traditional object detectors, they are much slower, and their object detection performance is still inferior compared to traditional object detectors [73]. Appendix B gives an extensive overview of state-of-the-art real-time 2D object detectors.

2.3. 3D Object Detector

3D object detectors are employed to compute an object's 3D location and dimensions. Obtaining 3D locations using 2D object detectors is impractical due to the lack of depth information. Methods such as Mono3D [41] can extract 3D object proposals from monocular images; however, their performance is poor compared to methods using 3D data from stereo cameras or LiDARs.

It is expected that not all of the LiDAR's laser pulses are reflected back to the sensor, and not all pixels in the stereo camera's images can be matched. Hence, point cloud data from LiDARs or stereo

cameras consist of a variable number of 3D points sparsely distributed over 3D space. Since neural networks typically expect fixed input sizes, there is no straightforward method to structure this 3D data. Methods such as [59] and [96] opted to project the 3D point cloud onto a cylindrical or top-down (bird-eye view) image plane. This reduces the data from three to two dimensions, enabling the integration of the matured 2D object detectors. Although these methods generally worked well, valuable information is lost by ignoring one dimension. Methods such as [114] and [54] retain the three dimensions by voxelizing the point cloud. The 3D space is divided into equally sized grid cells, and points are assigned to these cells. In [80] and [51], 2D images and 3D point cloud data are combined to achieve state-of-the-art results. For example, [80] uses a 2D object detector to output 2D region proposals for the 3D detector. These 2D regions are used to extrude a 3D bounding frustum, decreasing the size of the search area considerably. Although 3D object detection datasets allow multi-modal approaches, currently, methods exclusively using LiDAR data outperform their multi-modal competitors. Appendix C gives an extensive overview of state-of-the-art real-time 3D object detectors.

2.4. Sensor Fusion

When using a multi-modal detection system, it is necessary to fuse the data of the sensors. LiDAR data in the form of point clouds do not provide texture information, which is valuable for class discrimination in object detection and classification. Monocular camera data in the form of images do not contain depth information, which is required for accurate 3D localization and size estimation. As object distance increases, the density of the point cloud quickly decreases until there are only a few points per object. Cameras generally have a much higher resolution than LiDARs and suffer less from the increase in object distance. Thus, 2D object detectors can still detect objects further away with high confidence. In order to increase the overall performance of the detection pipeline, various fusion schemes and strategies are used to combine both modalities [4].

According to van der Sluis [98] there are 6 types of sensor fusion: early, deep, late, sequential, temporal and combination thereof. Arnold et al. [4] state that the first three fusion schemes are most commonly used:

1. Early fusion: Modalities are fused at the beginning of the process. The combined data creates a new representation dependent on all modalities. An example of methods using early fusion is Complex-YOLO [96]. It projects the point cloud onto the camera image plane and adds the corresponding RGB values to the point cloud data. The fused representation is then used as input to an object detector. When the representation type is designed well in this fusion scheme, the neural network could benefit from the associated data.

2. Late fusion: Modalities are first separately and independently processed by object detectors. The output of these object detectors is then combined, which results in fused detections. This fusion scheme benefits from the maturity of object detectors. As individual object detectors already tend to perform well on their own, combining the output detections of those two could increase overall performance. A drawback of this fusion scheme is that the method to combine the multiple detection outputs is not learned by the network. It is part of the design process of the detection pipeline.

3. Deep fusion: Modalities separately enter the same neural network and are fused at a certain stage. This allows the features to interact over different layers, resulting in a more general fusion scheme. The benefits of such a fusion scheme are that the network learn how to combine the different modalities. There is no need to design a clever data representation or fuse different object detectors' detection outputs. Also, the performance of such fusion types is typically better than early or late fusion schemes. A drawback is the necessity of complex network architectures.

Schlosser et al. [90] evaluates sensor fusion at different stages of a 3D pedestrian detection pipeline. Two modalities are used, RGB images and LiDAR data. The authors conclude that late fusion yielded the best performance. Early fusion experienced only a minor performance drop compared to late fusion. This contradicts the findings of van der Sluis [98] which shows that the best performing 3D object detection methods use the early fusion scheme. Feng et al. [32] state in their paper that they did not find any conclusive evidence that one scheme is better than others. It depends on the network architecture which fusion scheme is preferred.

2.5. Object Trackers

Object tracking is essential in many autonomous driving applications. Acquiring knowledge of the previous locations of an object allows the system to use this temporal information to predict the motion of objects. Most trackers formulate the tracking problem as a data association problem. These data associations have to be made between the multiple detections in the temporal window. Due to matured and highly developed 2D object detectors, most multi-object trackers focus on perceiving motion from 2D RGB images. Pang et al. [78] learn an instance similarity matching algorithm by extracting region proposals from a 2D object detector. Data association is then performed based on feature descriptors from the instance similarity matching algorithm. Zhou et al. [125] reduce the detections from a 2D detector to a single point at the center of its bounding box. Associating detections in multiple frames relies on predicting the point displacement. A drawback of this method compared to [78] is that object re-identification is not feasible due to the lack of feature descriptors.

In contrast to 2D object trackers, 3D object trackers perform data-association in the Cartesian space. On the one hand, depth information allows 3D trackers to separate objects that might overlap in 2D space. This could increase performance, especially in cluttered environments. On the other hand, re-identification of a lost object is challenging for 3D object trackers. It relies typically on the appearance of an object, which is more easily extracted from 2D images in contrast to point clouds. Weng et al. [111] present a fast and simple 3D tracker based on a 3D Kalman filter. Data association is performed using the Hungarian Algorithm to match predictions from the Kalman filter and detections from the 3D detector. Appendix D gives an extensive overview of state-of-the-art real-time 3D object trackers.

2.6. Perception Frameworks

Earlier methods such as [75] and [74] focused on pedestrian detection solely based on stereo vision. The former method extract features from the stereo images using a Sobel edge detector. A linear Support Vector Machine classifier is trained using these features to detect pedestrians in Cartesian space. The latter method proposed pedestrian detection using 3D optical flow sequences. First, 2D optical flow is computed and extended to 3D using depth images. Then a classifier is used to identify pedestrians in the 3D optical flow data. Other methods, such as [43] and [64], proposed to detect pedestrians in LiDAR data. A single horizontal scan line is extracted from the point cloud to obtain a two-dimensional laser scan. An Ada-Boost [89] based classifier is then used to detect the legs of pedestrians in this data. The former method tracks the legs solely based on a simple nearest-neighbor matching algorithm; the latter also implements a Kalman filter to smoothen the trajectories.

Recent works have focused on neural-network-based multi-object tracking (MOT). Methods such as [125] and [78] achieve state-of-the-art 2D object tracking performance by leveraging information from the matured and highly developed 2D object detectors. Although these methods generally perform well and allow for object re-identification, additional methods are required to extend the tracking to Cartesian space.

Similar to this thesis, Shenoi et al. [91] proposed a fusion-based 3D object detection and tracking framework for a small AGV. A 3D bounding frustum of an object is extracted by extruding bounding boxes from the 2D object detector to obtain 3D region proposals. A 3D object detector then uses the relevant points in these frustums to perform 3D object detection. Furthermore, feature descriptors are extracted from 2D appearances and 3D bounding box dimensions. These two feature descriptors are fused using a 3-layer fully connected network to output a single feature descriptor. Detections are then associated by calculating appearance similarity using this fused descriptor. Although this approach is quite sophisticated, it has several drawbacks. The employed 2D [84][40], and 3D [80] object detectors are significantly outperformed by current state-of-the-art detectors; hence they have become obsolete. Furthermore, the sequential order of their 2D and 3D sensor fusion requires a pedestrian to be detected by both detectors. This procedure increases the precision of the framework at the cost of a decrease in the recall. If one of the detectors fails to identify a pedestrian, the other detector will not compensate for it.

2.6.1. Previous AGV

Our AGVs previously implemented perception module followed a multi-modal approach by combining a single stereo camera and a LiDAR sensor. A single laser scan line is extracted from the 3D point cloud and is processed by the previously described leg detector from Linder et al. [64] to detect pedestrians in Cartesian space. Furthermore, the YOLOv3 [83] 2D object detector is utilized to detect pedestrians in RGB images. The pedestrian's depth is retrieved from the corresponding depth image by extracting the depth values at the center coordinate of the 2D bounding box. A sequential fusion approach is employed to leverage the high precision from the stereo camera and the high accuracy of the LiDAR. Detections from both object detectors are matched using the nearest neighbor algorithm. If both object detectors detect a pedestrian, the detection provided by the leg detector is propagated to the tracker since its location is assumed to be more accurate. Finally, a Kalman filter is employed to track the fused pedestrians.

One of the drawbacks of this method is the weak performance of the leg detector. Although this method outputs highly accurate pedestrian depth estimates, its precision is low due to many false positives. This renders tracking pedestrians solely on LiDAR detections impractical. By employing the YOLOv3 object detector, extracting the object's depth from the stereo images, and sequentially fusing this data with the LiDAR detections, this method tries to leverage the high precision from the 2D object detector to filter out the many false positives from the leg detector. However, detections are not always accurately fused due to the exponentially increasing depth error with the object's distance. Moreover, the YOLOv3 method struggles to achieve real-time pedestrian detection at a frequency of 10 Hz, and therefore, it suggests the utilization of the faster yet less accurate YOLOv3 tiny model. This smaller model achieves real-time inference at the cost of lower detection performance.

3

Hardware Design

The design of the AGV's sensor suite is vital for a robust perception pipeline. In this chapter, we first discuss the previous designs and their shortcomings. We then formulate the design criteria and give a brief overview of possible sensor solutions. Then, we explain our proposed design and go into the design choices and several setbacks. Finally, we discuss our sensor calibration approach.

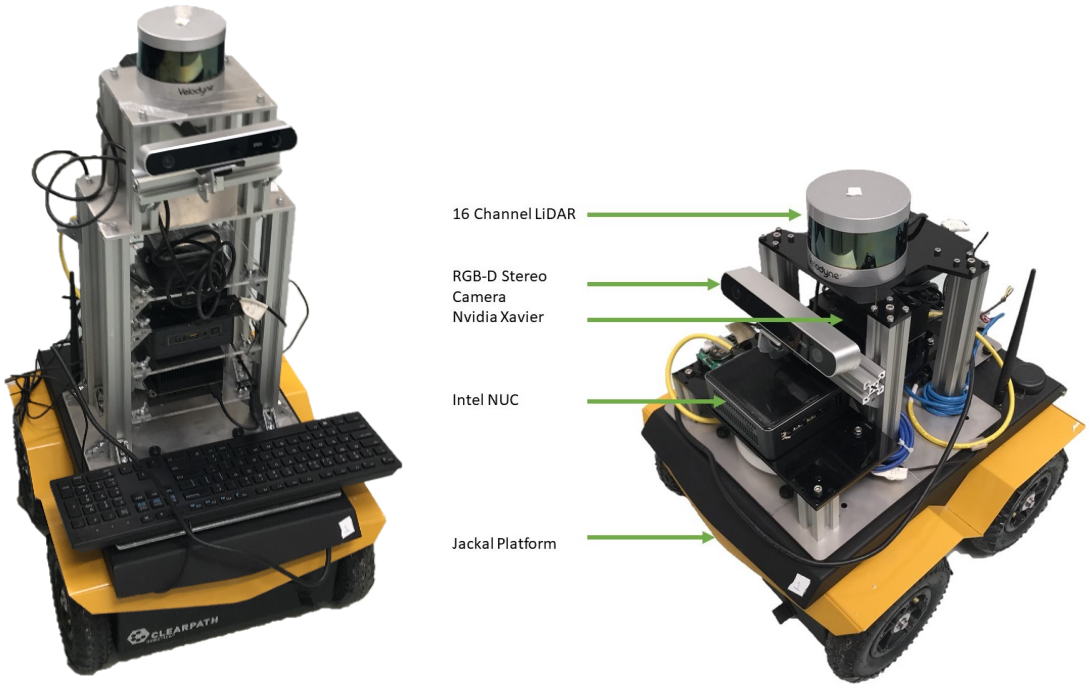
3.1. Previous Design

Before we can redesign the sensor suite for our AGV, we first need to examine the design choices of the previous designs and their limitations. Previous works on the perception system of the Jackal resulted in two sensor suite designs. The pedestrian detection and tracking framework described in section 2.6 was implemented on the Jackal V1, seen in fig. 3.1(a). In order to run this multi-modal detection and tracking pipeline, the bare-bone Jackal had to be equipped with the following components:

- **Velodyne VLP-16**: A 16 channel LiDAR sensor that outputs 3D point clouds.
- **ZED Stereolabs**: A stereo camera that outputs high-resolution RGB and depth images.
- **Intel NUC "NUC7i7BNH"**: A mini PC with a powerful CPU is used to increase the computational power of the robot needed to run all necessary algorithms onboard.
- **NVIDIA Jetson AGX Xavier**: A mini PC with a powerful GPU capable of efficiently running deep learning and computer vision algorithms. Used to run the driver for the ZED Stereolabs camera and the YOLOv3 object detector.
- **Router: TP-Link Deco M5**: Enables the user to connect to the robot wirelessly.
- **Switch: D-Link dgs-105**: A switch (or network hub) connects all devices to the same network.
- **Power Converter**: Two converters supply the 19V power required by the two mini PCs.

The sensor suite depicted in fig. 3.1(a) designed to mount these components and resembled a tower-like structure with multiple shelves. A single component was placed on each of these shelves for easy access to the hardware.

After testing the Jackal V1, one of the recommendations was to place the LiDAR and stereo camera closer to the ground plane. Because these sensors were placed relatively far away from the mobile platform, vibrations from high velocities or irregular ground planes would significantly distort the sensor readings. The sensor suite depicted in fig. 3.1(b) was designed to mitigate these vibrations. It consists of the same components as the sensor suite of Jackal V1, yet placed closer to the mobile platform. Another drawback of this sensor suite is its dependence on a single stereo camera. The detection and tracking pipeline performed reasonably well in the camera's FOV. However, the pipeline quickly lost track of the pedestrian when no longer visible in the camera frame. A solution to this problem is to mount multiple ZED stereo cameras on the robot to facilitate a horizontal 360° FOV. However,



(a) Jackal V1 (b) Jackal V2

Figure 3.1: The two previously designed sensor suites.

expanding this sensor suite with an additional 3 to 4 ZED stereo cameras is impractical since it demands too much computational power from the NVidia Xavier GPU while computing both the disparity images and running the YOLOv3 object detector. Since the pipeline already struggles to achieve real-time inference using a single camera, this solution is infeasible for this current sensor suite.

3.2. Design Criteria

- **Examine frequently used sensors in autonomous driving and assess their potential**: Since we need to redesign the sensor suite, we take the opportunity to assess if the current sensor selection is in line with our requirements. In section 3.3, we examine prominently used sensors capable of providing real-time sensory data at a minimum frequency of 10 Hz, determine their ideal operating range and the potential to classify pedestrians using their sensory data.

- **Minimize the effect of vibrations on sensor readings**: Similar to the Jackal V2, we place the sensors as close to the center of mass as possible.

- **Integrate an easy to access and convenient computer to increase development speed**: The previous design utilized two mini PCs to perform all the complex computations. Although these types of computers are preferred in embedded systems, their increased complexity hinders the rapid development and testing of new components. Moreover, the NVidia Xavier mini PC is based on an ARM CPU and a Volta GPU. Since these architectures are less commonly used, not all libraries, packages, and drivers are readily available for this system, complicating the development process.

- **Facilitate a multi-modal 360° horizontal FOV capable of detecting pedestrians up to a range of 10 meters**: The sensor suite must be designed to allow for a horizontal 360° FOV without objects blocking the sensors. Furthermore, one or multiple of its sensors must be able to accurately perceive the environment up to a range of 10 meters.

3.3. Sensor Selection

Autonomous robots rely on sensors to perceive their environment. Each sensor has its advantages and shortcomings, and often autonomous systems rely on multiple modalities to overcome the shortcomings of individual sensors.

Sensors can be divided into two groups, active and passive sensors. Active sensors actively transmit signal pulses that get reflected by objects. The sensor receives the reflected signal, and the distance to the object is measured using the Time-Of-Flight. Passive sensors receive signals which are emitted or reflected by the object being observed [103]. Van Brummelen et al. [103] researched prominent sensor arrangements on autonomous vehicles. We thoroughly discuss the sensor's selection criteria, working principles, application type, advantages, and disadvantages in appendix A. The sensor selection criteria are defined as follows:

- The sensors are frequently used in autonomous driving.
- The combined data from multiple sensors can be used to classify dynamic pedestrians.
- The combined data from multiple sensors provides a 360° horizontal field-of-view.
- The sensory data is captured at a minimum frequency of 10 Hertz.
- The combined data provides accurate depth perception up to a distance of 10 meters.

There are many sensors capable of perceiving the environment. The most frequently used active sensors in autonomous driving are active infrared cameras, LiDAR, RADAR, and Sonar, while passive sensors are the monocular and stereo cameras [4]. In practice, companies often combine monocular, stereo, and active infrared cameras and refer to them as RGB-D cameras. Each of these sensors satisfies the requirement to provide real-time data at a minimum frequency of 10 Hertz. Since our tracking framework requires the detection of pedestrians in 3D space, at least one of the modalities needs to perceive the environment in 3D. Table 3.1 provides an overview of the sensor's dimensionality, ideal operating distance, and the potential of its data to classify pedestrians. This table is based on an inquiry by Van Brummelen et al. [103] and explained in detail in appendix A.2.

	Dimension	<2m	>2m <5m	>5m <10m	>10m <20m	Pedestrian Classification
Monocular Camera	2D	✓	✓	✓	✓	✓
Stereo Camera	3D	✓	✓	~	✗	✓
Active Infrared Camera	3D	✓	✓	~	✗	~
LiDAR	3D	✗	~	✓	✓	~
RADAR	3D	✗	~	✓	✓	✗
Sonar	2D/3D	✓	✗	✗	✗	✗

Table 3.1: Sensor data dimensionality, ideal operating distance and pedestrian classification potential of frequently used sensors in autonomous driving.

Table 3.1 shows that the monocular and stereo cameras are most effective in classifying pedestrians. However, the monocular camera does not provide depth information, and depending solely on this modality is impractical. The depth accuracy of a stereo camera depends on the distance between the two cameras, the camera's resolution, and the quality of the stereo matching algorithm. Long-range depth estimation using stereo cameras typically requires high-resolution imagery and is computationally demanding. Two frequently used cameras. The ZED and RealSense D455, two frequently used stereo cameras, experience a depth error of approximately 25 and 10 centimeters at a distance of 5 meters and 100 and 40 centimeters at 10 meters, respectively. Within 5 meters, this error is small enough to generate 3D bounding boxes that overlap with the ground-truth pedestrian's 3D bounding box. However, this becomes increasingly difficult beyond the 5-meter range since depth accuracy drops quadratically with increased object distance. Therefore, we require an additional sensor to achieve accurate detection up to the desired 10-meter range. The Sonar sensor is impractical since it only performs well up to two meters and cannot be used for pedestrian classification. Although the RADAR provides accurate depth

estimates at these distances, its data is unfit to classify pedestrians; hence the RADAR is impractical. The only viable option is utilizing the LiDAR sensor and thus employing a sensor suite similar to the previous design.

3.4. Proposed Design

We based our design again on the Jackal research platform from Clearpath Robotics, and we refer to it as the Jackal V3. We use the bare-bone Jackal as a base platform and mount a custom-designed sensor suite on top, shown in fig. 3.2(b). We have equipped the sensor suite with the following components:

- **Dell G7 17 laptop**: The Jackal has an onboard computer with an Intel Core i5 4570T Dual-core processor at 2.9 GHz. Since state-of-the-art perception pipelines require both a dedicated CPU and GPU, we require extra computational power. We decided to replace the Intel NUC and NVidia Xavier mini PCs with a dedicated laptop. We decided to use the Dell G7 17, which has an Intel Core i7-10750H CPU @ 2.60GHz × 12 cores, 16 GB RAM, and an NVIDIA GeForce RTX 2070 GPU. To ensure our algorithms have access to sufficient RAM, we increased the memory from 16 GB to 32 GB. A laptop mount is fitted on the back of the Jackal to carry the laptop securely. This mount is easily accessible and allows the user to place and remove the laptop for easier testing and debugging of the detection and tracking framework.

- **Ouster OS1-64**: We have replaced the Velodyne VLP-16 LiDAR with the Ouster OS1-64. The OS1-64 is a 64 channel LiDAR sensor that outputs 3D point clouds. The LiDAR can operate at three different frequencies. At a frequency of 5, 10, and 20 Hz, the LiDAR's horizontal resolution is 2048, 1024, and 512, respectively. We aim to achieve real-time inference at a minimum of 10 Hz; hence we set the LiDAR's frequency to 10 Hz. We installed the OS1-64 driver on the Jackal's onboard computer and directly connected it to the LiDAR using an Ethernet cable to reduce the computational load of the laptop. Although this LiDAR's accuracy is slightly less than the Velodyne, this sensor provides four times as many points at a fraction of the cost. We expect the higher resolution to be beneficial to the 3D object detector's classification performance. Moreover, as stated in section 2.1, the KITTI, NuScenes, and Waymo datasets are most frequently used in the field of 3D object detection. Because the point clouds in these datasets are captured using 64 channel LiDARs, integrating a 64 channel LiDAR mitigates the problem of training at different resolutions. Lastly, since most 3D object detection methods rely on the KITTI dataset, much documentation is available on dataloaders and evaluation scripts. Since the scope of this thesis is already quite broad, we decided to use the KITTI dataset to prevent the tedious work of writing these scripts ourselves.

- **Power Converter**: The OS1-64 requires a 24V 1.5A power supply. Although the documentation of the Jackal states that it provides a 5V, 12V, and 24V power outlet, testing of the 24V power outlet indicated otherwise. The Jackal's 24V connector is directly linked to the battery. Although this battery has a nominal voltage of 24 Volts, its voltage can reach up to 30 Volts once fully charged. Because connecting the OS1-64 LiDAR directly to the 24V terminal could potentially damage the LiDAR, we decided to implement an additional 24V power converter. Furthermore, since we decided to omit the two mini PCs, we can remove their 19V power converters.

- **Intel RealSense D455**: We replaced the ZED Stereolabs stereo camera with the Intel RealSense D455 Stereo Depth Camera for close-range detections. This multi-modal camera is able to capture high-resolution RGB images and perceive depth using both a stereo camera and an active infrared camera. Whereas the performance of stereo cameras is known to decrease in difficult lighting conditions, the active infrared camera still enables the sensor to obtain depth information due to the active sensor technology. The reason for choosing this sensor is three-fold. Firstly, this RGB-D camera is ROS compatible. Secondly, in contrast to the ZED Stereolabs stereo camera requiring an external GPU to perform the complex depth computations, each Intel RealSense D455 camera performs these computations onboard. This significantly relieves the laptop's GPU, which we expect to be used extensively by the perception pipeline. Thirdly, the D455 cameras make use of a global shutter. Our predecessor mentioned a decline in performance due to vibrations and fast rotations of the robot. This phenomenon could be caused by the ZED Stereolabs

(a) CAD model of the Jackal V3 (b) Jackal V3

Figure 3.2: The Jackal V3, our mobile robot platform with its custom-designed sensor suite.

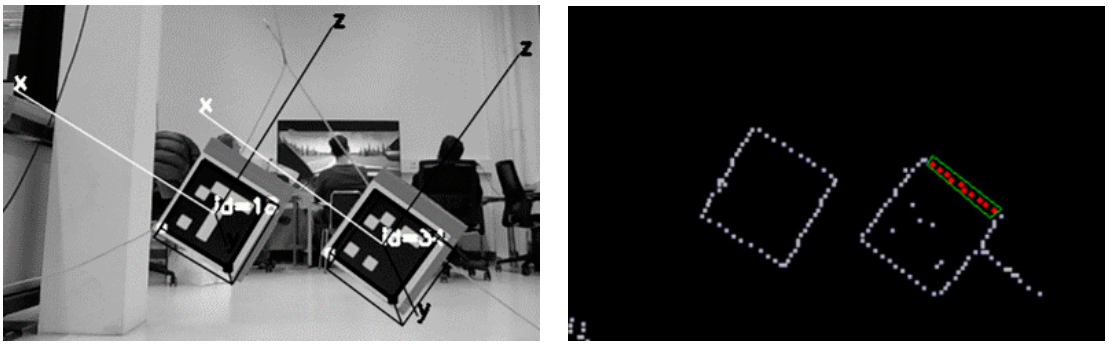
stereo camera using a rolling shutter. When capturing moving objects or translating or rotating the camera itself, images captured using a camera with a rolling shutter will appear distorted. This distortion negatively impacts the performance of the 2D object detector. Replacing the rolling shutter with a global shutter could mitigate these problems. Because the stereo depth cameras have a FOV of 87°×58°, we have arranged five of these cameras in a pentagonal shape to facilitate a 360° horizontal FOV.

- **Sensor Suite Frame**: We designed and manufactured a custom frame to mount all components on the Jackal mobile robot platform, shown in fig. 3.2(a). This frame consists of 5 vertical extrusion bars placed in a pentagonal shape. The two extrusion bars placed at the back of the robot are not fixed to the bottom plate since they would interfere with cabling to the laptop and limit the space to place components. We mount a single RGB-D stereo camera to each of the extrusion bars using a pivot joint. This joint enables us to incline the cameras upwards, increasing the visibility of pedestrians. The minimum distance at which a pedestrian of 1.80 meters height is fully visible in the camera's FOV depends on the inclination angle of the camera. The optimum is calculated using relatively simple geometry calculations and results in an upwards inclination of approximately 11°. We have placed the LiDAR on the top of the sensor suite and adjusted the height of the cameras to ensure they do not block the LiDAR's FOV. Furthermore, our design required us to mount the sensor suite backward on the mobile robot platform and reverse the AGV's driving direction. Not doing so would block the antennas of the robot due to the large dimensions of the laptop.

- **USB Hub**: Although the complex depth computations are performed onboard the Intel RealSense cameras, we decided to relieve the laptop further by running the Intel RealSense driver on the Jackal's onboard computer. Each five of the RGB-D cameras requires a separate USB 3.0 connection. Since the Jackal only provides four USB 3.0 ports, we require additional USB ports to connect all cameras. We noticed, however, that certain cameras experienced connection issues when we connected three to four cameras to the Jackal's onboard computer. We traced this problem to the USB bus not delivering enough current to power the RGB-D cameras adequately. By implementing an externally powered USB hub, we circumvented this problem.

- **Router: TP-Link Deco M5**: Enables the user to connect to the network wirelessly. Primarily used to start the robot's sensors and transmit data to Rviz for visualization purposes.

- **Switch: D-Link dgs-105**: A switch (or network hub) connects all devices to the same network.



(a) RGB image displaying detected Aruco markers and their corresponding reference frames.

(b) LiDAR points projected on the 2D image plane after filtering with an edge detection algorithm.

Figure 3.3: Visualization of the extrinsic calibration method.

3.5. Sensor Calibration

After assembling the sensor suite, sensor calibration is vital to ensure they perform at their best. Sensor calibration can be divided into two categories, intrinsic and extrinsic calibration. Intrinsic calibration focuses on optimizing the intrinsic parameters of the sensor. Intrinsic parameters solely depend on the sensor itself and not on the position of the sensor relative to the outside world. Extrinsic calibration focuses on optimizing the extrinsic parameters of the sensor. These parameters describe a sensor's pose (e.g., position and orientation) relative to an external frame of reference. Both the OS1-64 LiDAR and the RGB-D cameras have been intrinsically calibrated in the factory; hence we do not need to calibrate their intrinsic parameters. However, due to inaccuracies during the mounting of these sensors, we need to perform extrinsic sensor calibration to obtain accurate relative poses of each sensor.

3.5.1. Extrinsic Calibration

The goal of the detection and tracking pipeline is to provide pedestrian detections in the global frame. The robot uses the ROS gmapping package, which employs the Simultaneous Localization and Mapping (SLAM) algorithm, to obtain the robot's position in the global frame. SLAM uses a single horizontal LiDAR scan line to determine the robot's pose in the environment. Since SLAM computes the LiDAR's pose relative to the global frame, we decided to calibrate the RGB-D cameras pose relative to the LiDAR. This creates a clear top-down structure in the reference frames.

We decided to use the sensor calibration method from A. Dhall et al. [26] to calibrate each RGB-D sensor relative to the LiDAR[1]. This method uses Aruco markers [35] placed on square cardboard plates to determine the position and orientation of these markers relative to the RGB-D sensor, depicted in fig. 3.3(a). Aruco markers are binary square fiducial markers that let cameras easily determine their pose due to their unique appearance. Furthermore, the method project the LiDAR points onto a 2D plane and performs an edge detection algorithm to obtain the edges of the square cardboard plate, depicted in fig. 3.3(b). The user then selects the LiDAR points belonging to the eight edges of the two squares, visualized by the red-colored points in fig. 3.3(b). The method can then compute the relative pose between the RGB-D camera and the LiDAR using the selected edges and predefined information on the size of the square cardboard plates and the Aruco markers.

[1] https://github.com/ankitdhall/lidar_camera_calibration

4

Data Collection

4.1. Training & Validation Dataset

Training object detectors that generalize well typically requires large-scale datasets. As stated in section 2.1, most 3D pedestrian datasets focus on autonomous driving scenarios. Datasets tailored for small AGVs are scarce and generally are limited to specific sensor suites. To address this shortcoming, we introduce a 3D pedestrian dataset consisting of:

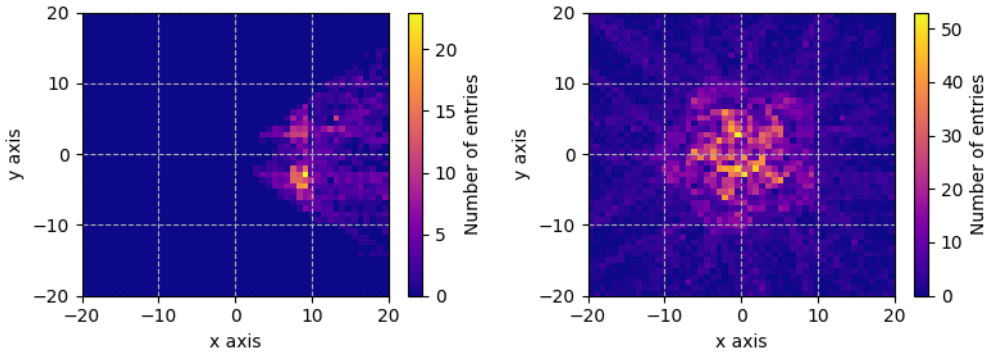
1. Data from the KITTI dataset that is augmented to resemble being captured from a small AGV.
2. Collected data using the robot's sensors in combination with accurate pose information from an OptiTrack motion capture system.
3. A hard-negative dataset without pedestrians collected by the robot's sensors in various indoor spaces.

4.1.1. Augmented KITTI

Although the NuScenes and Waymo datasets are larger than the KITTI dataset, research groups still use KITTI, and most state-of-the-art 3D object detection methods use KITTI to validate their detections. Documentation on the KITTI dataset is abundant, and many open-source data loaders and evaluation scripts are available; hence, this framework employs the KITTI dataset. However, we need to address several drawbacks before the quality of its data is sufficient to train object detectors for small AGVs. The first drawback is the difference in height between the sensor suites. The KITTI's sensor suite is located on top of a car at the height of 1.73 meters, whereas we placed our sensor suite at the height of 0.50 meters. Shifting the 3D point cloud and annotated labels upwards by 1.23 meters results in data mimicking the robot's perspective. The second drawback is the limited FOV of the KITTI dataset, seen in fig. 4.1(a). Although its point cloud consists of 360° data, the cameras only capture images in front of the vehicle with a FOV of approximately $\pm$ 43°. Since 2D and 3D annotations are only available for pedestrians visible in the camera frame, it is common practice to crop the point cloud to the camera's FOV. To facilitate 360° annotated training data for 3D object detector, four additional datasets are generated by rotating the cropped point clouds and annotations by 72°, 144°, 216°, and 288° around its axis perpendicular to the ground plane, respectively.

4.1.2. OptiTrack Dataset

Augmenting the KITTI data by shifting the point cloud upwards introduces a change in perspective. Since nearby pedestrians are only partially visible to the LiDAR due to its narrow vertical FOV, the effect of the perspective change is inversely proportional to its distance. A slight advantage is that most pedestrians are clustered at a distance of approximately 8 meters, while no pedestrians are present within the first 2 meters, seen in fig. 4.2. This mitigates the impact of the change in perspective. Nevertheless, we expect an object detector trained solely on this data to perform poorly at detecting close-range pedestrians. Yet, since indoor environments result in closer encounters with humans, these



(a) KITTI dataset (b) Custom dataset

Figure 4.1: Heatmap of pedestrian locations as seen from a top view.

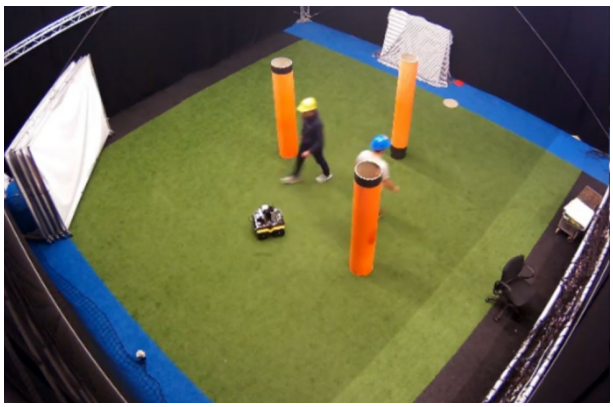


Figure 4.2: Dataset recording using the OptiTrack motion capture system. Two pedestrians are walking around wearing safety helmets fitted with reflective markers.

nearby pedestrians pose the greatest risk of collision.

We introduce a custom recorded dataset to resolve these problems. Incorporating this data into the training phase of the 3D object detector is expected to increase overall performance, especially for close-range detections. Since manually labeling the 3D point clouds is labor-intensive, we employ the OptiTrack motion tracking system to generate annotations automatically. This system tracks the poses of the robot and several pedestrians while both are moving in an enclosed environment, as seen in fig. 4.2. Ideally, we would take full advantage of the OptiTrack's capabilities by wearing suits extensively covered with reflective markers to obtain the precise location and dimensions of the pedestrian. However, since we only have helmets with reflective markers at our disposal, we can only extract the pedestrian's pose and height. By placing a minimum of three reflective markers on the robot and helmets worn by the pedestrians, an overhead camera array can accurately track the objects' locations and orientations.

We designed a python-based software package that automatically extracts 3D bounding box annotations. We first synchronize the LiDAR point clouds and the OptiTrack pose information based on their timestamps. Since the OptiTrack poses are represented in the global coordinate frame, we compute the pedestrian's pose relative to the AGV's LiDAR sensor. Because we only have information on the pedestrian's height, we use a fixed width and length value to obtain a rough estimate of the 3D bounding box. To obtain accurate 3D bounding boxes, we introduce a bounding box refinement algorithm. We first extract all 3D points from the point cloud that fall within the rough bounding box. We refine the obtained cuboid's dimensions by cropping its width and length values until the cuboid's planes encounter the outer points of the pedestrian. Since adjusting the boundaries of the bounding box shifts the center, we need to recalculate the center based on its new dimensions. Finally, we transform the

	Training		Validation	
Dataset	**# Frames**	**# Labels**	**# Frames**	**# Labels**
Original KITTI	3712	2207	3769	2280
Augmented KITTI	18560	11035	18845	11400
OptiTrack	1789	3419	1740	3330
Hard-Negative	3211	0	0	0
Ours	23560	14454	20585	14730

Table 4.1: Overview of the number of frames and labeled pedestrians in the original KITTI dataset, our augmented KITTI dataset, and our collected OptiTrack and Hard-Negative datasets.

3D bounding box from LiDAR coordinate frame to camera coordinate frame to match the KITTI format.

Our refinement method is also capable of determining the accurate height of a pedestrian. However, we decided to use to original height provided by the OptiTrack poses instead. The reasoning behind this decision is that pedestrians close to the LiDAR experience truncation due to its narrow vertical FOV. These truncated regions are characterizing for the pedestrian's visual representation and add valuable information to the annotation. We expect that including these empty regions in the pedestrian's 3D bounding box helps the classification task of the 3D object detector.

4.1.3. Hard Negative Dataset

The KITTI and OptiTrack datasets involve respectively outdoor urban environments and one highly specific indoor environment. Training an object detector on these datasets will most likely result in a biased model, and we expect a decline in detection performance when transitioning to unfamiliar indoor spaces. To support good generalization of the object detector, we collected an additional dataset featuring various indoor environments. The sensory data is collected after faculty closing hours to ensure the absence of pedestrians in the data. This procedure allows for relatively long recording periods yet prevents the tedious work of manually labeling the data. By introducing these hard negatives, we expect that the object detector will generalize better to new environments.

These three subsets are combined to form our custom dataset. 3D object detectors trained on the KITTI dataset commonly use a split ratio of 50/50 [114][68][54][21][20][127]. Similar to those methods, we adopt the same policy and use approximately half of the data to train the 3D object detector and the other half to validate its performance. Figure 4.1(b) shows the heatmap of the pedestrian locations in the original KITTI dataset as well as in our newly introduced dataset, and table 4.1 gives an overview of the number of frames and labeled objects in the datasets.

4.2. Testing Dataset

To assess the capabilities and limitations of our proposed implementation, we require a custom annotated dataset captured in real-time with our AGV. We specifically designed the previously described custom dataset to train and validate the 3D object detector and not to test the performance of our proposed framework. We decided not to use this dataset for testing for the following reasons:

1. The dataset is partially populated with data from the augmented KITTI dataset, which is slightly different from our sensory data.

2. The OptiTrack environment is too simplistic. Although we placed several different objects inside this ten by ten-meter space, there is not much variation. Training and testing the 3D detector in the same environment will most likely bias the results.

3. The number of pedestrians in the dataset is limited. Due to Covid-19 restrictions, it was impossible to populate a large group of pedestrians inside the relatively small OptiTrack environment. This resulted in the OptiTrack dataset only featuring two different pedestrians.

To guarantee the reliability of the results, we introduce an additional testing dataset containing RGB-D images, point clouds, and 2D/3D annotations. The dataset contains 5 minutes of sensor data collected

by our mobile robot while driving in an uncontrolled indoor environment, out of which 1 minute is manually labeled. We include the following ground truth labels in our dataset: 1) 6041 2D bounding boxes of the human/pedestrian class in 2400 images captured by 4 out of the 5 RGB cameras. Due to time constraints, we did not yet annotate the images from the fifth camera. 2) 4954 annotated 3D oriented bounding boxes for the pedestrian class in 571 point clouds captured by the LiDAR sensor. 3) temporal ID association for all 3D annotated pedestrians in the point clouds. The data is manually labeled using the open-source annotation software 3D-BAT[1] [130] and CVAT[2] [104].

Although we designed our framework to run real-time and online on our robot platform, we performed inference offline while evaluating the performance. Recording the sensory data (e.g., point clouds, RGB, and depth images) slows down computations considerably and affects our framework's real-time capabilities. As an alternative, the sensory data is recorded and stored inside a rosbag. This rosbag enables us to playback the recorded data and simulate the real-time behavior of the robot.

[1] https://github.com/walzimmer/3d-bat
[2] https://cvat.org/

5

Software Design

In this chapter, we propose a novel multi-modal multi-pedestrian detection and tracking framework. Our proposed framework is depicted in fig. 5.1. We follow a multi-modal approach by using both a LiDAR sensor and five RGB-D cameras. This framework employs two separate state-of-the-art object detectors. One detector processes 2D RGB images, and one processes 3D point cloud data. We start by explaining the implementation details of the 2D object detector in section 5.1 and the procedure to translate 2D to 3D bounding boxes in section 5.2. In section 5.3, we discuss the implemented 3D object detector and in section 5.4 we elaborate on the sensor fusion and filtering scheme. Finally, we describe the 3D object tracker in section 5.5.

5.1. 2D Detection

This framework employs a 2D object detector that processes the images from five RGB cameras and outputs 2D bounding boxes. Since this framework relies on tracking 3D bounding, and RGB images do not contain depth information, we need to translate these 2D bounding boxes to 3D. Section 5.2 further elaborates on this method.

An extensive literature study conducted by Bochkovskiy et al. [10] indicated that, at the time of writing, the YOLOv4 [10] method was the best performing detector that satisfies the real-time criteria of 10 Hz. For that reason, we employed the state-of-the-art YOLOv4 architecture, shown in fig. 5.2, as our 2D object detector. YOLOv4 is a one-stage detector and focuses on fast object detection in real-time applications. Typically, the three different stages in the network are referred to as the Backbone, the Neck, and the Head. The Backbone (also known as Dense Connection Block or CSPResBlock) of the network consists of multiple convolutional layers to downsample the input data and extract feature maps at various downsampling stages. In order to enrich the information that feeds into the Head, the Neck consists of a Spatial Pyramid Pooling (SPP) that concatenates the various feature maps from the Backbone and a Path Aggregation Network (PAN) that propagates low-layer information to the top-

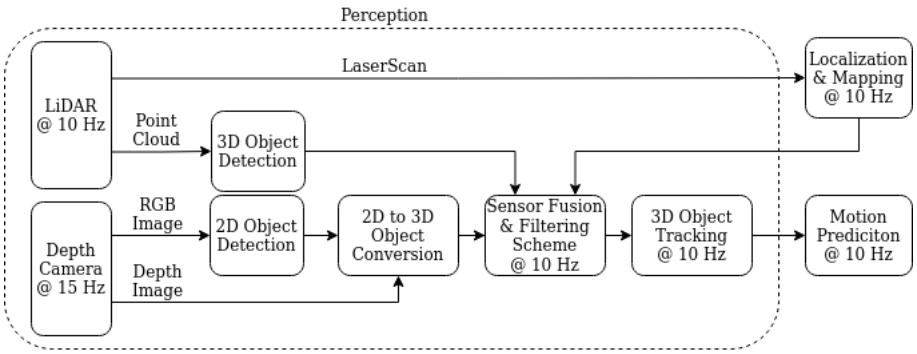


Figure 5.1: A detailed schematic the proposed perception framework.

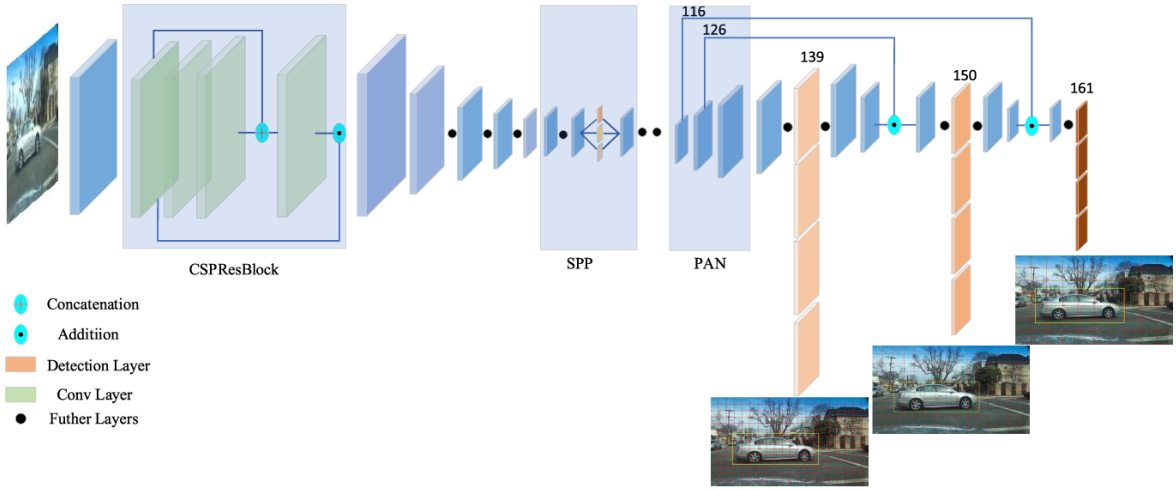


Figure 5.2: A detailed schematic the YOLOv4 architecture [15].

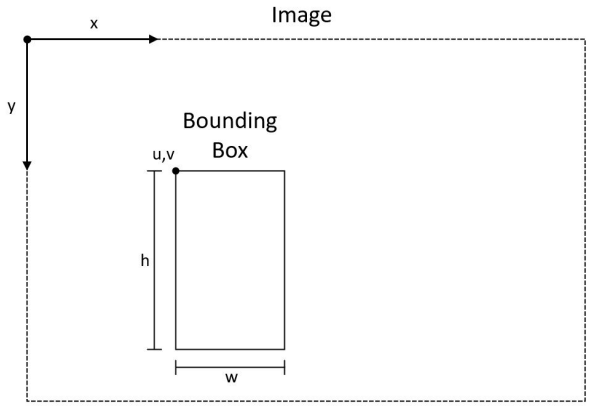


Figure 5.3: Example of the coordinate system of an image and of a single 2D bounding box.

layers. The SSP layers ensure the Head's input contains spatial rich information from the bottom-up stream and semantic rich information from the top-down stream. The PAN layers introduce a shortcut from bottom layers to top layers to prevent the loss of valuable spatial information. This short-circuit makes fine-grain localized information available to top the layers. Finally, the Head (also known as Dense Prediction Block) performs anchor-based object detection. Object proposals are extracted at three levels of granularity (e.g., three different layers), allowing the model a choice of possible combinations to focus on when learning the image detection task at hand. The three layers used to extract the detections are layers 139, 150, and 161, and are depicted in fig. 5.2.

The input of our YOLOv4 implementation is a batch of 5 RGB images and the output is a batch of $5 \times N$ detections in 2D and their confidence scores,

$$\begin{aligned} D^{2D} = &\{\{(u,v,w,h,C)_0^0, ..., (u,v,w,h,C)_{N-1}^0\}, ..., \\ &\{(u,v,w,h,C)_0^{c-1}, ..., (u,v,w,h,C)_{N-1}^{c-1}\}\} \end{aligned} \tag{5.1}$$

To ensure the model can process the image batch in real-time, we reduced the image resolution to 416×416 pixels and converted the model to TensorRT. This conversion optimizes the model, minimizing the GPU memory footprint and accelerating inference by utilizing tensor cores. We utilize a pre-trained model of YOLOv4 trained on the MS COCO dataset[1].

[1] https://github.com/Tianxiaomo/pytorch-YOLOv4

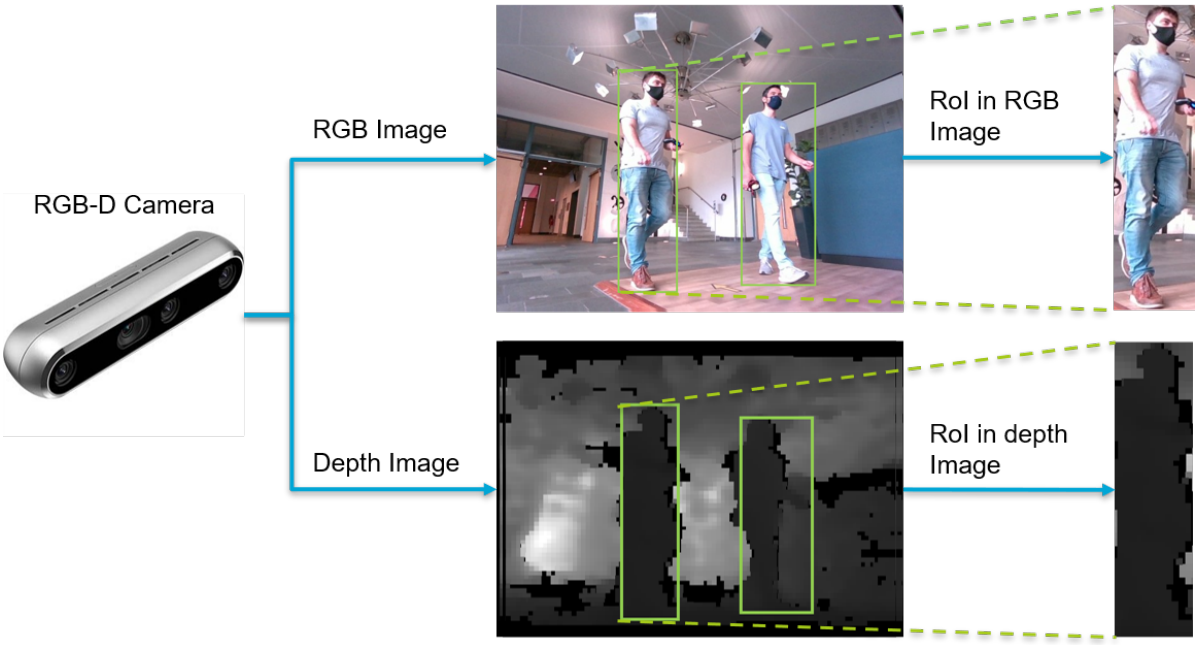


Figure 5.4: The extraction of the RoI in the depth image visualized. The 2D object detector provides bounding boxes in the RGB image. These bounding boxes are used to extract the RoI in the corresponding depth image.

5.2. 2D to 3D Object Conversion

Obtaining accurate depth information is essential in order to translate 2D object detections into 3D. We utilize a standard feature of the RGB-D cameras that aligns the depth images with their corresponding RGB images. This alignment allows us to directly extract the region of interest (RoI) in the depth image by examining the bounding boxes provided by the 2D object detector, shown in fig. 5.4.

5.2.1. Depth Estimation

To estimate the depth of a pedestrian, we utilize the 2D bounding boxes provided by the 2D object detector to extract the RoI in the depth image. Since 2D bounding boxes do not provide information on exactly which pixels belong to the object, selecting the relevant depth pixels to extract the object's distance is a challenging procedure. Our approach to addressing this problem is to exploit the fact that the YOLOv4 network performs bounding box regression to generate 2D bounding boxes. This regression network aims to trim the 2D bounding boxes until they fit tightly around the object. As a result of this property, pixels belonging to the object are expected to cover the RoI's entire vertical and horizontal range.

We examined several approaches to obtain the pedestrian's depth. In essence, we can formulate the challenge of extracting the depth as a pixel clustering problem. We tested several widely used clustering techniques, such as DBSCAN [30], K-Means [69], Mean Shift [34] and Region Growth [25]. Although these methods appeared to be performing reasonably well, real-time implementation proved infeasible due to their computational complexity. Furthermore, we tested a method that would create a histogram of all depth values in the RoI and select the pedestrian's depth based on peaks in the histogram. We examined several different rules to filter the peaks and extract the correct depth value. Although this method performed well in unpopulated scenarios, performance dropped considerably when transitioned to dense scenarios. Hence, we could not realize robust depth estimation using this method.

We propose a depth estimation method similar to Iloie et al. [44]. By creating an u-disparity and v-disparity [53], we can extract a distance estimation from the RoI. Please note that, in the original paper, the u-disparity and v-disparity images are computed based on disparity images. Since our RGB-D cameras output depth images instead of disparity images, our method uses depth images instead. We

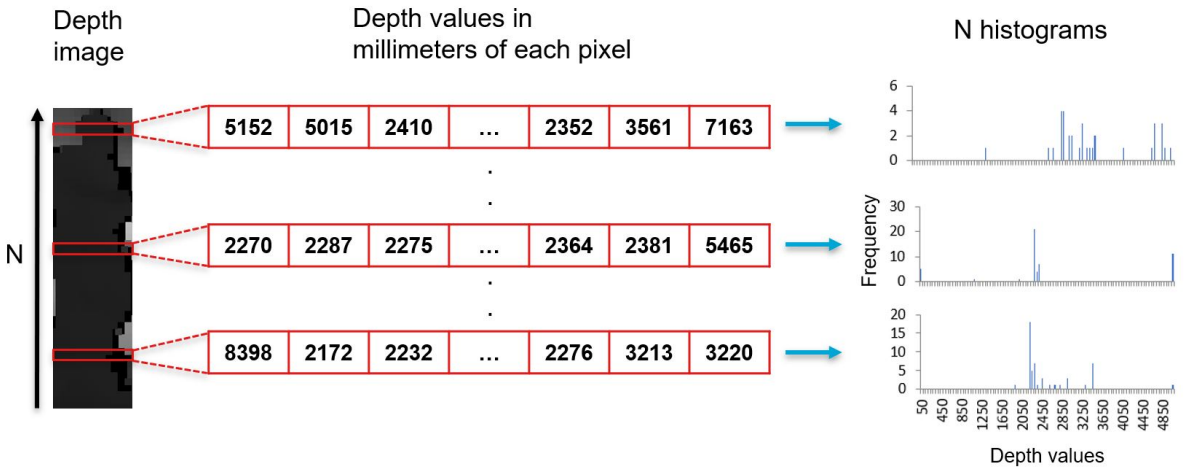


Figure 5.5: First step visualized of the creation of a v-depth image. We iterate through each of the N rows of the depth image and extract their depth values (in millimeters). Subsequently, we create an individual histogram of these rows, resulting in N unique histograms per depth image.

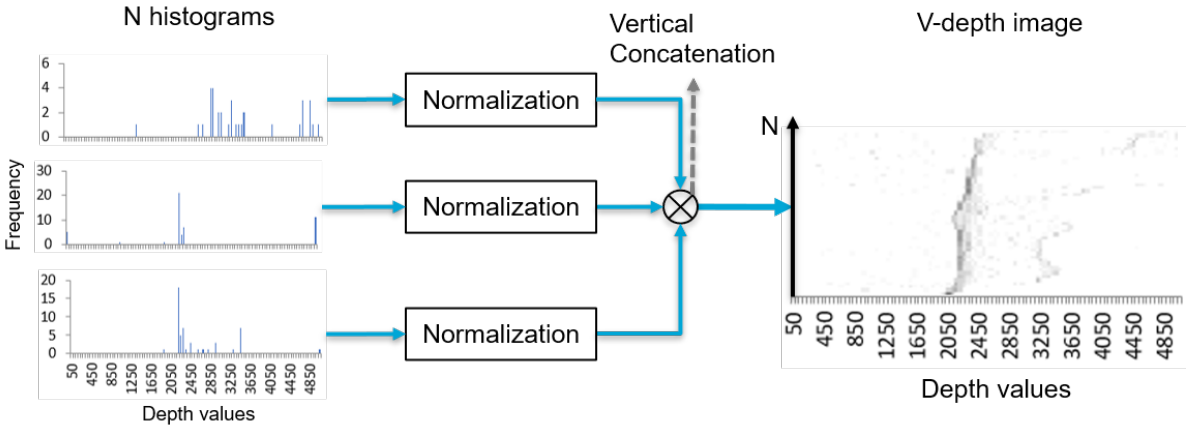


Figure 5.6: The second step visualized of the creation of a v-depth image. Each of the N histograms is normalized and vertically concatenated, resulting in the v-depth image. Note that each of the N rows in the v-depth image corresponds to one of the N normalized histograms.

follow the same approach to compute u- and v-disparity images, but we refer to them as u-depth and v-depth images. For clarity, we will explain our approach using only the v-depth image as an example.

Using the extracted RoI from the depth image, we generate the v-depth image by creating N histograms of depth values for each of the N rows in the RoI, as depicted in fig. 5.5. To generate the histograms, we use 100 equally spaced bins in the range between 0 and 5 meters. The N histograms are normalized and concatenated vertically to form the v-depth image shown in fig. 5.6. Each of the N rows in this v-depth image corresponds to a single normalized histogram. The pixel intensities in these u- and v-depth images correspond to the frequency distribution of the bins in the histograms.

Because our 2D object detector regresses the bounding boxes until they tightly fit around the pedestrian, we can assume the object spans the entire vertical and horizontal range in the RoI. This means that the pedestrian will appear as a line reaching from top to bottom in the v-depth image, as can be seen in the V-depth image in fig. 5.6. By computing the v-depth image, we reduced the problem of extracting the object's depth to the search of a pixel line satisfying our assumption.

Using the Hough Transform [29], we efficiently fit lines in the v-depth image, see fig. 5.7. Lines are filtered based on a minimum score threshold and length. The line that fits these criteria best is selected,

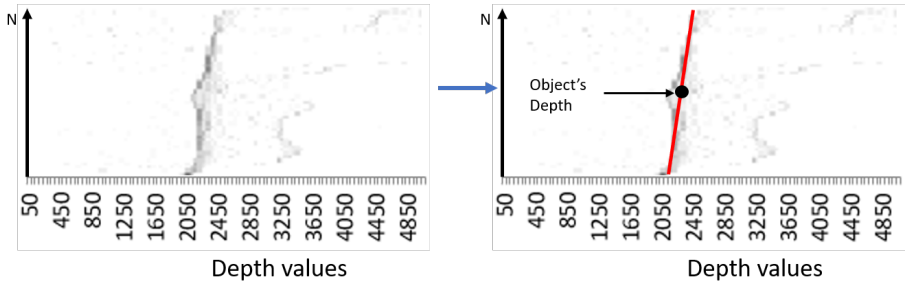


Figure 5.7: Visualization of the best fitting line provided by the Hough Transform. The object's depth is defined as the depth value at the center of the line.

and we extract the depth value at the center of the line as the object's depth value.

The method to generate the u-depth image is similar to that of the v-depth image and only requires column-wise instead of row-wise extraction of the RoI's depth values. Since the u- and v-depth images both provide one depth proposal, we employ a simple rule-based decision scheme to obtain the pedestrian's final depth. If both proposed depths are within a certain distance of each other, we use the mean value. However, if the proposed depths of the two methods are inconsistent, the closest depth estimate is selected. Although selecting the closest depth is in some cases incorrect, the robot should avoid collision with the detected object nonetheless, making this the safest approach.

5.2.2. 2D to 3D Bounding Box

Since the estimated depth value represents the distance towards the surface of the object and not the object's center, we add a constant value of l_{offset} meters to the estimated depth value to compensate for the offset. Subsequently, the center of the 3D bounding box can be computed from the center of the 2D bounding box and the estimated depth using the intrinsic camera matrix. This matrix is unique for each camera and describes the relation between 3D camera coordinates and 2D pixel coordinates. Equation (5.2) projects 3D points onto a 2D image plane:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \alpha & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{5.2}$$

In which f_x and f_y denote the focal length in pixel coordinates, c_x and c_y denote the offset of the principal points in pixel coordinates relative to the top left corner of the image, α denotes the skew factor that accounts for a shear distortion, u and v are the pixel coordinates in the image and X and Y are the 3D point coordinates in the camera frame.

This formula can be rewritten into the following two equations to extract the X and Y center point coordinates belonging to the depth value Z:

$$X = ((u - c_x) \cdot Z)/f_x \tag{5.3}$$

$$Y = ((v - c_y) \cdot Z)/f_y \tag{5.4}$$

We use the same two equations to compute the height and width of the 3D bounding box. By inserting the pixel coordinates of the corners of the 2D bounding box, and for convenience assuming these corners are at the same distance as the center point, we can calculate the 3D locations of these corners. The width and height can then be calculated using eq. (5.5) and eq. (5.6):

$$w = X_{tl} - X_{tr} \tag{5.5}$$

$$h = Y_{bl} - Y_{tl} \tag{5.6}$$

X_{tl} and X_{tr} denote the Cartesian X coordinate of the top left and top right corner, respectively. Y_{bl} and Y_{tl} denote the Cartesian Y coordinate of the bottom left and top left corner, respectively.

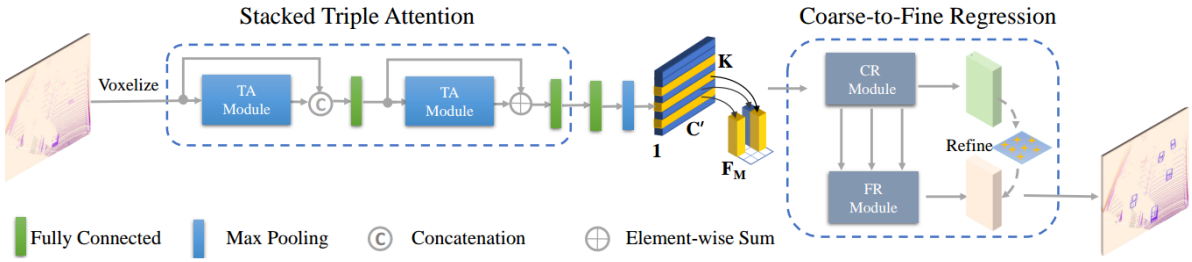


Figure 5.8: A detailed schematic the TANet architecture [68].

The depth cameras and the LiDAR sensor have different vertical FOVs, resulting in nearby pedestrians experiencing different amounts of truncation. Since the performance evaluation is heavily dependant on the IoU 3D, we model each 3D bounding box to have a fixed height of h_{fix} meters. The orientation θ is challenging to extract from 2D images; hence we model the angle θ as facing the RGB-D camera. Furthermore, reliable and accurate length extraction from the depth images proved difficult; hence we use constant lengths of l_{fix} meters to model the 3D bounding boxes.

Whereas the depth accuracy of LiDAR sensors is relatively unaffected by distance, the depth measurements error of RGB-D cameras increases quadratically with the measured depth, as shown in eq. (5.7).

$$\Delta z = \frac{z^2}{b \cdot f} \cdot \Delta D \tag{5.7}$$

In which Δz is the depth measurement error, z^2 is the depth, b is the distance between the two stereo cameras (e.g. baseline), f is the focal length in pixels and ΔD is the disparity error in pixels.

To ensure the predicted 3D bounding boxes still overlap the ground-truth 3D bounding boxes, we chose to limit the maximum depth estimation range of the RGB-D cameras to $range_{max}$ meters. This parameter was selected according to the datasheet of the sensor.

5.2.3. Camera to LiDAR Transformation

As stated in section 3.5.1, SLAM relies on the LiDAR's point cloud to compute the robot's pose in world coordinates. To create a clear top-down reference frame tree, we decided to calibrate the RGB-D cameras' reference frames relative to the LiDAR's reference frame. It is irrelevant which frame we fuse the 3D detections as long as we use the same reference frame. Since our 3D object tracker requires detections in world coordinates, it is straightforward that the fusion of the 3D detections takes place in the LiDAR's coordinate system or the world coordinate system. It is of no significance whether fusion is performed in the LiDAR or world coordinate system; hence we decided to fuse the 3D detections in the LiDAR's coordinate system and later transform them to the world coordinate system.

The fusion and filtering scheme described in section 5.4 fuses the 3D detections from the *3D Detection* and *2D to 3D Object Conversion* methods in the LiDAR's coordinate system. Therefore, we need to transform the camera-based 3D detections of each camera from their camera's reference frame to the LiDAR's reference frame. Using the extrinsic sensor calibration method from Dhall et al. [26], we determine the correct transformation matrices to transform points between various coordinate systems. Finally, the tf package inside ROS uses these matrices to deal with all the transformations.

5.3. 3D Detection

We integrate the TANet[68] algorithm in this framework as our 3D object detector[2]. TANet is a state-of-the-art method to obtain 3D object detections from LiDAR point cloud data. We have selected this method because it has a relatively fast inference time, is more robust to noise than other 3D object detectors. At the time of writing, this method was the best performing open-source algorithm listed on the KITTI 3D Object Detection Benchmark. Similar to the popular 3D object detectors [127][114][54],

[2]https://github.com/happinesslz/TANet

TANet first equally divides the point clouds into a rectangular voxel grid consisting of a predefined set of voxels. Then, the Stacked Triple Attention component separately processes each voxel to obtain a more discriminative feature representation. The first Triple Attention (TA) module directly operates on the original point cloud features and provides higher-dimensional features for the second TA module. Each TA module takes channel-wise, point-wise, and voxel-wise attention into consideration jointly.

1. The point-wise attention is designed for describing the spatial correlation among the points inside each voxel. It performs a max-pooling operation to aggregate point features across the channel-wise dimensions, resulting in its point-wise response.

2. The channel-wise attention is similar to the point-wise attention. However, it performs a max-pooling operation to aggregate the channel features across their point-wise dimensions, which obtains the channel-wise response, indicating the importance of the feature channels in each voxel.

3. The voxel-wise attention averages the coordinates of all points inside each voxel to obtain the voxel center. This technique provides accurate location and dimensional information of the voxel.

Finally, the Coarse-to-Fine regression is employed to generate the final 3D bounding boxes. This component consists of a Course Regression (CR) and a Fine Regression (FR) module. The CR module is based on [127]. It utilizes the feature maps from the Stacked Triple Attention module to perform classification and regression, which results in coarse 3D bounding boxes. The FR module regards these coarse boxes as the new anchors to fine-tune the regression and classification of the 3D bounding boxes.

TANet employs a multi-task loss function to jointly optimizing the CR and FR module. The loss function is defined as follows:

$$\begin{aligned} L_{\text{total}} &= \alpha L_{\text{cls}}^{\mathcal{C}} + \beta \frac{1}{N_{\text{pos}}^{\mathcal{C}}} \sum L_{\text{reg}}^{\mathcal{C}} \left(\Delta_{\mathbf{p}}^{\mathcal{C}}, \Delta_{\text{g}}^{\mathcal{C}}\right) + \\ & \lambda \left\{ \alpha L_{\text{cls}}^{\mathcal{R}} + \beta \frac{1}{N_{\text{pos}}^{\mathcal{R}}} \sum L_{\text{reg}}^{\mathcal{R}} \left(\Delta_{p}^{\mathcal{R}}, \Delta_{g}^{\mathcal{R}}\right) \right\} \end{aligned} \tag{5.8}$$

In which $N_{pos}^{\mathcal{C}}$ and $N_{pos}^{\mathcal{R}}$ represent the number of positive anchors in the CR module and FR module, respectively. α and β stand for the balance weights for the classification loss and the regression loss, respectively. λ is used to balance the weight for the CR module and FR module. The superscript $\mathcal{C}$ and $\mathcal{R}$ represent the CR module and FR module, respectively.

The offsets of the bounding box regression between prior anchor a and the ground-truth box g can be computed as:

$$\begin{aligned} &\Delta_x^g = \frac{x_g - x_a}{d_a}, \Delta_y^g = \frac{y_g - y_a}{d_a}, \Delta_z^g = \frac{z_g - z_a}{h_a} \\ &\Delta_w^g = \log\left(\frac{w_g}{w_a}\right), \Delta_l^g = \log\left(\frac{l_g}{l_a}\right), \Delta_h^g = \log\left(\frac{h_g}{h_a}\right) \\ &\Delta_\theta^g = \theta_g - \theta_a \end{aligned} \tag{5.9}$$

In which $d_a = \sqrt{(w_a)^2 + (l_a)^2}$. For convenience, the residual vector $\Delta_g = (\Delta_x^g, \Delta_y^g, \Delta_z^g, \Delta_w^g, \Delta_l^g, \Delta_h^g, \Delta_\theta^g)$ is defined as the regression ground-truth. Similarly, $\Delta_p = (\Delta_x^p, \Delta_y^p, \Delta_z^p, \Delta_w^p, \Delta_l^p, \Delta_h^p, \Delta_\theta^p)$ represents the offsets between prior anchors and the predicted 3D bounding boxes. TANet uses SmoothL1 [38] as their 3D bounding box regression loss L_{reg} and the Focal Loss [62] as classification loss L_{cls} .
The input of our TANet implementation is the 3D point cloud. The output is a set of N detections in 3D LiDAR coordinates (fig. 6.8) and their corresponding confidence scores:

$$D^{3D} = \{(x, y, z, w, h, l, \theta, C)_0, ..., (x, y, z, w, h, l, \theta, C)_{N-1}\} \tag{5.10}$$

We use a similar training strategy as proposed by the authors of the TANet [68] 3D object detector. However, we needed to modify several settings to adapt the model to suit our mobile robot. The first alteration is the number of classes. In contrast to TANet's goal to detect pedestrians, cyclists, and cars, our goal is only to detect pedestrians. Since TANet performs anchor-based classification for each class separately, excluding the cyclist and car classes from training will decrease the number of operations and therefore decrease inference time. Secondly, we increase the voxel dimensions to

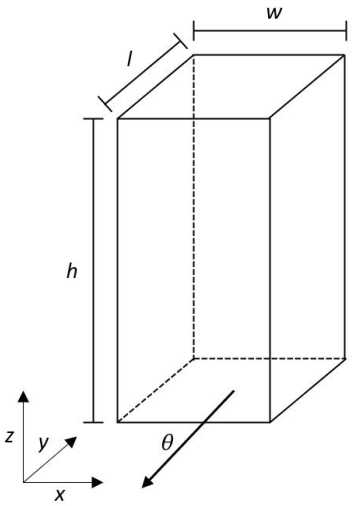


Figure 5.9: Visualization of a single 3D bounding box.

vox_{dim} to reduce the GPU memory footprint of the network and inference time. Training at the original voxel dimensions is currently infeasible since it demands too much GPU memory. Thirdly, we lower the anchor boxes to match our mobile robot. TANet's Region Proposal Network (RPN) employs predefined anchor boxes to provide the CR module's Regions of Interest (RoI). Since we mounted our LiDAR at a different height than the one used in the KITTI dataset, we need to modify the height of our anchor boxes to compensate for the height difference. Lastly, the design requirements of our framework demand a minimum detection range of 10 meters. In order to satisfy the real-time detection criteria, we limit the search area of the voxel grid to this minimum range of $\pm$ 10 meters in the X and Y direction.

5.4. Sensor Fusion & Filtering Scheme

The fusion algorithm combines the 3D detections from the 3D object detector and the five RGB-D cameras to construct an accurate representation of surrounding pedestrians. We employ a series of filtering steps to achieve robust fusion. The following sub-sections explain the filtering steps in detail. Algorithm 1 supports these sections by presenting a general overview of the filtering steps.

5.4.1. Discard Duplicate Camera Detections

Due to the overlapping regions in the cameras' FOV, the same pedestrian can be detected twice in neighboring cameras, as seen in fig. 5.10 (a) and fig. 5.10 (b). Figure 5.10 (c) depicts the point cloud corresponding to the duplicate pedestrian detection from fig. 5.10 (a) and fig. 5.10 (b) and its assigned 3D bounding boxes. We obtain the green-colored 3D bounding box by converting the 2D bounding box in fig. 5.10 (b) and the large blue 3D bounding box from the pedestrian detection in fig. 5.10 (a). Since we need to assign a single 3D bounding box to each pedestrian, duplicate detections must be filtered out. A common practice to identify duplicate 3D bounding boxes is to determine their volumetric overlap (i.e., IoU 3D, further discussed in section 6.1.2). However, we noticed that in certain scenarios, this criterion could not correctly identify duplicate detections. One of these scenarios occurs when a pedestrian is fully visible in one image and only partially visible in the neighboring image. Because we derive the 3D bounding box's width from the 2D bounding box's width, heavily occluded or truncated pedestrians receive a relatively small width value. This results in one larger and one smaller yet elongated 3D bounding box. When the larger 3D bounding box overlaps or completely embodies this smaller bounding box, filtering out these false positives using IoU 3D is impractical since it would result in a relatively low value. A better metric would be to use the volumetric fraction (V_{frac}) each of the volumes of the 3D bounding boxes (V_{bbox}) contributes to the intersecting volume (V_{inter}).

$$V_{fraq} = \frac{V_{bbox}}{V_{inter}} \tag{5.11}$$

If the V_{frac} exceeds a certain threshold $thresh_{vol}$, the 3D bounding box with the least volume is dis-

Algorithm 1 Sensor Fusion & Filtering Scheme

Discard Duplicate Camera Detections :
Load 3D camera detections: DC
for each pair $\langle DC_i, DC_j \rangle$ $(i < j)$ **do**
  Compute 3D IoU between each combination of detections and extract the intersecting volume: $V_{inter,i,j}$
  Compute volumes of detection i and j: $V_{bbox,i}$, $vV_{bbox,j}$
  if $(V_{bbox,i}/V_{inter,i,j})$ or $(V_{bbox,j}/V_{inter,i,j})) > thresh_{vol})$ **then**
    Remove duplicate detection with lowest volume from DC
  end if
end for

Discard Colliding False-Positive :
Load 3D camera confidences: CC
for each pair $\langle DC_i, DC_j \rangle$ $(i < j)$ **do**
  Compute distance between the center points of the two detections DC_i and DC_j: dis
  if $(dis < r_{dist})$ **then**
    Remove detection with lowest confidence CC from DC
  end if
end for

Combining Camera & LiDAR Detections :
Load LiDAR detections: DL
Load LiDAR confidences: CL
Initialize empty list: $List$
for $i = 0$ to length(D_l) **do**
  Compute distance between detection and robot: dis
  if $(dis \geq 3)$ and $(dis < 10)$ and $(CLi > thresh_{LiDAR})$ **then**
    Append detection to $List$
  end if
end for

for $i = 0$ to length(DC) **do**
  for $j = 0$ to length($List$) **do**
    Compute distance between the center points of the accepted detection $List_i$ and candidate detection DC_j: dis
    if $(dis < r_{dist})$ **then**
      Do not append detection DC_j to $List$
    end if
  end for
end for

Transformation to World Coordinates :
Retrieve transformation matrix between LiDAR coordinate system and world coordinate system: M_{tf}
Transform detections in $List$ to world coordinates using M_{tf}

Figure 5.10: Visualizations of false-positive detections. Images (a) and (b) are captured from two neighboring cameras at the same time instant. Green-colored bounding boxes from the 2D object detector are drawn in (a) the left camera image and (b) the right camera image. Figure (c) depicts three 2D to 3D converted bounding boxes assigned to the same pedestrian. After filtering, the green 3D bounding box is the accepted detection, and the two blue 3D bounding boxes are discarded.

carded.

5.4.2. Discard Colliding False-Positive

After this first filtering step, we identified two additional situations that resulted in false positives. First, the disparity images provided by the RGB-D camera tend to output noisy depth measurements around the outer contours of pedestrians. In situations where the 2D object detector inaccurately estimates the 2D bounding box dimensions of a pedestrian partially occluded by another pedestrian, these noisy pixels could result in the wrong depth estimates of 3D objects. This results in the partially occluded pedestrian's 3D bounding box being placed too close to the camera (at the other pedestrian's distance). An example of this phenomenon can be seen in fig. 5.10 (a). The bounding box corresponding to the pedestrian in front completely embodies the bounding box of the partially occluded pedestrian at the back. Due to the noisy depth pixels, the bounding box of the occluded pedestrian is placed at the location of the nearby pedestrian. This smaller blue-colored bounding box in fig. 5.10 (c) corresponds to the partially occluded pedestrian. Since only a fraction of the pedestrian is visible in the image, it often results in a relatively small 2D and 3D bounding box. The previously used method of computing the V_{fraq} does not identify all cases since these two bounding boxes have minor to none volumetric intersection. The second situation occurs due to the pentagonal arrangement of the RGB-D cameras. The FOV of these cameras does not overlap within approximately the first meter, possibly resulting in two detections of the same nearby pedestrian. Since the FOV does not overlap, most of the time, the resulting 3D bounding boxes will not overlap either. This renders close-range duplicate identification by volumetric overlap infeasible.

To filter out these false positives, we check for collisions between the multiple 3D bounding boxes. If the distance between the centers of two bounding boxes is smaller than r_{dist} meters, we assume these two bounding boxes are too close and thus colliding. We use the confidence scores to determine which one of the two colliding boxes is incorrect. Since 2D object detectors typically struggle with detecting occluded objects, we presume a partially occluded object negatively influences its confidence score. This results in the partially occluded pedestrian having a lower confidence score than a clearly visible pedestrian; hence, we discard the 3D bounding box with the lowest confidence score.

5.4.3. Combining Camera & LiDAR Detections

When fusing the LiDAR and camera detections to form our final detection list, duplicate detections can occur since we process the LiDAR and camera detections independently, as seen in fig. 5.11. Before fusing the detections, these duplicates must be matched and discarded. Because the LiDAR detections have more accurate 3D locations than the camera detections, we first append the LiDAR detections to the final detection list. Suppose the distance to the detection is in the far range (e.g., 3 to 10 meters), and the detection confidence is higher than $thresh_{LiDAR}$, we append the detection to the list. We discard the close range (e.g., 0 to 3 meters) 3D LiDAR detections because the narrow FOV of the LiDAR renders it challenging to classify pedestrians. After filtering the LiDAR detections, we start appending the camera detections. Before adding a candidate detection, we perform another duplicate

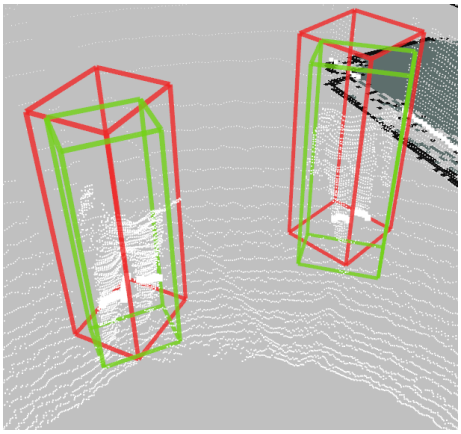


Figure 5.11: Visualization of duplicate LiDAR and camera detection of pedestrians. The green-colored and red-colored 3D bounding boxes depict the camera-based 3D and the LiDAR-based detections, respectively.

detection check. We compute the distance to all previously accepted detections in the detection list for each candidate camera detection. If any one of the computed distances is smaller than r_{dist} meters, we discard the candidate camera detection.

5.4.4. Transformation to World Coordinates

Since 3D object tracking is performed in world frame coordinates, the detections need to be transformed from the LiDAR coordinate frame to the real-world Cartesian space. We again use the ROS tf package to transform the 3D detections to the appropriate coordinate frame.

5.5. 3D Tracking

We employ the state-of-the-art AB3DMOT [111] method as our 3D object tracker[3]. The main reason we have selected AB3DMOT is due to its simplicity. Most state-of-the-art object trackers require a GPU to associate complex appearance descriptors. However, AB3DMOT only utilizes the CPU for its data association and filtering. Moreover, the performance of AB3DMOT is on par with most of its competitors. Integrating the CPU-based AB3DMOT relieves the graphics card, which is already running two object detectors.

The goal of the 3D object tracker is to associate K fused 3D detections at time t to L previously tracked objects at time $t-1$. The AB3DMOT algorithm uses a Kalman filter [48], based on a constant velocity model, to predict the state vector of a tracked object at time t using its state vector at time $t-1$. Subsequently, a 3D bounding box is extracted from the object's predicted state vector. Data association at time t is then performed between the fused 3D detection and the predicted 3D bounding boxes.

A cost matrix $M_c \in \mathbb{R}^{K \times L}$ is introduced to find the optimal combination of associations. Element $M_{c,ij}$ represents the cost associated with matching prediction i to detection j and is defined as the 3D IoU between both 3D bounding boxes. It is calculated for each combination of detections and predictions. Obtaining the optimal combination can then be regarded as a linear sum assignment problem (e.g., Hungarian Algorithm [52]). This algorithm aims to find a complete assignment of predictions and detections at a minimal cost. Matches with a 3D IoU lower than 0.25 are filtered out. This results in two sets of detections, one set of matched detections, and one set of unmatched detections.

A new track, containing an independent Kalman filter, is initialized for each unmatched detection. An inactive track is created (not immediately part of the tracker's output) to prevent an overflow of false-positive tracks. The track is promoted to active after n_{init} number of consecutive matches. A track is terminated if there has been no association for n_{term} consecutive frames to account for objects leaving the scene.

[3]https://github.com/xinshuoweng/AB3DMOT

Finally, in the update step of the Kalman filter, the Kalman Gain is computed for each new measurement. The Kalman Gain determines how much the input measurement (e.g., fused 3D detection) will influence the system state estimate. This procedure ensures the stability of the estimated state vector when the measurement is noisy by relying more on its own prediction. The effect of the Kalman Gain results in smoothed trajectories of the tracked pedestrians.

Since the AB3DMOT tracker is designed to track objects in the KITTI dataset (which includes pedestrians), we use the default parameters for the internal system and noise matrices. Our implementation of the AB3DMOT method expects a set of K fused 3D bounding boxes as input and outputs a set of L 3D bounding boxes in combination with their track IDs and velocities.

6

Experiments

This section presents real-world experiments to validate our proposed approach. In section 6.1, we explain the metrics used to validate the performance of our detection and tracking framework. Section 6.2 lists the experimental settings used throughout our pipeline. Our experiments are conducted in the form of an ablation study to gain insight into the effectiveness of the various sub-modules in the framework. We explain our three experiments and provide the results in section 6.4, section 6.5, and section 6.6.

6.1. Evaluation Metrics

Our three experiments each require a different set of metrics. We first explain a set of basic metrics used in all experiments, after which section 6.1.6 and section 6.1.7 explains the metrics used to evaluate the 2D and 3D bounding box detections, respectively.

6.1.1. Confusion Matrix

The core of most evaluation metrics relies on the basic principles of the confusion matrix. For each detection there are four possible outcomes, True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN). Their mutual relationship can be seen in table 6.1.

		Actual class	
		Positive	Negative
Predicted class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 6.1: Example of confusion matrix in abstract form

6.1.2. Intersection over Union

We employ the Intersection over Union (IoU) metric to assess the correctness of each detected pedestrian's 2D or 3D bounding box. This will divide the detections into a set of TP and FP detections. The IoU is defined as the intersection (or overlap) of a detected bounding box (bbox_{det}) and ground-truth bounding box (bbox_{gt}) divided by the union of the detected and ground-truth bounding box as seen in eq. (6.1), eq. (6.2), and fig. 6.1.

$$\text{IoU 2D} = \frac{\text{area}(\text{bbox}_{\text{det}} \cap \text{bbox}_{\text{gt}})}{\text{area}(\text{bbox}_{\text{det}} \cup \text{bbox}_{\text{gt}})} \tag{6.1}$$

$$\text{IoU 3D} = \frac{\text{volume}(\text{bbox}_{\text{det}} \cap \text{bbox}_{\text{gt}})}{\text{volume}(\text{bbox}_{\text{det}} \cup \text{bbox}_{\text{gt}})} \tag{6.2}$$

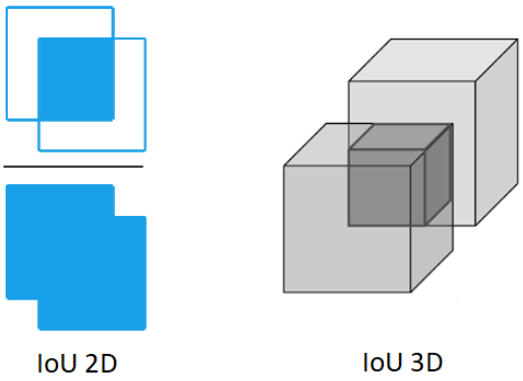


Figure 6.1: Visualization of IoU 2D [1] and IoU 3D [27].

6.1.3. Precision & Recall

The precision of an algorithm measures the accuracy of the predictions. It is defined as the ratio of TP to the total number of detections (e.g., TP + FP).

$$\text{Precision (p)} = \frac{TP}{TP + FP} \tag{6.3}$$

The recall, also know as true positive rate or sensitivity, is a measure on how many true positives are detected. It is defined as the ratio of TP to the total number of ground-truth detections (e.g., TP + FN).

$$\text{Recall (r)} = \frac{TP}{TP + FN} \tag{6.4}$$

To calculate the precision or recall, we need to determine the elements in the confusion matrix. To assess whether a detection is a TP, we employ the IoU 2D or IoU 3D metric. If the IoU exceeds a certain threshold, the detection is labeled as TP. In 2D object detections, standard IoU thresholds are 0.50, 0.75, and a combined threshold of 0.50:95 (precision and recall averaged over 10 IoU thresholds). In 3D object detection, frequently used IoU thresholds are 0.25 and 0.50.

6.1.4. Average Precision

The Average Precision is defined as the area underneath the precision-recall curve. A higher AP implies a better performance of the detector. As the name suggests, the precision-recall curve is the relation between precision and recall. Detections of a single class are gathered, placed in a list, and sorted in descending order according to their confidence value. Iteratively the precision and recall values are calculated and placed in the precision-recall curve. Because the general definition of the AP is susceptible to small variations in the ranking of the prediction values, it is common practice to use the 40-point interpolated AP instead. It filters out valleys in the precision-recall curve by replacing each precision value with the precision value of the first local maxima to the right, as shown in fig. 6.2. Mathematically this can be described as eq. (6.5). The result is a precision-recall curve that decreases monotonically without any local minima. The interpolated precision (p_{interp} is calculated at 40 equally spaced recall values. The AP is then defined as the average of the 40 interpolated precision values, eq. (6.6).

$$p_{interp}(r) = \max_{\tilde{r} > r} p(\tilde{r}) \tag{6.5}$$

With p_{interp} being the interpolated precision, r being the corresponding recall level, $p(\tilde{r})$ being the precision of any recall level greater than r.

$$\text{AP} = \frac{1}{40} \cdot \sum_{r \in \left\{\frac{1}{40}, \frac{2}{40}, \ldots, 1\right\}} p_{interp}(r) \tag{6.6}$$

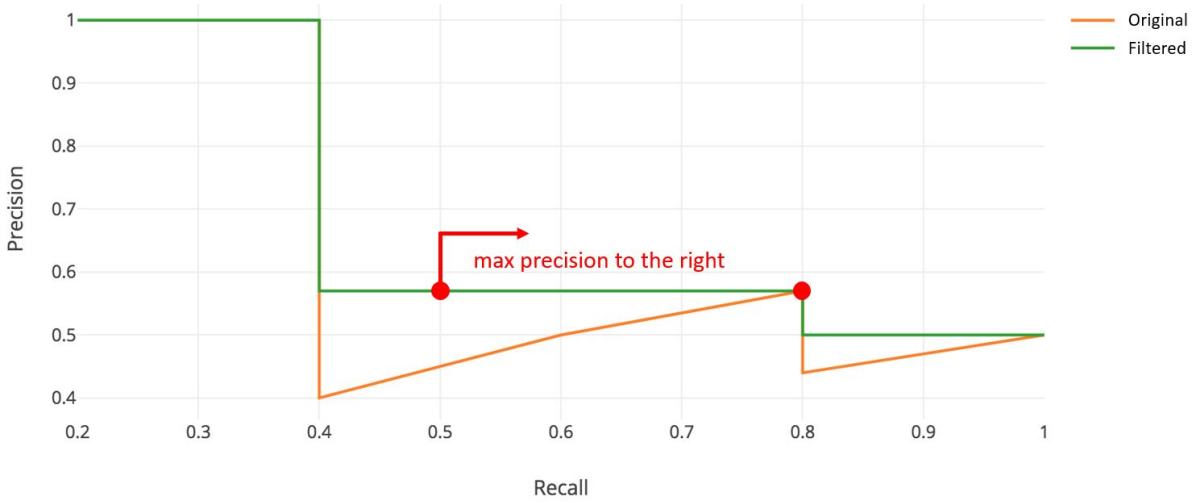


Figure 6.2: Example of an original Precision Recall curve and it's interpolated (filtered) version. [47]

6.1.5. Tracking

Several metrics are used in the field of Multi-Object Tracking (MOT). Ciaparrone et al. [24] state that the most relevant tracking metrics are the classical metrics defined by Wu et al. [112], the CLEAR MOT metrics [9], and the ID [86] metrics. Both the ID and CLEAR MOT metrics are used to quantify the performance of the MOT algorithms and build upon the classical tracking metrics. Since we perform an ablation study and do not benchmark our model against other methods, we adopt the classical metrics to analyze the qualitative tracking performance and explain some of the frequently occurring tracking errors.

The classical metrics defined by Wu et al. [112] identifies three common tracking errors:

- **Fragments (FRAG)**: A trajectory fragment occurs when the tracker temporarily loses track of an object.
- **False Trajectories**: A false trajectory (or False Alarm) occurs when a trajectory does not correspond to a real object (e.g., to a ground-truth trajectory).
- **ID switches (IDS)**: An ID switch occurs when the object is correctly tracked, but the associated ID of the object is mistakenly changed.

Figure 6.3 provides clear examples of common tracking errors. In our qualitative performance analysis, we adopt the Trajectory Fragments, False Trajectories, and ID switches metrics.

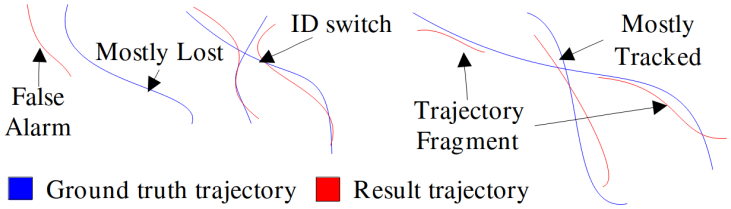


Figure 6.3: A visualization of the classical tracking metrics [112].

6.1.6. 2D Bounding Box Detections

To validate the performance of our optimized 2D object detector, we adopt the extended Average Precision (AP) metric from the MS COCO challenge [63]. It uses the IoU 2D metric to assess the quality of the 2D bounding boxes. The MS COCO based AP metric evaluates the performance at several IoU

thresholds and object scales to gain insight into the performance at various difficulty levels. Furthermore, the inference time is measured to examine the real-time capabilities of the 2D object detector. We investigate the inference time while propagating a single image versus propagating a batch of five images. Comparing these values reveals the significance of utilizing the TensorRT architecture. To summarize, we use the following eight metrics:

- **AP$^{\text{IoU=0.50:0.95}}$**: AP, averaged over 10 IoU thresholds: $\{0.50, 0.55, ..., 0.95\}$.
- **AP$^{\text{IoU=.50}}$**: AP at IoU=0.50 [31].
- **AP$^{\text{IoU=.75}}$** AP at IoU=0.75.
- **AP$^{\text{small}}$**: AP for small objects: area $< 32^2$, averaged over 10 IoU thresholds: $\{0.50, 0.55, ..., 0.95\}$.
- **AP$^{\text{medium}}$**: AP for medium objects: $32^2 < \text{area} < 96^2$, averaged over 10 IoU thresholds: $\{0.50, 0.55, ..., 0.95\}$.
- **AP$^{\text{large}}$**: AP for large objects: area $> 96^2$, averaged over 10 IoU thresholds: $\{0.50, 0.55, ..., 0.95\}$.
- **Inference Time$^{\text{batch=1}}$**: Inference time in milliseconds of the network at a batch of one image.
- **Inference Time$^{\text{batch=5}}$**: Inference time in milliseconds of the network at a batch of five images.

6.1.7. 3D Bounding Box Detections

The standard approach to evaluate 3D detections is by adopting the metrics from the NuScenes Object Detection Benchmark [14]. This AP 3D metric extends the previously discussed AP to Cartesian space by utilizing the IoU 3D. However, after performing an evaluation using the AP 3D metric, the obtained results were doubtful since the fused detection experienced a significant drop in performance. This performance drop is the result of the incomparability of the 2D and 3D detectors' confidence scores. By definition, the AP 3D metric sorts all detections based on their confidence values before computing the precision-recall curve. Since the camera detections typically have a very high confidence score in contrast to our LiDAR detections, the sorted list would have a preference to camera detections. However, converting our 2D detections into Cartesian space using the *2D to 3D Conversion* module introduces an uncertainty that increases over distance. Since we do not adjust the confidence scores during conversion, using the original confidence scores provided by the 2D object detector is thus no longer reliable. Hence, evaluating the fused detections using the AP 3D metric is ambiguous.

We examined a method that would adjust the confidence scores of the camera detections based on the object's distance. The accuracy of the 3D bounding box strongly depends on the accuracy of the depth estimation. Since the RGB-D camera's depth error increases with distance, coupling the confidence score to the object's distance could enhance the reliability of the confidence score. Preliminary results indicated a more plausible AP score. However, since we observed that the AP was very susceptible to the method we used to compute the new confidence score, we decided not to use this metric.

Because the confidence scores of our 2D and 3D detector are incomparable, computing the 3D AP for these detections and comparing them is ambiguous. Therefore, we chose to adopt the precision and recall metrics instead.

We evaluate both precision and recall at the 0.25 and 0.50 IoU 3D thresholds to examine the ability of the detectors and tracker to output accurate bounding box locations and dimensions. Finally, we measure the inference time (or runtime) in milliseconds of the modules while processing the detections from a single frame to assess the real-time capabilities of these modules.

6.2. Experimental Settings

In our experiments, we will use the following parameters. Although the RGB-D cameras can capture RGB and depth images at a resolution of 1280×800 and 1280×720 pixels, respectively, we use a resolution of 640×480 instead. This lower resolution is sufficient for our 2D object detector and

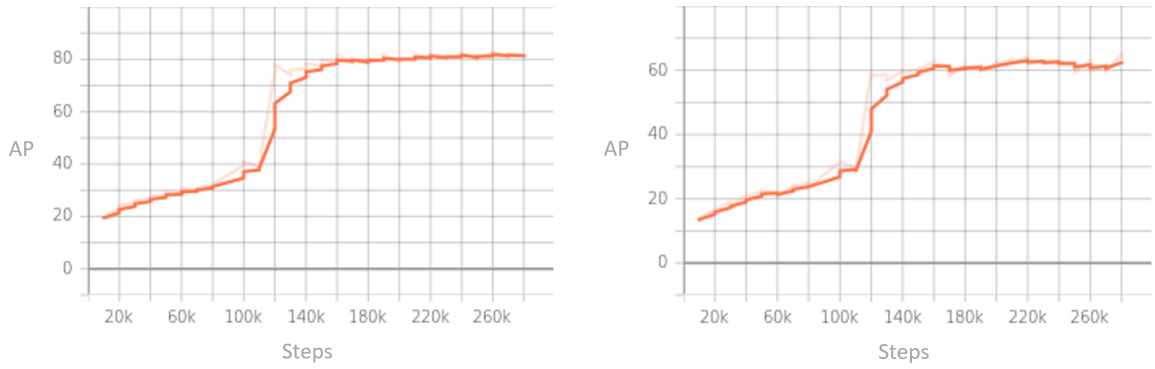
mitigates network bandwidth problems while transferring these large amounts of data. Moreover, we set the camera's frequency to its bare minimum of 15 Hz. The 3D bounding boxes derived from the RGB-D cameras are prone to noise; hence we set the $range_{max}$ value to 5 meters and discard detections that fall outside this range. For the fixed 3D bounding box l_{fix} and h_{fix} dimensions, we use the values 0.40 m and 1.80 m, respectively. To compensate for the offset in depth between the object's surface and center, we use a value of $l_{offset} = 0.2m$. To check for collision between two pedestrians, we use a minimum distance of 0.40 m for r_{dist} and a threshold value $thresh_{vol}$ of 0.25. The voxel dimensions, vox_{dim}, of the 3D object detector are slightly increased from the original (0.16x0.16x3.0) to (0.20x0.20x3.0) to decrease inference time and the 3D object detector's confidence threshold $thresh_{LiDAR}$ is set to 0.5. Finally, similar to Weng et al. [111], we will set n_{init} to 3 frames before we promote an initialized track to active.

6.3. 3D Object Detector Training

We train and validate our 3D object detector using the dataset described in section 4.1. The loss during training is visualized in fig. 6.5. Furthermore, the model's performance is visualized by the two graphs in fig. 6.4. The graph in fig. 6.4(a) depicts the model's performance in AP at the easier 0.25 IoU 3D threshold, and the graph in fig. 6.4(b) at the harder 0.50 IoU 3D threshold. The vertical axis shows the AP, and the horizontal axis shows the number of training steps. The conventional method to represent the training duration is stating the number of training epochs, eq. (6.7).

$$\text{Epochs} = \frac{\text{Steps} * \text{Batch Size}}{\text{Training Dataset Size}} \tag{6.7}$$

Similar to TANet, we employ a batch size of 2. With 280k training steps and a training dataset size of 23560 frames, we trained for approximately 24 epochs. We stop training once we notice a significant performance drop when evaluating the validation dataset to prevent overfitting. As can be seen in fig. 6.4(b), performance on the 0.50 IoU threshold starts to degrade after 220k steps while we observe no significant increase in performance on the 0.25 IoU threshold. We will use the 3D object detector model at 220k steps to conduct our real-world tests.



(a) Validation at the easier 0.25 IoU

(b) Validation at the harder 0.50 IoU

Figure 6.4: Performance of the 3D object detector on the validation dataset measured in AP. The performance is evaluated every 10k training steps. The transparent line corresponds to the raw performance, and the bold line represents a smoothed approximation.

Figure 6.5: Total loss during the training phase of the 3D object detector at each training step. * Please note that certain sections of the curve contain multiple loss values. This phenomenon occurs when the training is stopped and, at a later moment, continued.

		YOLOv4		
		Ours	Original	Original
	Resolution	416x416	416x416	608x608
AP	IoU=0.50:0.95	0.624	0.630	0.621
	IoU=0.50	0.859	0.878	0.868
	IoU=0.75	0.724	0.731	0.730
	Small	0.268	0.277	0.295
	Medium	0.511	0.519	0.508
	Large	0.738	0.741	0.730
Inference Time	Batch size = 1	∼**15.5 ms***	31.1 ms	55.5 ms
	Batch size = 5	**48.4 ± 3.0 ms**	∼155.5 ms**	∼277.5 ms**

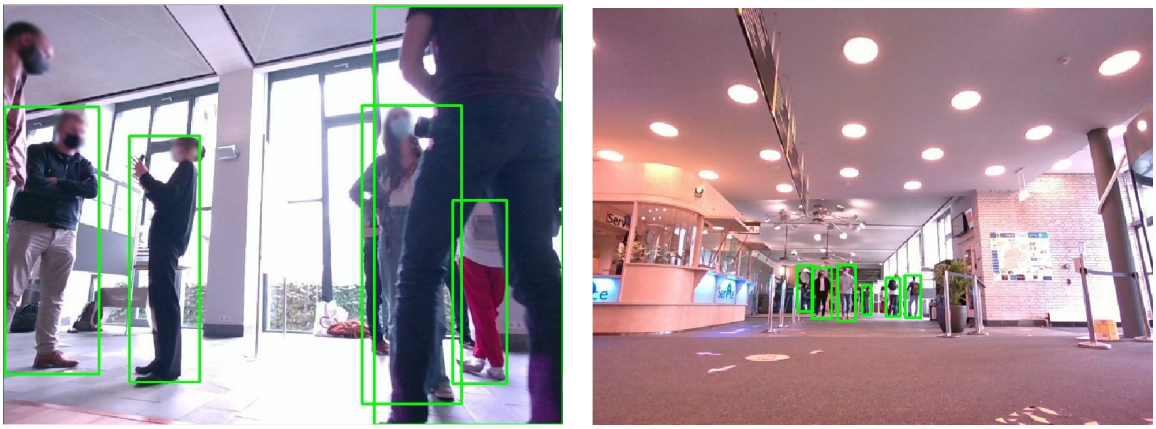
Table 6.2: Comparison of 1) the average precision (AP) at various IoU thresholds and object sizes, 2) Inference time of 2D detector for a single image and a batch of 5 images. * Note that this value is estimated using [10] and not directly measured. ** Note that these values are present to illustrate the gain in inference time and are estimated by multiplying the inference time of a single batch times 5.

6.4. 2D Bounding Box Detections

The first experiment aims to assess the performance of the implemented 2D object detector on the testing dataset described in section 4.2. The conversion of the 2D detection architecture to TensorRT optimized the 2D detection model. This optimization enables us to achieve real-time inference at the expected cost of a performance drop. In addition to the TensorRT conversion, we expect the downscaling of the images to degrade performance further. Retaining a high 2D detection accuracy is critical since uncertainties in the 2D detector propagate to the *2D to 3D Conversion* module. In this experiments, we compare the following three models:

1. **Ours @ 416x416:** Our optimized YOLOv4 model using an image resolution of 416 by 416 pixels.
2. **Original @ 416x416:** Pre-trained YOLOv4 model on MS COCO dataset using an image resolution of 416 by 416 pixels.
3. **Original @ 608x608:** Pre-trained YOLOv4 model on MS COCO dataset using an image resolution of 608 by 608 pixels.

Our goal is to examine if the conversion and image downscaling introduced a significant drop in performance. Baseline model 2) is set to use the exact image resolution as our optimized model, whereas baseline model 3) processes the images at a resolution close to the original image resolution (e.g., 480x640). Comparing the results between models 1 and 2 allows us to assess the effect of the TensorRT conversion. Comparison between baseline models 2 and 3 enables us to determine the influence



(a) Close-range detections (b) Far-range detections

Figure 6.6: Examples of 2D object detector performance. Figure 6.6(a) shows the performance of the detector at close-range and heavily occluded pedestrians. fig. 6.6(b) shows the performance of the detector when detecting far-range pedestrians.

of the image downscaling.

6.4.1. Quantitative Results

The evaluation results are shown in table 6.2. We are achieving 85.9% performance at the 0.50 IoU threshold and 72.4% at the more challenging 0.75 IoU threshold. The YOLOv4 architecture is known to trade-off bounding box accuracy against inference time. We indeed observe the effect of this trade-off since a relatively large drop in performance occurs at higher IoU thresholds. Downscaling the image resolution does not seriously affect the AP. Only a significant degradation of performance of approximately 3% is measured while evaluating small objects. However, the downscaling does impact the inference time of the three models considerably. Whereas both baseline models cannot process the image streams from 5 RGB cameras in real-time, our optimized model achieves a frame rate of approximately 21 FPS. We estimate that it outperforms the closest baseline model in terms of inference time by a factor of three.

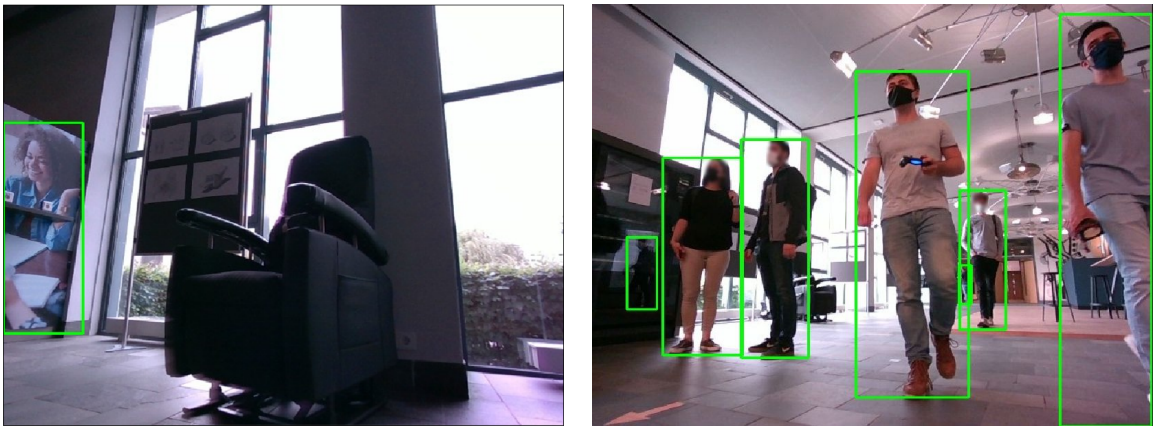
6.4.2. Qualitative Results

Qualitative performance of the 2D object detector are shown in fig. 6.6 and fig. 6.7. As can be seen in fig. 6.6, the 2D object detector works reasonable well for close-range and far-range pedestrians. Even under heavy occlusions, the detector can detect pedestrians accurately. Figure 6.7 shows two scenarios in which the 2D object detector outputs undesirable detections. Stickers and posters depicting humans, as seen in fig. 6.7(a), are mistakenly detected as pedestrians. Furthermore, a pedestrian's reflection in glass occasionally also leads to false detections. These types of false positives are almost inevitable when using 2D object detectors.

6.5. 3D Bounding Box Detections

The second experiment investigates the quality of the 3D bounding boxes provided by the *3D Detection*, *2D to 3D Object Conversion* and *Sensor Fusion & Filtering Scheme* modules, in this section referred to as LiDAR detections, RGB-D detections and fused detections, respectively. We use our testing dataset described in section 4.2 to evaluate these three modules. To avert the tedious work of creating a custom evaluation script, we employ the standard evaluation method from the TANet algorithm. This method evaluates our detected 3D bounding boxes using ground-truth data and outputs a performance in AP. As stated in section 6.1.7, computing the AP for the LiDAR, RGB-D, and fused detections and comparing the results is ambiguous. Therefore, we slightly modified TANet's evaluation method to extract the performance in terms of precision and recall.

Similar to TANet's rectangular grid voxelization, its evaluation method uses a rectangular region of



(a) Pictures of humans (b) Reflections of pedestrians

Figure 6.7: Two scenarios in which the 2D object detector outputs false positive detections. In fig. 6.7(a), the sticker on the vending machine is detected as a person. In fig. 6.7(b), the reflection of one of the pedestrians in the glass window of the vending machine is considered a person.

Method	Precision @ IoU 3D=0.50	Recall @ IoU 3D=0.50	Precision @ IoU 3D=0.25	Recall @ IoU 3D=0.25
3D Detection	0.7397	0.5829	0.9232	0.7276
2D to 3D Object Conversion	0.3787	0.3731	0.7868	0.7750
Sensor Fusion & Filtering Scheme	0.5530	0.5601	0.8804	0.8917

Table 6.3: Table containing precision and recall values and corresponding inference times at RoI: $(X_{min} = -5\ m,\ Y_{min} = -5\ m,\ X_{max} = 5\ m,\ Y_{max} = 5\ m)$. * Note that the *2D to 3D Object Conversion* module processes detections from the 2D object detector. This value represents the average processing time of these two components combined.

interest (RoI). This region of interest is defined as $(X_{min},\ Y_{min},\ X_{max},\ Y_{max})$ relative to the LiDAR's coordinate system. X_{min} and X_{max} represent the minimum and maximum allowed x-coordinate of the detected object, respectively. Similarly, Y_{min} and Y_{max} represent the minimum and maximum allowed y-coordinate of the detected object, respectively. During evaluation, detections and ground-truth objects that fall outside this RoI and discarded.

Since we limited the maximum range of the RGB-D detections to 5 meters, evaluating the RGB-D detections using ground-truth data up to the maximum detection range of 10 meters would negatively bias the results. To allow for a fair comparison between the performance of the three modules, we evaluate the detections using two different RoI. The smaller RoI includes 3D bounding boxes that are within $(X_{min} = -5\ m,\ Y_{min} = -5\ m,\ X_{max} = 5\ m,\ Y_{max} = 5\ m)$. Comparing the evaluation results using this RoI enables us to directly compare the LiDAR, RGB-D and fused detections. The larger RoI includes 3D bounding boxes that are within $(X_{min} = -10\ m,\ Y_{min} = -10\ m,\ X_{max} = 10\ m,\ Y_{max} = 10\ m)$. Since our framework is designed to operate up to a maximum detection range of 10 meters, this RoI allows us to evaluate the actual performance of our detection and tracking framework. We use this second RoI to evaluate the performance of the *3D Detection* and the *Sensor Fusion & Filtering Scheme* modules.

Please note that the fused detections provided by the *Sensor Fusion & Filtering Scheme* module do not contain all detections from the *3D Detection* and *2D to 3D Object Conversion* modules, but only the detections that passed the *Sensor Fusion & Filtering Scheme* criteria.

6.5.1. Quantitative Results

The evaluation results at the smaller RoI are shown in table 6.3. When analyzing the performance at the challenging 0.50 IoU 3D threshold, we observe that LiDAR detections achieve higher performance in terms of precision and recall than the camera detections. This can be attributed to a higher 3D

Method	Precision @ IoU 3D=0.50	Recall @ IoU 3D=0.50	Precision @ IoU 3D=0.25	Recall @ IoU 3D=0.25
3D Detection	0.7287	0.6059	0.8846	0.7355
Sensor Fusion & Filtering Scheme	0.5996	0.5839	0.8706	0.8478

Table 6.4: Table containing precision and recall values and corresponding inference times at RoI: $(X_{min} = -10\ m,\ Y_{min} = -10\ m,\ X_{max} = 10\ m,\ Y_{max} = 10\ m)$.

	Inference Time	
Method	Battery power	AC power
3D Detection	94.5 $\pm$ 22.7 ms	35.3 $\pm$ 3.6 ms
2D to 3D Object Conversion	134.4 $\pm$ 22.6 ms	68.7 $\pm$ 5.4 ms
Sensor Fusion & Filtering Scheme	-*	1.2 $\pm$ 1.1 ms

Table 6.5: A table containing the inference (or runtime) of each of the modules while running the entire pedestrian detection and tracking pipeline. The inference times are measured when the laptop is battery-powered and AC-powered. * We did not measure the inference time of this method again, but since this method does not rely on a GPU, it is less susceptible to GPU throttling.

bounding box accuracy from the *3D Detection* module. We identified multiple factors that negatively influence the accuracy of the 3D bounding boxes generated by the *2D to 3D Object Conversion* module, and we extensively discuss them in section 7.2.

The inability of the *2D to 3D Object Conversion* module to generate accurate 3D bounding boxes subsequently affects the performance of the fused detections. Since the fused detections are comprised of both LiDAR and RGB-D detections, we expect its precision value to lie in between that of the LiDAR and RGB-D detections. Table 6.3 indeed confirms this hypothesis.

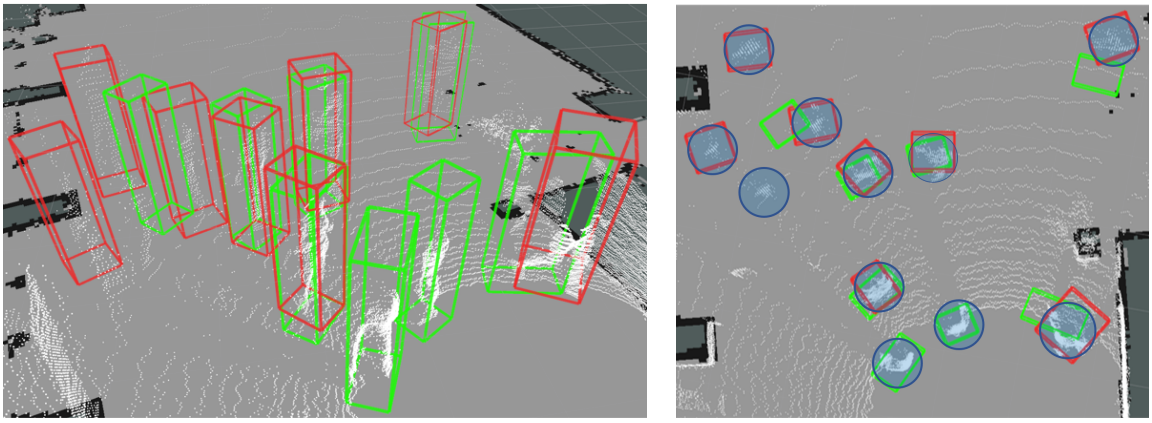
As for the fused detection's recall, we observe a slight drop in performance compared to the LiDAR detections. Since we are fusing the detections from both modalities, a proper *Sensor Fusion & Filtering Scheme* module would include TP detections from both modalities and at least achieve a performance comparable to the detector with the highest recall. Since the fused detection's recall experienced a slight drop in performance, it indicates that the filtering criteria are too strict when using the 0.50 IoU 3D threshold.

A slight advantage is that the inaccuracy of the *2D to 3D Object Conversion* module is mostly mitigated because our 3D object tracker employs an IoU 3D threshold of 0.25 to associate detection and predictions. Evaluated at this IoU 3D threshold, we observe that the fused detections significantly outperform the LiDAR and RGB-D detections in terms of recall, yet experiencing only a minor drop in precision compared to the LiDAR detections.

The evaluation results using the larger RoI are shown in table 6.4. When analyzing the performance at the 0.25 IoU 3D threshold, we again observe that the fused detections significantly outperform the LiDAR detections in terms of recall, at the cost of only a minor drop in precision. This result demonstrates the ability of the *Sensor Fusion & Filtering Scheme* module to successfully extract relevant information from both detectors and fuse the data to increase the overall 3D detection performance at the 0.25 IoU 3D threshold.

	Inference Time
Method	Battery power
3D Detection	40.0 $\pm$ 3.0 ms
2D to 3D Object Conversion	75.1 $\pm$ 4.8 ms

Table 6.6: A table containing the inference (or runtime) of each of the modules while running the pipeline with only one of the two object detection modules active. The inference times are measured when the laptop was battery powered.



(a) 3D view (b) Top-down view

Figure 6.8: Two visualizations of the same 3D point cloud showing detected pedestrians represented as 3D bounding boxes. Although not visible in these images, the mobile robot is located at the bottom right corner of the images at the center of the point cloud. The white-colored points portray the LiDAR's 3D point cloud. The green and red-colored 3D bounding boxes depicts 3D detections from the *2D to 3D Object Conversion* module and *3D Detection* module, respectively. For clarity, we added blue-colored circles in fig. 6.8(b) depicting the rough ground-truth locations of the pedestrians.

n_{term}	2D/3D	Precision	Recall
3	2D	0.8998	0.8665
	3D	0.8979	0.8595
5	2D	0.8655	0.8852
	3D	0.8768	0.8807
7	2D	0.8518	0.9017
	3D	0.8449	0.8958

Table 6.7: Table containing the evaluation results of the 3D object tracker.

Table 6.5 lists the inference time (or runtime) when running the pedestrian detection and tracking pipeline. The laptop can operate at maximum performance when powered with an external AC power adapter. From table 6.5, we observe that our pipeline achieves real-time pedestrian detection and tracking within the maximum 100 ms criteria. However, the inference times increase significantly when running the laptop solely on battery power, undermining the pipeline's real-time capabilities. Examination of the laptop's GPU statistics indicated that the GPU's power usage decreased from 120 W to 40 W. This decrease in performance can be attributed to the laptop's battery limiting the current that goes to the GPU to prevent overheating. Moreover, we noticed that the inference times increase slightly over time when using battery power, suggesting that the GPU is throttled even further as the temperature rises. Table 6.6 lists the inference times when we run our pipeline with only one of the object detectors active. This table show s that our pipeline is able to achieve real-time inference while running on battery power when only a single detector is activated.

6.5.2. Qualitative Results

Qualitative performance of the 3D detections from the *2D to 3D Object Conversion* and *3D Detection* modules is shown in fig. 6.8. Figure 6.8(b) supports the results from the quantitative performance analysis. Whereas the *3D Detection* module retains its ability to generate accurate 3D bounding boxes at larger distances, the *2D to 3D Object Conversion* module's accuracy decreases over distance due to the depth error. However, these figures also indicate that the camera detections outperform the LiDAR detections in terms of recall for pedestrians near the robot. Out of the four relatively nearby pedestrians, all of them are detected by the *2D to 3D Object Conversion*, compared to half of them are detected by the *3D Detection*. This result underlines the importance of our multi-modal sensor fusion approach; Fusing the detections to combine the strengths of each method and mitigate its weakness results in robust pedestrian detection.

6.6. 3D Object Tracking

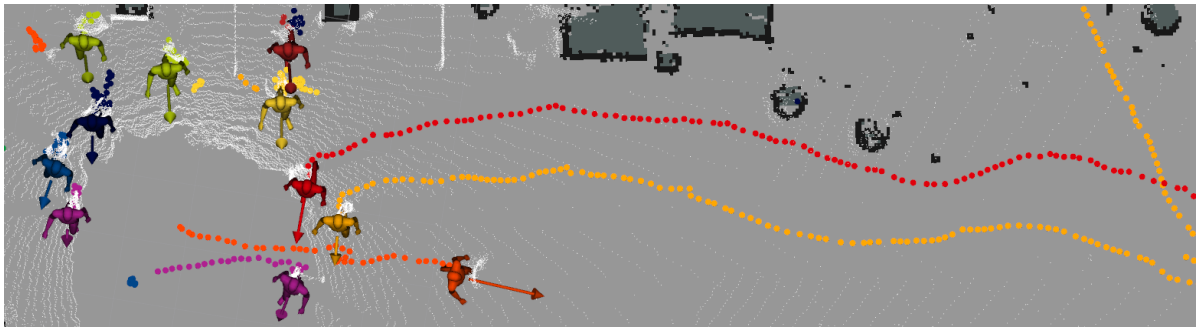
The last experiment is focused on gaining insight into the 3D object tracking performance of the framework. The original implementation of the object tracker performs 3D bounding box tracking in 3 dimensions. Since pedestrians are mostly constraint to move along the ground plane, tracking in 2 dimensions might benefit the tracking performance. Therefore we will investigate the influence of reducing the tracking problem to 2 dimensions. We constrain the 3D bounding box by allowing it to translate along the X and Y axes solely. Another hyperparameter that affects the tracking performance is the maximum age (n_{term}) a trajectory is kept active without matching any detections. A lower value will promptly terminate tracks, increasing precision at the cost of decreasing the recall. A higher value will prolong the propagation of missing tracks, decreasing the precision with the potential to increase the recall.

6.6.1. Quantitative Results

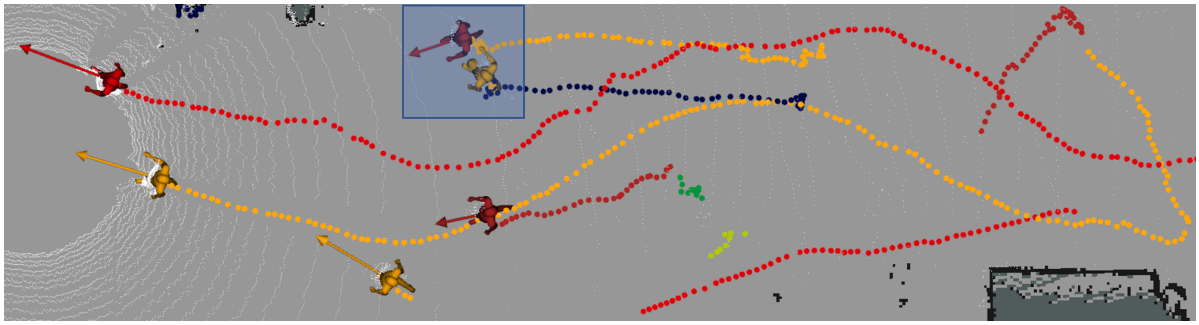
The results of the tracking evaluation are depicted in table 6.7. The simplification of the tracking problem to 2 dimensions slightly improves tracking, but only by a small margin. When we analyze the influence of the n_{term} parameter, we observe that the longer tracks are kept active, the higher the recall. This comes however at the cost of a reduced precision. Since safety is key in autonomous driving, avoiding collisions is the main priority of the autonomous navigation process, hence a high recall is desired. When tracking is performed in 2 dimensions and a n_{term} of 7 frames is used, our 3D object tracker achieves the best performance in terms of recall. When we compare these results to the performance of the fused detections from table 6.4, we observe that the recall has increased from 84.78% to 90.17%. This demonstrates the ability of the tracker to persistently keep tracking objects despite of the occasional missing detections from the object detectors. The average processing time of a single tracking step takes 3.97 ± 2.48 ms, clearly fulfilling the real-time criteria.

6.6.2. Qualitative Results

Qualitative performance of the 3D object tracker module is shown in fig. 6.9(a) and fig. 6.9(b). These figures visualize the tracked pedestrians as colored person meshes and their trajectories as dotted lines drawn in the same color as the mesh. From both fig. 6.9(a) and fig. 6.9(b), we observe the object tracker's ability to track pedestrians over relatively long periods. Inside the square blue-colored RoI in fig. 6.9(b), two commonly known tracking error can be observed. The first one, a track ID switch, mistakenly switches a pedestrian's unique track ID with that of another pedestrian. An ID switch predominantly occurs when two pedestrians are standing or walking near each other. In the RoI, the yellow-colored person mesh is switched to the blue-colored trajectory. The second common error, track fragmentation, occurs when the object tracker temporarily loses track of a pedestrian. Once the pedestrian is re-tracked at a later stage, a new and unique track ID is assigned to the pedestrian. This new track ID results in the pedestrian being visualized by different color. An example of track fragmentation occurs with the red-colored pedestrian in the RoI. Although the pedestrian is correctly tracked, the color of its trajectory changed from yellow to red. Since there is no red-colored trajectory nearby, this indicates a track fragmentation. Figure 6.10 shows a visualization of the complete pedestrian detection and tracking pipeline. It depicts the LiDAR's point cloud, the 3D bounding boxes from the *2D to 3D Object Conversion* and *3D Detection* modules, and the tracked pedestrians in Cartesian space. Furthermore, 2D bounding boxes provided by the 2D object detector are drawn in each of the 5 camera images.



(a) Scenario 1



(b) Scenario 2

Figure 6.9: Two top-down figures visualizing the framework's pedestrian tracking performance on the testing dataset. The colored person meshes represent the tracked objects, and the vectors emerging from those persons depict the orientation and magnitude of the pedestrian's velocity. The dotted lines visualize the past trajectories of these tracked objects. The blue RoI in fig. 6.9(b) highlights the two common tracking errors discussed in the text.

Figure 6.10: Sample visualization of the framework's performance on the testing dataset. Bottom: RGB images from the 5 RGB-D cameras with 2D bounding boxes of detected pedestrians. Top: 3D point cloud with red and green-colored 3D bounding boxes depicting 3D detections from the *3D Detection* module and *2D to 3D Object Conversion* module, respectively. The colored person meshes represent the tracked objects, and the arrows emerging from those persons depict the magnitude of the pedestrian's velocity. The dotted lines visualize the past trajectories of these tracked objects.

7 Discussion

In this thesis work, we present a robust real-time pedestrian detection and tracking system. To our knowledge, we were first to present a 3D object detection framework consisting of both the state-of-the-art YOLOv4 and TANet methods. In general, the proposed framework is an effective solution to detect and track multiple pedestrians in uncontrolled environments reliably. Quantitative and qualitative analysis showed robust detection and tracking performance. Moreover, real-world tests indicate that third-party motion planners can reliably leverage information from this framework to plan safe and efficient trajectories for the mobile robot. In the following sections, we provide a thorough discussion on our conducted research.

7.1. 2D Object Detector

We have presented real-time 2D object detection by implementing the YOLOv4[10] method. Research indicates that this method was the most promising object detector at that time and is capable of processing data from five cameras while still satisfying the minimum frequency of 10 Hz. Experiments showed that the optimization and pruning of the model by converting it to the TensorRT architecture did not significantly affect the model's performance. Furthermore, reducing the image resolution to decrease inference time did not influence performance either. We achieve real-time pedestrian detection for five cameras with an average inference time of 48.4 milliseconds. One of the major drawbacks of this approach is the limited field-of-view of these five cameras. There are five triangular blind spots due to the pentagonal arrangement of the cameras. Because of this phenomenon, pedestrians standing close to the cameras are difficult to detect.

7.2. 2D to 3D Object Conversion

The presented *2D to 3D Object Conversion* method showed promising results. Qualitative analysis indicated the ability to reduce pedestrians' minimum detection range significantly compared to our LiDAR detections. Furthermore, experiments showed that this method outperforms the 3D object detector in the range of 0 to 5 meters in terms of recall at the 0.25 intersection of union (IoU) threshold. However, there are still several factors that negatively influence the performance of this method. Analyzing the performance at the 0.50 IoU threshold, we observe a considerable drop in performance. This drop in performance indicates that this method has difficulty estimating the correct bounding box locations and dimensions. The following factors can explain the cause of this performance drop:

- This method relies on the disparity images from the RGB-D cameras. The exponential growth of the depth error with increasing distance considerably impacts the performance.
- Inaccurately estimated bounding box dimensions affect the IoU 3D. In our current implementation, we use constant values for the length and height dimensions of the 3D bounding boxes. In an early phase of the project, we estimated the average length and height of a pedestrian to be 0.40 m and 1.80 m, respectively. Unfortunately, we only examined the correctness of these

assumptions at the final stage of the project. The results from this examination indicated an average length and height of 0.685 m and 1.785 m, respectively. This discrepancy between our assumptions and the dataset's mean length and height values results on average in lower IoU 3D scores . We expect that adopting the more accurate width and height estimates slightly increases the detection and tracking performance.

Moreover, truncation of 2D bounding boxes at the outer boundaries of the image will result in smaller width estimates. Since the annotations in the testing dataset are based on the LiDAR point cloud, they do not experience these truncations. Comparing a truncated 3D bounding box with a ground-truth bounding box will result in a relatively low IoU value.

Finally, we examined the possibility to extract a variable bounding box length from the depth image. However, due to time constraints, we only managed to design and implement a relatively simple method. Preliminary results through visual observation did not indicate a significant improvement in overall tracking performance; hence we excluded this method and shifted our attention to the other components.

- Qualitative analysis indicated that the magnitude of the depth error varies between cameras. Although the extrinsic and intrinsic parameters of the RGB-D cameras are calibrated, this inconsistency suggests otherwise.
- The absence of adequate sensor synchronization between the five RGB-D cameras and the LiDAR results in each sensor capturing data at different time instances. Fast rotation or translation of the AGV might induce unwanted discrepancies between 3D LiDAR and RGB-D detections. Since the annotations in the testing dataset are based on the LiDAR point cloud, these discrepancies could partially explain the degradation of performance.
- The stereo depth calculations are performed by each camera individually. These cameras are designed to operate at maximum resolution in order to achieve a minimum depth error. However, the 10 HD images streamed by the cameras caused bandwidth issues when being transmitted to the laptop. In order to decrease the bandwidth allocation of the RGB-D cameras, we lowered the resolution of the RGB and depth camera stream in the camera's driver settings. Altering these settings resulted in an amplified depth error, which in turn affects accurate depth estimation.
- This method relies on the 2D bounding box detections from the YOLOv4 model. Our implementation uses a pre-trained YOLOv4 model and is trained on the MS COCO[63]. This dataset consists of 80 object classes, out of which we only use the "person" class. We have trained our 3D object detector on the KITTI[36] dataset, which consists of 8 object classes out of which three depict humans (e.g., pedestrian, cyclist, and sitting person). This class difference will result in cyclists and sitting persons being detected as relevant objects by our 2D object detector, whereas our 3D object detector will exclude these detections. Since we designed our testing dataset in accordance with the KITTI format, these detected sitting persons and cyclists will result in false positives and impede performance. We attempted to train the YOLOv4 model on the KITTI 2D Object Detection dataset to mitigate this problem. However, our model did not generalize well since the KITTI dataset includes only $\sim$1800 pedestrians compared to $\sim$250.000 persons in the MS COCO dataset. We chose to implement the YOLOv4 model trained on the MS-COCO dataset and tolerate a slight bias in the evaluation.

When analyzing the inference times, we observe that we achieve real-time inference when the laptop uses an external AC power supply. However, utilizing the laptop on battery power significantly degrades the real-time capabilities; hence we cannot guarantee real-time inference at 10 Hz using battery power.

7.3. 3D Object Detector

We have demonstrated real-time 3D object detection using point clouds by implementing the state-of-the-art TANet[68] method. Our literature survey showed that, at that time, the TANet algorithm was the best performing 3D object detector capable of real-time processing. We trained this 3D detector on our custom dataset, consisting of data from a modified version of the KITTI dataset and data collected with

our sensor suite. After validating the performance of the 3D detector using the validation dataset, we observe that the detector achieves similar performance in terms of Average Precision when comparing the $\pm$ 5 m RoI and the $\pm$ 10 m RoI. In contrast to the *2D to 3D Object Conversion* method, the *3D Detection* method is able to retain much of its performance when analyzing the differences between the 0.25 and 0.50 IoU threshold. This indicates that the locations and dimensions of the predicted 3D bounding boxes are more accurate than those provided by the *2D to 3D Object Conversion* method. When observing the performance of the *3D Detection* method in terms of recall and precision, we notice that the *2D to 3D Object Conversion* method outperforms the *3D Detection* method slightly in terms of recall at the 0.25 IoU threshold. However, the *3D Detection* method significantly outperforms the *2D to 3D Object Conversion* method in terms of precision. Finally, we analyze the real-time capabilities of the 3D detector. Although we observe that the real-time criteria is satisfied when using an AC-powered laptop, we still cannot always guarantee real-time 3D object detection when using a battery-powered laptop.

Results indicated that the 3D object detector performs generally well on the validation dataset. However, the following factors restrict the 3D object detector from reaching an even higher performance:

- Although the modified KITTI dataset accounts for the height difference between the sensor suite on top of the car and the AGV, there is still a minor change in perspective in the point clouds. This change in perspective will especially affect the detections of nearby pedestrians.

- The custom recorded dataset inside the OptiTrack environment consists of only two different adult male pedestrians. The lack of women or children in the custom dataset could affect the performance. Furthermore, there are no uncommon poses recorded, such as sitting, crouching, or bending over. When a pedestrian strikes an uncommon pose, it is unclear how the detector would react. Another issue is the lack of variation in clothing since both pedestrians are wearing trousers and a t-shirt. If the 3D object detector would encounter pedestrians wearing, for example, long coats, dresses, backpacks, or a niqaab, it is uncertain if the 3D object detector would detect these pedestrians.

7.4. Sensor Fusion & Filtering Scheme

Evaluating the results of our proposed sensor fusion and filtering scheme, we observe that the fused detections significantly outperform the two individual detectors on the 0.25 IoU threshold in terms of recall. We notice a slight decline compared to the best performing method (e.g., 3D object detector) when comparing the precision. When we analyze the 0.50 IoU threshold, we observe that the 3D object detector outperforms the fused detections on every metric. The reason for this is two-fold: Firstly, the filtering scheme filters out detections from the LiDAR, which are closer than three meters to the AGV. We have implemented this filter to prevent false positives since we expected the 3D detector to struggle with nearby pedestrians. After analyzing the precision of the 3D detector at the $\pm$ 5 meters RoI, we can conclude that this filter is too harsh and thus discards too many pedestrians. Secondly, especially within $\pm$ 3-meter range the fused detections primarily consists of detections from the *2D to 3D Object Conversion* method. As stated before, this method performs poorly on the 0.50 IoU metric and thus decreases the overall precision and recall of the fused detections.

The following factors could result in a decline in fusion performance:

- We use several hyperparameters throughout our Sensor Fusion & Filtering Scheme. Due to time constraints, the scope of our thesis did not include the optimization of these hyperparameters.

- Our *Sensor Fusion & Filtering Scheme* is currently a rule-based decision process. This approach enables us to add or remove certain rules to adjust the fusion procedure. However, we did not examine how much each of the rules contributes to the final fusion performance. Further investigation into the limitations of this fusion scheme is required to assess the significance of currently implemented rules and identify areas of improvement in order to devise new rules.

7.5. 3D Object Tracker

The 3D object tracking performance analysis indicates a significant improvement in recall and precision compared to the fused detections. Since our AGV is navigating close to humans, we desire a high recall. When we set the maximum track termination age to 7 frames, we observe an increase in recall of over 5% compared to the fused detections. This increase comes at the cost of a slightly decreased precision. The ability of the 3D object tracker to correctly propagate missed detections can explain the significant increase in recall. Moreover, we can attribute the decline in precision to tracks being propagated under heavy occlusions.

The original AB3DMOT object tracker is designed to track objects from the KITTI dataset and has its internal system matrices and noise matrices optimized for this dataset. Tuning the process noise covariance matrix, Q, and the measurement noise covariance matrix, R, is outside the scope of our thesis. Nevertheless, since our goal is to solely track pedestrians and the noise of the 3D bounding boxes provided by our object detectors differs from the KITTI dataset, we expect that tuning these parameters could improve tracking.

7.6. 3D Detection & Tracking Framework

This thesis presents a ROS-based pedestrian detection and tracking framework. We demonstrated that our pipeline achieves real-time inference when the laptop utilizes an AC power adapter. When using a battery-powered laptop, the pipeline fails to meet the real-time criteria by a small margin. We designed the pipeline to operate while one of the two object detectors is disabled. When real-time inference is required, disabling one object detector enables real-time inference at the cost of a decrease in detections range or recall, depending on which of the two detectors are disabled. The pipeline will then be able to achieve real-time pedestrian detection and tracking while running the laptop on battery power.

We are currently using four different ROS nodes for our 2D object detector, 3D object detector, sensor fusion & filtering scheme and 3D object tracker. The reason for this is two-fold. Since our framework is written in Python and Python is typically single-threaded, these four Python script (e.g., ROS nodes) enables us to distribute the complex computations over several threads. The second reason is these ROS nodes allow for a more flexible environment during the development phase. Since these ROS nodes subscribe and publish to certain topics, replacing a single module in the pipeline should come down to linking the subscribers and publishers to the correct topics. However, our current ROS implementation has some drawbacks. In our ROS nodes, we are using a timer (rospy.rate) to control the loop frequency. This timer ensures the loop iterates once at regular time intervals of 10 Hz and puts the node to rest until the next iteration. Although the hypothetical delay of our pipeline is approximately 74 milliseconds, due to our current implementation, our delay is closer to 250 milliseconds with spikes up to 350 milliseconds. We expect that redesigning the ROS pipeline to minimize downtime, for example based on subscriber callbacks instead of a timer, could further enhance the framework's performance.

8

Conclusion

This chapter will reflect on the following research objectives set out in this thesis:

1. Design and construct a multi-modal sensor suite for the Jackal Clearpath mobile research platform that facilitates a 360° horizontal field-of-view.
2. Design and implement a multi-modal multi-pedestrian 3D detection and tracking framework based on state-of-the-art methods that can process the sensory data online at a minimum frequency of 10 Hz.
3. Create a training and validation dataset by combining large-scale online 3D pedestrian dataset and a custom labeled dataset collected with our robot.
4. Create a testing dataset by collecting and annotating real-world data from an uncontrolled environment to evaluate the performance of the implemented methods.
5. Evaluate the performance of the proposed detection and tracking framework using the validation dataset.

For our first research objective, we have designed and constructed a custom sensor suite tailored for the Jackal Clearpath research platform. By mounting five RealSense D455 RGB-D cameras in a pentagonal shape and an Ouster OS1-64 LiDAR sensor, we have created a multi-modal sensor suite that is capable of perceiving its environment using a 360° horizontal field-of-view.

The custom sensor suite is vital for our second research objective, the design, and implementation of a real-time multi-modal multi-pedestrian detection and tracking framework. We have selected state-of-the-art 2D and 3D object detection methods based on extensive literature research. The framework consists of 1) a pre-trained 2D object detector, 2) a custom-designed 2D to 3D Object Conversion method to derive the 3D bounding boxes from 2D bounding boxes and their corresponding depth images, 3) a 3D object detector trained on our custom dataset, 4) a custom-designed Sensor Fusion & Filtering Scheme and 5) a 3D object tracker to track pedestrians and estimate their velocities. Although our framework achieves real-time inference when running on an externally powered laptop, we cannot always guarantee real-time inference while using a battery-powered laptop.

Our third research objective is to generate and collect a custom pedestrian dataset to train and validate our 3D object detector. We have presented a custom dataset containing 1) augmented data of the popular KITTI dataset to match the conditions of our AGV, 2) automatically labeled dataset collected by our sensor suite using an OptiTrack motion capture system, and 3) a hard-negative dataset to improve generalization of the 3D object detector in unknown indoor environments.

With our fourth research objective, we aimed to mitigate the bias in our framework's performance evaluation. Since the training and validation dataset does not contain any complex real-world scenarios,

evaluating the framework solely on this dataset would bias the results. Therefore, we have collected and annotated a 60 seconds dataset in which the AGV drives through a complex indoor environment. Our last research objective seeks to determine the performance of our proposed framework. We perform an ablation study to identify the contributions of each of the framework's modules. Quantitative analysis shows that each module positively contributes to our framework's performance and can real-time process the sensory data. Real-world experiments demonstrate robust 3D multi-pedestrian tracking in uncontrolled environments. Qualitative analysis of third-party motion planning methods indicates robust collision avoidance by successfully leveraging information from our proposed framework.

Finally, our conducted research combined with our developed AGV lays a foundation for future research in 2D/3D object detection and tracking, motion planning, and motion prediction.

9 Future Work

This chapter will discuss the recommendations for future research and development of this framework.

- **Train object detectors on larger datasets**
  Although we achieved decent 3D object detection performance while training on the KITTI dataset, there are several reasons to transition to another dataset. Firstly, we expect the 3D object detector to generalize better when trained with a larger dataset, for example, the NuScenes, Waymo, or the JRDB datasets. These recently released datasets are gaining popularity among researchers, increasing the chance for well-documented and open-source development scripts. Using the NuScenes or Waymo dataset has the benefit that they allow training of the 2D object detector on the same dataset as the 3D object detector. Currently, the KITTI dataset contains insufficient 2D annotations to train a 2D object detector that generalizes well. Training the detectors on the same dataset will mitigate some of the current problems.

- **Implement real-time instance segmentation method**
  Bounding boxes from the 2D object detector only indicate the presence of an object inside the rectangular box. However, it is unknown which pixels exactly belong to the object. Our proposed 2D to 3D Object Conversion method leverages information from the 2D bounding box to estimate the object's depth using the disparity image. Implementing a real-time instance segmentation algorithm instead of the 2D object detector would make this cumbersome method to estimate the object's depth obsolete. The instance segmentation method can precisely determine the pixels belonging to the object, significantly simplifying the depth estimation. Current state-of-the-art instance segmentation algorithms are more computationally expensive, more GPU memory demanding, have higher inference times, and significantly underperform in terms of AP compared to traditional 2D object detectors. Although overcoming these challenges is challenging, it is an interesting and promising approach.

- **Maximize the performance of the RGB-D cameras**
  We obtained the current settings of the RealSense depth cameras through trial and error. A thorough investigation into the numerous settings of the RealSense RGB-D camera can significantly improve the quality of the disparity images. Moreover, we decided to reduce the resolution of the RGB-D cameras to prevent bandwidth issues. We transmit less data to the laptop at the cost of reduced quality of the disparity images. Reverting the camera settings, processing images at maximum resolution, and creating a script on the AGV that resizes the RGB and depth images before transmitting them to the laptop could significantly improve the quality of the depth (e.g., disparity) images.

- **Extend the 2D to 3D Object Conversion method**
  The 2D to 3D Object Conversion module currently employs the depth images from the RGB-D cameras to extract an object's depth estimate. These depth images are subject to a depth estimation error that increases with distance. At the current RGB-D settings, we estimate that the depth error equals the length of a 3D object at a distance of 7.75 meters. For this reason,

we limited the maximum range of 3D objects provided by this method to 5 meters. Furthermore, a point cloud can be interpreted as a cylindrical depth image. The currently mounted LiDAR, an OS1-64, can output these cylindrical depth images by default. It would be interesting to see an extension of this method incorporating the cylindrical LiDAR depth images. This requires an even better extrinsic calibration of the RGB-D cameras and LiDAR to obtain accurate projection matrices. By following this approach, we can possibly omit the maximum depth estimation range of 5 meters. When correctly implemented, we expect that this extension can significantly improve the 3D detection performance.

- **Replace the *2D to 3D Object Conversion* and *Sensor Fusion & Filtering Scheme* methods**
  Our current *2D to 3D Object Conversion* method is based on sophisticated yet relatively old computer vision techniques. With the recent advancements in neural networks, it would be interesting to see if a neural network-based detector could accurately extract 3D bounding boxes from depth and RGB images, or from 2D bounding boxes and depth images. Furthermore, our *Sensor Fusion & Filtering Scheme* module is currently a rule-based decision process. It would be interesting to see if a neural network-based fusion method could enhance the fusion performance.

- **Re-calibrating RGB-D and LiDAR**
  Experiments indicated some problems with the intrinsic and extrinsic sensor calibration. Although the intrinsic parameters should have been calibrated in the factory, visual observation suggested different depth estimation errors between sensors. Furthermore, re-calibrating the extrinsic parameters of the sensors could be beneficial to the overall detection and tracking performance.

- **Synchronize sensors**
  Synchronizing the five RGB-D cameras properly with the LiDAR sensor could improve the detection performance slightly when quickly rotating or translating the AGV. The framework does not seem to be affected much by fast rotations and translation when visually inspecting the performance. However, adequate synchronization could be considered if future trackers or motion planners require more accurate pedestrian locations.

- **Implement a re-identification algorithm that recovers lost tracks**
  The current 3D object tracker is not capable of recovering lost tracks. We terminate the track when the tracker cannot match a track to a detected pedestrian for a certain number of frames. When the pedestrian is detected again, it takes a certain number of detections before an initialized track is active again. This tracking delay is added to limit the number of false trajectories and slightly impedes the tracking performance. By implementing a re-identification algorithm, the framework will instantly recover the lost track and avoid the delay. Furthermore, the assigned track ID of the pedestrian will stay the same when using a re-identification algorithm. If future motion planners require accurate track ID information, implementing a re-identification algorithm could be considered.

- **Integrate an external power supply for the laptop**
  The laptop achieves real-time pedestrian detection and tracking when using an external power supply. However, if the laptop is battery-powered, the GPU is throttled down to prevent overheating. This decreases the performance of the GPU and prevents the pipeline from achieving real-time inference. Since the Jackal has sufficient internal space to add an additional power supply, it might be interesting to explore the possibilities. Running an externally powered laptop decreases inference times. Moreover, running the pipeline drains the in approximately 45 minutes, after which it must be recharged for two hours. Adding an additional power supply enables operating the robot for longer periods.

A

Sensors

Autonomous vehicles rely on sensors to perceive their environment. Each sensor has its advantages and shortcomings, and often autonomous systems rely on multiple modalities to overcome the shortcomings of individual sensors.

A.1. Criteria

There are numerous different sensor types described in the literature. In order to make a selection of the most promising sensors, we need to define a set of criteria to narrow down the scope of this thesis. First, we will explore which sensors are frequently used in autonomous driving applications. Since the focus is to classify dynamic pedestrians, we expect these commonly used sensors to be better supported by datasets and detection methods. In order to navigate safely and efficiently, it is essential to obtain an accurate and complete perception of the surroundings. Therefore combining the multiple sensor modalities should result in a 360 degrees horizontal field of view. There is no clear definition of real-time, and it differs based on the application type. However, this thesis considers capturing sensory data at a minimum frequency of 10 Hz as real-time. Because this thesis focuses on AGVs navigating through crowded scenes, we expect its velocity to be similar to that of a pedestrian (e.g., 1.5 to 2 m/s). With a maximum relative velocity of 4 m/s to an approaching pedestrian, a maximum depth perception of 10 meters would allow for 2.5 seconds time to collision. Combined with the minimum frequency of 10 Hz, this results in 25 frames to detect the pedestrian and calculate its trajectory. Given that the perception and tracking pipeline performs well, we expect that depth perception up to a distance of 10 meters would suffice. A compact list of the previously described criteria is as follows:

- The sensors are frequently used in autonomous driving.
- The combined data from multiple sensors can be used to classify dynamic pedestrians.
- The combined data from multiple sensors generates a horizontal field of view of 360 degrees.
- The combined data from multiple sensors is captured at a minimum frequency of 10 Hertz.
- The combined data provides accurate depth perception from 0.1 meters up to a distance of 10 meters.

A.2. Prominent Sensors

Sensors can be divided into two groups, active and passive sensors. Active sensors actively transmit signal pulses that get reflected by objects. The sensor receives the reflected signal, and the distance to the object is measured using the Time-Of-Flight. Passive sensors receive signals which are emitted or reflected by the object being observed [103]. Van Brummelen et al. [103] researched prominent sensor arrangements on autonomous vehicles. The sensor application type, advantages, and disadvantages are listed in table A.1. Active sensors commonly used in autonomous applications are active infrared cameras, LiDAR, RADAR, and Sonar; passive sensors are monocular cameras and stereo cameras [4].

Sensors	Applications	Advantages	Disadvantages
Monocular Camera	• Obstacle detection and classification • Lane detection	• Computationally inexpensive relative to stereo vision • Good for classification • Wide Field of View while maintaining good resolution • Provides additional information about the environment (color, texture, etc.) • Long range (with high resolution cameras)	• Computationally expensive relative to active sensors • Difficulty to measure distances • Velocity information must be calculated • Poor performance in poor weather conditions • Sensitive to lighting conditions • Long range applications require more computation
Stereo Vision	• Obstacle detection and classification • Lane detection • 3D mapping	• Depth perception similar to human eyes (effective at close range) • 3D reconstruction • Good for classification • Provides additional information about the environment (color, texture, etc.) • Long range (with high resolution cameras) • Better detection than regular vision	• Computationally expensive • Velocity and distance must be calculated • Poor performance in poor weather conditions • Long range applications require more computation • Sensitive to lighting conditions
LiDAR	• Obstacle detection • 3D mapping (with multi layered LiDAR) • Lane detection (via intensity measurements)	• Direct distance measurements • Large field of view with high resolution and medium range • Multi-layer LiDAR allows robust 3D construction • May have internal mechanism to limit the impact of poor weather conditions	• Poor classification compared to vision • Velocity information must be calculated • Difficulty detecting highly reflective objects • Typically, poor detection in rain, fog and snow • Poor very near (<2 m) measurements
RADAR	• Obstacle detection	• Direct distance measurements • Direct velocity measurements • Long-range, mid-range and short-range options available • Does well in poor weather conditions • High accuracy	• Long range radar have small field of view • Poor classification • Poor very near (<2 m) measurement • Poor pedestrian detection • Poor static object detection • Interference of multiple reflections can cause false alarms
Sonar	• Near obstacle detection (e.g., parking assistance systems)	• Direct distance measurements • Very near range (<2 m) • Can operate in fog and snow	• Poor angular resolution • Poor detection beyond 2 m • Poor classification

Table A.1: Typical applications, advantages and disadvantages of common autonomous vehicle sensors. Table source: [103], [97][82][6][70][5][58] [42]

A.3. Cameras

Camera systems have been extensively used in the field of object detection. A lens focuses light rays on a photosensitive chip called a CMOS. There are two camera architectures to retrieve data from a CMOS chip, rolling and global shutter exposure modes. The former architecture continuously reads lines of pixel data from the CMOS chip. Rows of the CMOS pixel array are exposed to ambient light at different time instances. The latter architecture starts and stops the exposure of all pixels simultaneously before extracting the data from the CMOS chip.

A benefit of a rolling shutter is that the system does not have to wait until the entire CMOS chip has been read. This continuous operation enables cameras with rolling shutter modes to output data two times faster than cameras with global shutters. A drawback is that not all pixels represent the same moment in time due to the continuous mode. Fast-moving objects might appear deformed due to image distortions. The frequency at which cameras operate varies heavily. Most cameras output data at a rate of 30 frames per second; however, this rate could exceed 250 frames per second for industrial applications.

A.3.1. Monocular Camera

The monocular camera captures information in the form of pixel intensities. Most cameras utilize three channels per pixel, red, green, and blue intensities. These cameras have the advantage of having a high resolution and a wide field of view in both horizontal and vertical directions. Furthermore, cameras can capture data at relatively high frame rates. A disadvantage of monocular camera-based systems is that they lack depth information. This depth information is required to estimate the object position and pose accurately. Another disadvantage is that cameras are susceptible to changes in weather and lighting conditions. Since cameras are passive sensors, they heavily rely on sufficient ambient lighting.

A.3.2. Stereo Camera

A stereo camera-based system is a setup where two monocular cameras are placed next to one another at a predefined distance (e.g., baseline). The disparity of objects seen in both camera frames is then computed using stereo matching algorithms. This disparity value is then used to compute the depth information of the object. Stereo cameras perform especially well in estimating the depth of nearby objects; however, the performance drops significantly as distance increases. Although stereo cameras address the lack of depth information of monocular cameras, they are still susceptible to changes in weather and lighting conditions. There are several drawbacks of stereo cameras, the first one being the increased computational complexity. The matching algorithm has to find correspondences in both images and calculate the depth of each point relative to the camera [4]. Another shortcoming of stereo cameras occurs when contrast is low and objects blend into the background. These textureless regions render stereo matching difficult and result in poor depth estimation performance. Furthermore, when many repetitive patterns are visible in the camera image, the stereo matching algorithm might output wrong feature associations, resulting again in poor depth estimations.

A.3.3. Active Infrared Camera

There are two types of frequently used active infrared depth cameras. The first type employs structured light technology [102]. A small infrared projector projects points in a predefined pattern on a surface. An infrared camera detects these points and computes their depth based on the observed pattern. The second type is called a Time-of-Flight camera. These cameras perceive depth by sending modulated infrared pulses and measuring the delay between emitting and receiving these pulses. The advantages of such systems are that they resolve the problem of poor performances in different lighting conditions and decrease computational complexity. The drawbacks are the relatively low resolutions compared to monocular and stereo cameras.

A.4. LiDAR

A LiDAR sensor consists of one or multiple lasers which emit laser pulses. Objects reflect these laser pulses, and the LiDAR again detects these reflections. The time between emitting and receiving the reflection is used to measure the distance to an object. The strength of the received reflection is a measure of the reflectance of an object. LiDARs with multiple lasers generate a set of 3D points

combined in a point cloud. Because the LiDAR is an active sensor, pulses that are not reflected by an object result in an infinite distance. These infinite distances are filtered out and result in sparse point clouds. An advantage of the LiDAR is that it is more robust to extreme lighting conditions. Standard LiDAR models, such as the VLP-16 [79], use an array of multiple lasers inclined at different angles. This array is mounted on a rotating platform generating 3D point clouds in a 360 degrees angular field of view. The minimum distance the LiDAR can accurately measure points is around 2 meters, and the maximum distance is 100 meters. The point cloud resolution decreases as the frame rate increases, with most LiDARs having a maximum frame rate of 20 Hz. A drawback of LiDAR sensors is the vertical field of view. The application type of most LiDARs is medium to long-range detection, which means to sensor requires a small vertical field of view. When using this LiDAR type to classify nearby pedestrians, the point cloud will not cover the entire pedestrian. When the LiDAR is placed at the height of 0.5 meters, an average pedestrian will be fully visible from around 5 meters. There exist LiDARs with a broader field of view that can detect objects much closer by, but they are less suited for long-range detection.

A.5. RADAR

RADAR sensors consist of one or multiple antennas used to transmit and receive radio waves. The distance to the object can be calculated by measuring the time between transmitting and receiving the radio wave. The minimum distance for the RADAR to accurately detect an object is around 2 meters, the maximum distance depends on the type of RADAR, but they work over relatively long distances. When using multiple receiving antennas, the RADAR can determine the angle of the object. One of the benefits of a RADAR is that it can directly extract the velocity of an object by measuring the Doppler effect. Another benefit of the RADAR is its robustness. In contrast to the LiDAR, there are no moving parts prone to mechanical failure. The drawbacks are its poor angular resolution and object classification capabilities compared to the LiDAR. A typical automotive RADAR sensor updates at a rate of 5 to 30 frames per second.

A.6. Sonar

Ultrasonic sensors consist of an ultrasonic sound wave emitter and receiver. Objects reflect the emitted sound waves, and the receiver detects these reflections. The object's distance can be computed by using the time between the emitting and receiving and the speed of light. These sensors are being used to detect obstacles up to 2 meters distance and therefore are commonly used in autonomous driving for slow-moving procedures, such as parking assist. One of the benefits of the ultrasonic sensor is that they conduct direct distance measurements. They do not require expensive computations, which is the case for stereo cameras, and they are inexpensive to manufacture compared to LiDAR or RADAR. However, due to their poor classification performance and poor angular resolution, they are not suited for object classification. The frequency at which these sensors operate depends on the propagation of sound waves through the air. For a maximum detection range of 2 meters at room temperature, this sensor could reach up to 100 frames per second.

A.7. Conclusion

There is a vast number of sensors capable of perceiving the environment. Most frequently used in autonomous driving are the monocular camera, stereo camera, active infrared camera, LiDAR, RADAR, and Sonar. Each of these sensors satisfies the requirement to provide real-time data at a minimum frequency of 10 Hertz. Table A.2 gives an overview of the operating distance of the previously discussed sensors.

The maximum operating distance of a high-resolution camera depends on the type of object it tries to capture. For pedestrian detection, the monocular camera performs well within the entire operating distance of 20 meters. The well-established datasets and the maturity of camera-based object detection architectures are invaluable for robust object detection. The absence of depth information of these camera-based object detectors can be addressed in several ways.

The sonar is an inexpensive solution for 3D obstacle detection within two meters. Several sensors

	Dimension	<2m	>2m & <5m	>5m & <10m	>10m & <20m
Monocular Camera	2D	✓	✓	✓	✓
Stereo Camera	3D	✓	✓	~	✗
Active Infrared Camera	3D	✓	✓	~	✗
LiDAR	3D	✗	~	✓	✓
RADAR	3D	✗	~	✓	✓
Sonar	2D/3D	✓	✗	✗	✗

Table A.2: Operating distance of various sensors

can be placed around the AGV to detect surrounding obstacles. These sensors suffice when used for low-speed collision avoidance. However, they are not suited for object detection tasks due to their poor angular resolution.

If the application requires the classification of close-range objects within 2 meters, the stereo camera and active infrared camera are better alternatives than the sonar. The active infrared camera is preferred over the stereo camera for systems with limited computational power, relatively low-resolution requirement, or operating under difficult lighting conditions. When the system can deliver the high computational demand of a stereo camera and the application type requires more accurate depth perception, the higher resolutions of the stereo camera are preferred over the active infrared camera.

For object detection from a range of 2 meters up to 5 meters distance, the LiDAR, RADAR, stereo camera, and active infrared camera are viable solutions. LiDAR and RADAR sensors provide accurate depth information. However, due to their narrow vertical field of view, large objects can get truncated at closer distances. For application types where relatively large objects need to be accurately detected at close distances, the larger field of view of the stereo and active infrared camera is preferred.

In the range of 5 to 10 meters, the LiDAR and RADAR start to outperform the stereo and active infrared cameras. Although the latter two sensors still output data within this range, their depth accuracy drops significantly as distance increases.

The LiDAR and RADAR are the only two frequently used sensors for long-range object detection. It depends on the application task which sensor is preferred. The RADAR outperforms the LiDAR by using the Doppler effect for detecting dynamic objects. The LiDAR again outperforms the RADAR when used for object classification tasks. Using the high-resolution point cloud of the LiDAR, object detectors can classify dynamic and static objects. Another benefit of the LiDAR is its horizontal field of view. The sensor setup requires multiple RADAR sensors in contrast to a single LiDAR sensor for surround vision.

There are several possible sensor setups to combine 2D and 3D data for the entire operating distance, and it depends on the application type which setup is preferred.

B

2D Object Detector Benchmark

Image-based 2D object detectors can be divided into two categories, one-stage detectors, and two-stage detectors. Generally, two-stage detectors are known to have high localization and object recognition accuracy. The first of the two stages is a Region of Interest (RoI) pooling layer, sometimes referred to as a Region Proposal Network (RPN). This layer proposes candidate bounding boxes. The second stage uses these candidate bounding boxes and performs classification and bounding-box regression tasks. A one-stage detector predicts bounding boxes directly from the images without a region proposal layer. By omitting the separate region proposal network, the one-stage detectors are significantly faster than two-stage detectors and can be used in real-time applications [45].

B.1. Criteria

As discussed in chapter 1, we have set the criteria for real-time 360 degrees detection on processing a minimum of 10 frames per second. Our AGV requires five Intel RealSense D455 depth cameras to achieve a 360° horizontal FOV, translating to a minimum frame rate of 10 Hz for five cameras or 50 Hertz for a single camera.

When comparing the various detection methods, they must be evaluated on the same dataset. Because section 2.1 showed that the MS COCO dataset gives the best representation of state-of-the-art 2D object detectors, we decided to use that dataset to evaluate the methods. It is also important that a paper has been published on the topic to verify the quality of the author's work. For implementation purposes, it is also vital that the source code of the object detector is publicly available. We have summarized these criteria in the following list:

- A single RGB image from a camera has to be processed at a frame rate of 50 frames per second.
- The algorithm is evaluated on the MS COCO dataset.
- A paper has been published on the topic.
- The code of the object detector is publicly available.

B.2. Object Detectors

A common pitfall in comparing the real-time capabilities of object detectors is comparing the inference times directly. Due to the rapid advancements of GPUs and budget differences between research groups, many different GPUs are used in the research field. The computational power of these GPUs varies considerably, which directly affects the methods' inference time.
Bochkovskiy et al. [10] conducted an extensive inquiry into real-time 2D object detection algorithms evaluated on the MS COCO dataset. In their paper, they addressed the challenge of comparing the inference time of various object detection methods by dividing all methods among three subsets, each subset being a different GPU architecture, see fig. B.1. Figure B.1 consists of 6 subfigures, the three on the left depicting performance measured against the relatively hard mAP 0.95 metric, the three on

the right measured in the relatively easy mAP 0.50 metric. The networks depicted in the two uppermost figures are evaluated using a GPU with Maxwell architecture. Networks depicted in two figures in the middle used a GPU with Pascal architecture, and the bottom two used a GPU with Volta architecture. We have divided the individual figures into two sections, a white area, and a blue area. The blue area represents real-time performance at a minimum of 50 FPS. Figure B.2 gives an overview of methods capable of reaching inference times substantially surpassing the real-time criteria of 50 FPS.
Due to the outstanding work of Bochkovskiy et al., this thesis will build upon their results.

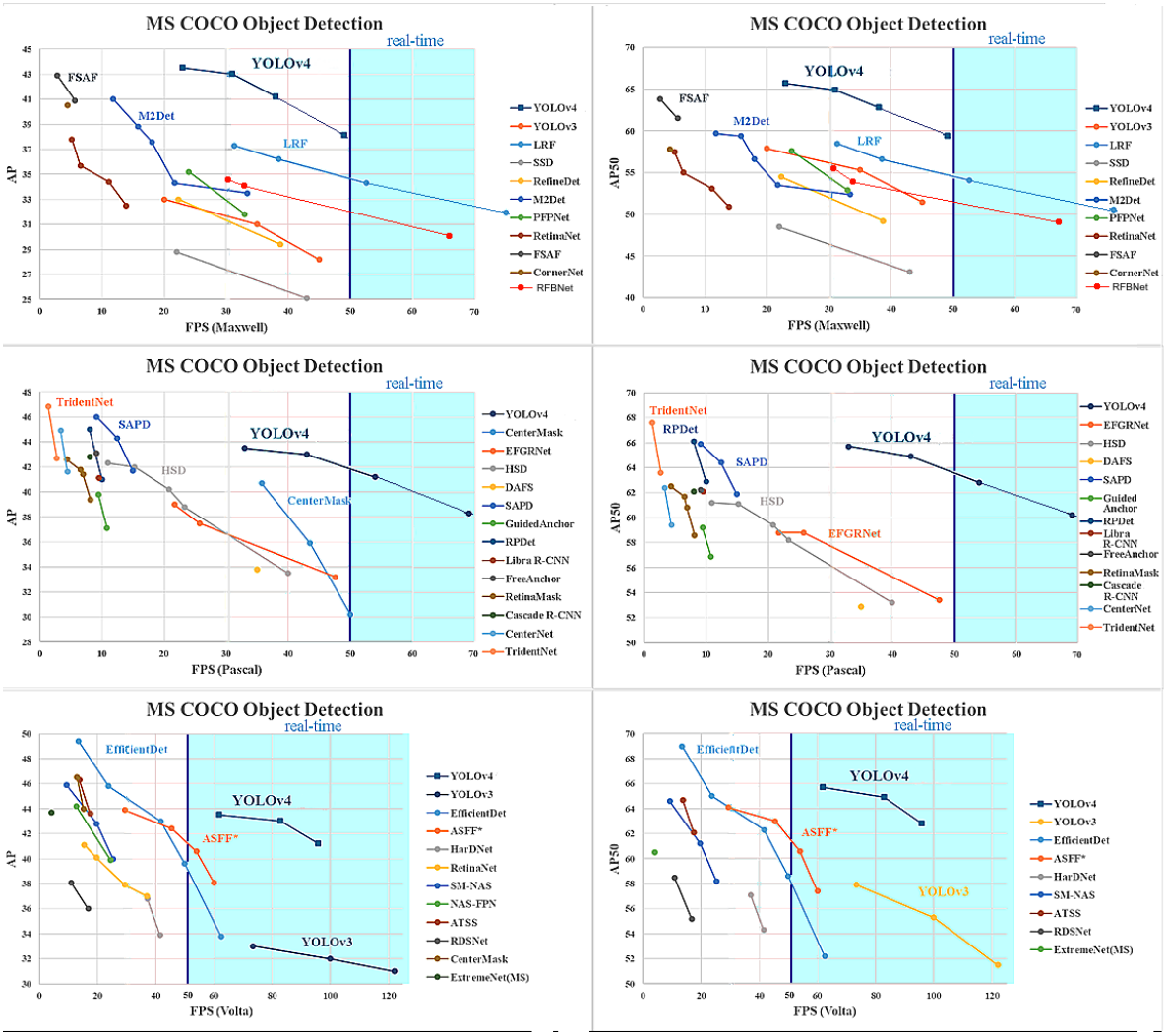


Figure B.1: Comparison of various real-time detection algorithms. Original figure retrieved from GitHub [2]
YOLOv4[10], YOLOv3[83], LRF[110], SSD[67], RefineDet[121], M2Det[124], PFPNet[50], RetineNet[62], FSAF[128], CornerNet[55], RFBNet[66], CenterMask[57], EFGRNet[76], HSD[17], DAFS[60], SAPD[129], GuidedAnchor[107], RPDet[115], Libra R-RCNN[77], FreeAnchor[122], RetinaMask[33], Cascade R-CNN[16], CenterNet[28], TridentNet[61], EfficientDet[101],Liu2019LearningDetection ASFF*[65], HarDNet[18], SM-NAS[118], NAS-FPN[37], ATSS[120], RDSNet[109], ExtremeNet[126]

These results show that the LRF method performs best in the real-time region on GPUs with a Maxwell architecture, followed closely by the RFBNet method. Although YOLOv4 achieves the best performance in the sub-real-time region, it slightly underperforms on the real-time criteria. Compared to methods evaluated on the Pascal architecture, YOLOv4 is the only method achieving real-time performance. On the Volta architecture, YOLOv3 achieves the highest frame rates. However, YOLOv4 outperforms YOLOv3 in terms of AP by roughly 10 percent. For methods evaluated on the NVidia GTX1080Ti, the CSP method achieves the best performance in the real-time region of at least 50 FPS. The two methods with the fastest inference times are the CPS method and the YOLOv3-tiny-RPN method.

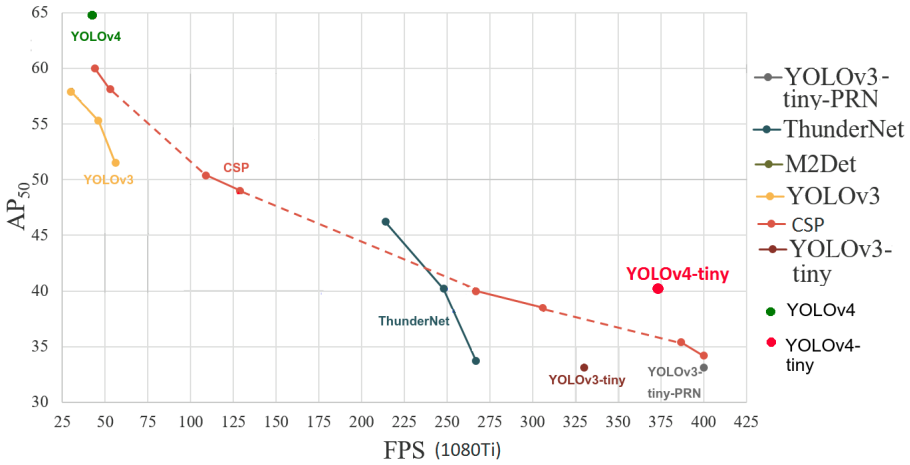


Figure B.2: Comparison of various fast real-time detection algorithms. Original figure retrieved from GitHub [3].
*Methods are evaluated on a NVidia GTX1080Ti.
YOLOv3-tiny-RPN[106], ThunderNet[81], M2Det[124], YOLOv3[83], CPS[105], YOLOv3-tiny[83], YOLOv4[10], YOLOv4-tiny[10]

B.3. Image Segmentation

Image segmentation can be formulated as a classification problem of pixels. It involves partitioning images into multiple segments of predefined object categories. Image segmentation can be sub-divided into two groups, semantic segmentation, and instance segmentation. Semantic segmentation labels each pixel of the image and classifies them to a set of object categories. Instance segmentation extends the semantic segmentation algorithm by detecting objects of interest and delineating them. The research field of real-time instance segmentation is emerging as only recently technology has progressed enough to deal with the complexity. Whereas object detectors only predict the presence of an object in that box, image segmentation provides pixel-level information of the location of an object. However, semantic segmentation and especially instance segmentation, is more computationally expensive than state-of-the-art object detectors [73]. Table B.1 lists the current state-of-the-art real-time instance segmentation algorithms. The inference time of all methods has been evaluated using GPUs similar to the Pascal architecture in fig. B.1. The best instance segmentation algorithms only reach a performance of 35.2 AP at 33.3 FPS. Compared to the classical object detectors in fig. B.1, the state-of-the-art algorithm reaches a performance of 43.8 AP at 33 FPS. Only the CenterMask + MobilenetV2 algorithm barely satisfies the 50 FPS criteria set in this thesis.

Model	CenterMask + MobileNetV2 [57]	YOLACT [11]	YOLACT-550 [11]	BlendMask [19]	YOLACT-550++ [11]
AP	25.2	24.9	28.2	**35.2**	34.6
FPS	**50**	45.5	33.3	33.3	27.3

Table B.1: A list of current real-time instance segmentation algorithms evaluated on the MS COCO dataset.

B.4. Conclusion

The inference times of object detection methods heavily depend on the computational power available to the system. Because most methods use different processing units, it is not easy to compare their real-time capabilities.

In general, the segmentation approach is preferred over the object detection approach. Pixel-level labeling offers precise information on the exact location of the objects, whereas bounding boxes only indicate that there is an object present in the rectangle. The drawback of the segmentation approach is the high computational load and the inferior performance compared to object detectors. Implementing

an instance segmentation algorithm is the preferred approach when an application requires a precise understanding of the environment, has access to a cutting-edge GPU, and relies less on inference speed and false-positive detections.

If the segmentation approach does not meet the design requirements, classical object detectors should be considered. When high performance in terms of AP is desired, the EfficientDet algorithm is the preferred method. With an AP of 49.8 and AP50 of 69, this network is the best sub-real-time algorithm. As for real-time detection networks, it is clear that YOLOv4 outperforms all other models on the Pascal and Volta GPU architectures. Only on the Maxwell architecture GPU, YOLOv4 does not meet the real-time criteria, and instead, the LRF model is the preferred method.

When frame rates well beyond real-time are required for the application and performance is less critical, the algorithms in fig. B.2 should be considered. The selection of the best method depends on the required AP and FPS.

C

3D Object Detector Benchmark

The 2D object detectors discussed in appendix B acquire information on the presence of objects and their location in the camera images. In many dynamic robotics applications, this is not sufficient. Due to the absence of depth information, tasks such as path planning and motion prediction become infeasible. Depth cameras and LiDARs enable capturing a 3D representation of the environment. 3D object detection methods can then be employed to classify objects in the 3D data and estimate their oriented bounding boxes.

C.1. Representations

One of the biggest challenges of 3D object detection is the sparse nature of the 3D data. LiDAR point clouds rely on emitted laser pulses reflecting off objects and returning to the sensor. The distance towards an object can be computed by using the time it took the LiDAR pulse to return to the sensor and the speed of light. If there is no object reflecting the pulse, no distance is computed, and thus no point is added to the point cloud. For stereo cameras, the same phenomenon occurs when the camera is unable to match certain pixels. This results in sparse point clouds whose size fluctuates in subsequent frames.

Because almost all state-of-the-art Deep Learning architectures rely on fixed-size input data, the sparsity of point clouds is a complex topic. In order to circumvent this problem, various solutions have been proposed to transform the point cloud into an acceptable format. Van der Sluis [98] discussed these representations thoroughly in his literate review and states that the most frequently used 3D representations are Voxels, Pillars, Stixels, Frustums, and Pointsets. As for 2D representations, popular representations are the Birds Eye View and the Range-View.

Because current networks are still not capable of handling complete point cloud data, discretization of the data is required. Each type of representation results in different levels of information loss due to discretization.

C.2. Criteria

As discussed in chapter 1 and chapter 3, the criteria for real-time 3D object detection is set on processing a minimum of 10 frames per second. To compare the performance of the various detection methods, they must be evaluated using the same dataset. From section 2.1, we concluded that the KITTI dataset is preferred due to its popularity and mandatory submission of the methods inference time. Therefore we use the KITTI dataset to evaluate the performance of the methods. Since this thesis tries to give an overview of methods capable of being implemented on a small AGV, it is vital that these methods are both publicly available and supported by a published paper. A summary of these criteria is as follows:

- The point cloud data can be processed at a minimum frequency of 10 frames per second.

- The algorithm is evaluated on the KITTI pedestrian dataset.
- The code of the 3D object detector is publicly available.
- A paper has been published on the topic.

C.3. Detection algorithms

We used the previously stated criteria to make a list of state-of-the-art 3D detection methods, seen in table C.1. The modality column indicates which sensor data the method utilizes. The easy, moderate, and hard columns refer to the annotation's difficulty levels. The performance at each difficulty level is measured in mAP. The FPS column shows the inference speed in frames per second stated by the authors. The last column states the GPU used to evaluate the method.

Method	Modality	Easy	Moderate	Hard	FPS	Device
TANet [68]	LiDAR	53.72 %	**44.34 %**	**40.49 %**	28.57	Titan V GPU
3DSSD [116]	LiDAR	**54.64 %**	44.27 %	40.23 %	25	Titan V GPU
MMLab-PartA^2 [93]	LiDAR	53.10 %	43.35 %	40.06 %	12.5	Tesla V100 GPU
MMLab-PV-RCNN [94]	LiDAR	52.17 %	43.29 %	40.29 %	12.5	8 GTX 1080 Ti GPU
STD [117]	LiDAR	53.29 %	42.47 %	38.35 %	12.5	Titan V GPU
AVOD-FPN [51]	RGB+LiDAR	50.46 %	42.27 %	39.04 %	10	Titan X (Pascal) GPU
PointPillars [54]	LiDAR	51.45 %	41.92 %	38.89 %	**62.5**	1080ti GPU and Intel i7 CPU
epBRM [95]	LiDAR	49.17 %	41.52 %	39.08 %	10	GTX Titan X (Maxwell) GPU
MMLab-PointRCNN [92]	LiDAR	47.98 %	39.37 %	36.01 %	10	GPU *
SECOND [114]	LiDAR	45.31 %	35.52 %	33.14 %	20	GTX1080 Ti GPU
SECOND-Small [114]	LiDAR	45.31 % **	35.52 % **	33.14 % **	40	GTX1080 Ti GPU
BirdNet+ [7]	LiDAR	37.99 %	31.46 %	29.46 %	10	Titan Xp GPU
AVOD [51]	RGB+LiDAR	36.10 %	27.86 %	25.76 %	12.5	Titan X (Pascal) GPU

Table C.1: List of state-of-the-art 3D detection algorithms[36].
* No GPU specified in paper
** Only data available from class "Car" available, minor drop in performance noticed.

C.4. Comparing Inference Time

Comparing inference time of algorithms is difficult due to the wide variety of GPUs used to evaluate the methods. Each GPU has its advantages and drawbacks, and a method's inference time depends on it. A method utilizing a relatively weak GPU might seem to underperform in terms of inference time compared to a method using a powerful one. However, if both methods are evaluated on the same GPU, the seemingly inferior method might outperform the latter. The best comparison would be to train and evaluate all methods on the same GPU. As this approach is unattainable due to limited time and GPU availability, we explored two other approaches.

The first one uses an online GPU benchmark database [8]. A large number of GPUs is trained and evaluated on 19 state-of-the-art neural networks. An inference score is computed for each GPU, indicating its performance. The second approach relates to the teraFLOPS ratio. Using the number of operations a GPU can execute in a second, it is possible to estimate the inference time as if all methods were evaluated using the same GPU. Because the benchmarked data is not published in a paper, we follow both approaches to verify the validity of the benchmark database. The teraFLOPS and inference score ratios are computed relative to the GPU with the highest number of teraFLOPS, the Titan V GPU.

To obtain the adjusted FPS of each method, we divide the original FPS by the corresponding ratio. The number of teraFLOPS, the inference score, and the corresponding inference ratios of each network are listed in table C.2. The performances of each method along with their corresponding inference times in FPS are listed in table C.3

Name	**teraFLOPS (FP32)**	**TeraFLOPS Ratio (FP32)**	**Inference Score**	**Inference Score Ratio**
Jetson AGX Xavier	1.410	0.095	2226	0.137
GTX Titan X	6.691	0.449	7443	0.459
Titan X	10.97	0.736	9714	0.599
Titan Xp	12.15	0.815	11948	0.738
Titan V	**14.90**	**1**	16192	1
NVIDIA RTX 2070 Super Mobile	7.066	0.474	8133	0.502
Tesla V100	14.13	0.948	**16511**	**1.019**
GTX 1080 Ti	11.34	0.761	11914	0.736

Table C.2: Specification of GPUs. Source: [39]

Method	Easy	Moderate	Hard	FPS Original	FPS teraFLOPS	FPS Inference Score
TANet [68]	53.72 %	**44.34 %**	**40.49 %**	28.57	28.57	28.57
3DSSD [116]	**54.64 %**	44.27 %	40.23 %	25	25	25
MMLab-PartA^2 [93]	53.10 %	43.35 %	40.06 %	12.5	13.18	12.27
MMLab-PV-RCNN [94]	52.17 %	43.29 %	40.29 %	12.5	16.43	16.98
STD [117]	53.29 %	42.47 %	38.35 %	12.5	12.5	12.5
AVOD-FPN [51]	50.46 %	42.27 %	39.04 %	10	13.59	16.69
PointPillars [54]	51.45 %	41.92 %	38.89 %	**62.5**	**82.13**	**84.92**
epBRM [95]	49.17 %	41.52 %	39.08 %	10	22.27	21.79
MMLab-PointRCNN [92]	47.98 %	39.37 %	36.01 %	10 *	10 *	10 *
SECOND [114]	45.31 %	35.52 %	33.14 %	20	26.28	27.17
SECOND Small [114]	45.31 % **	35.52 % **	33.14 % **	40	52.56	54.35
BirdNet+ [7]	37.99 %	31.46 %	29.46 %	10	12.27	13.55
AVOD [51]	36.10 %	27.86 %	25.76 %	12.5	16.98	20.87

Table C.3: Adjusted inference time independent of GPU.
* No GPU specified in paper
** Only data available from class ”Car” available, minor drop in performance noticed.

C.5. Conclusion

Interestingly, methods utilizing LiDAR data solely outperform the methods combining both RGB and LiDAR data. An explanation of this anomaly could be the sensor fusion method used in the algorithm. The best performing multi-modal method AVOD-FPN uses Bird-Eye-View as LiDAR data representation and combines it with RGB data. The Bird-Eye-View representation projects 3D points on a 2D plane by removing the height (Z-axis) data. This type of representation discards much valuable information, and this might influence the performance. The TANet method achieves the highest performance on the moderate and hard difficulty. On the easy difficulty, the 3DSSD method performs slightly better than the TANet method.

When comparing the two approaches to eliminate the dependency of the inference time on GPU type, the teraFLOPS ratio and the Inference Score ratio retrieved from the benchmark database give similar results. We can conclude that the teraFLOPS ratio is a metric that could be used to estimate the inference time of algorithms. It is worth noting that in this literature review, we assumed that the inference time solely depends on the speed of the GPU and that the GPU is the bottleneck in the pipeline. It is possible that for some methods, the pre or post-processing step consumes most of the time, and switching to a more powerful GPU will not increase inference time substantially.
Although some methods have seen an increase of over 50 % in inference time, the absolute ranks do not shift much. The PointPillars algorithm is still the fastest method, with the SECOND_Small and TANet methods in the second and third place, respectively. The SECOND method claims fourth place and pushes the 3DSSD method to the fifth spot. Interestingly, the epBRM method, which was ranked last in terms of original FPS, is now ranked in sixth place.

When designing a system with a dedicated GPU that requires the integration of a 3D object detector, the TANet method is the preferred algorithm. The inference time is fast enough to process all data in real-time, and the TANet method outperforms all other methods. When high frame rates are required, and the system uses 3D sensors with high FPS such as stereo cameras, the PointPillars method and, to a lesser extent, the SECOND method are viable solutions. Although the performance is only 2 % mAP lower, the PointPillars method sees an increase in inference time of nearly 200% compared to the TANet method.

D

3D Object Tracker Benchmark

Object tracking is an essential component in many autonomous driving applications. Acquiring knowledge of the previous locations of the object allows the system to use this temporal information to predict the motion of objects. Object tracking can be divided into two groups, Single-Object Tracking (SOT) and Multi-Object Tracking (MOT). Whereas in SOT, the object's appearance is known a priori, in MOT, an object detector must identify the targets first. These targets are not restricted to the image plane, meaning they can enter and leave the scene. The main difficulty in simultaneously tracking multiple targets are the many occlusions of objects, and similar appearances between objects [24].

D.1. Criteria

As discussed in chapter 1 the criteria for real-time 3D object tracking is set on processing a minimum of 10 frames per second. In order to differentiate between the tracking methods, they must be evaluated on the same dataset. In section 2.1, we concluded that the KITTI dataset is preferred due to its popularity and mandatory submission of the methods inference time. Therefore we will use the KITTI dataset to evaluate the performance of the tracking methods. For implementation purposes, the tracking methods must be both publicly available. A published paper must support the performance of the tracker in order to validate the integrity of the tracker. A summary of these criteria is as follows:

- The algorithms tracks objects at a minimum frequency of 10 frames per second.
- The code of the object tracker is publicly available.
- A paper has been published on the topic.
- The algorithm is evaluated on the KITTI MOT pedestrian dataset.

D.2. Object Trackers

The KITTI MOT evaluation server provides a leaderboard of all submitted methods. Table D.1 lists the methods that pass the previously set criteria. KITTI MOT evaluates the performance of the trackers using 2D IoU. Methods that output 3D detections first need to project these 3D bounding boxes to the image plane before they can be evaluated. The absence of depth information in the KITTI MOT metrics creates a biased leaderboard towards 2D trackers due to the increased complexity and dimensionality of tracking in three dimensions.

The two best-performing methods solely use the image plane to track objects. The third and fourth methods use 3D detections as input. The biased leaderboard makes it challenging to compare the various methods. Weng et al. [111] proposed a new evaluation metric in their AB3DMOT paper to solve this problem. It involves separating 2D from 3D trackers and evaluating the 3D trackers using 3D IoU instead of 2D IoU. The NuScenes Tracking dataset embraced this proposal and made this their standard evaluation metric. Although the NuScenes leaderboard lists 30 methods as of Nov 25th, 2020, the submission of the networks inference time is not mandatory, and only four methods have submitted this

data. This lack of information complicates the ability to compare real-time object trackers and renders the use of the NuScenes Tracking dataset impracticable. This thesis follows the recommendations of Weng et al. and split the object trackers into a 2D and 3D category. Table D.1 lists the trackers that pass the previously set criteria.

Method	Space	MOTA	MOTP	MT	ML	IDS	FRAG	FPS	Device
Quasi-Dense [78]	2D	**56.81%**	73.99%	31.27%	**18.90%**	254	1121	14.29*	Tesla V100 GPU
CenterTrack [125]	2D	55.34%	**74.02%**	**34.71%**	19.93%	95	751	22.22	Intel Core i7-8086K CPU Titan Xp GPU**
JRMOT [91]	2D/3D	46.33%	72.54%	23.37%	28.87%	345	1111	14.29	4 core CPU
AB3DMOT [111]	3D	39.63%	64.87%	16.84%	41.58%	170	940	**212.77**	1 core CPU
NOMT [23]	2D	36.93%	67.75%	17.87%	42.61%	**34**	789	11.11	16 core CPU
LP-SSVM [108]	2D	33.33%	67.38%	12.37%	45.02%	72	818	20	1 core CPU
CEM [72]	2D	27.54%	68.48%	8.93%	51.89%	96	**608**	11.11	1 core CPU

Table D.1: Object tracker performance on the KITTI pedestrian dataset.
*FPS of both detector and tracker combined, no FPS available of solely the object tracker.
**No information available on which device is used for the object tracker.

D.3. 2D Trackers

Due to matured and highly developed 2D object detectors, most multi-object trackers focus on perceiving motion from 2D RGB images. Shenoi et al. [91] state that it is computationally more expensive to detect and track objects in 3D than in 2D due to the *curse of dimensionality*. A benefit of using 2D images to track objects is the information density of RGB images. The dense information allows detectors and trackers to discern object appearances to detect, identify, and classify the object effectively.

Most trackers formulate the tracking problem as a data-association problem. In the temporal window, data associations are made between the multiple detections. The two best performing 2D tracking methods are Quasi-Dense [78] and CenterTrack [125].

D.3.1. Quasi-Dense

The Quasi-Dense method uses a 2D object detector to predict regions of interest. Most object detectors detect multiple regions of interest per object. These regions of interest are then filtered to remove the redundant regions to output a single bounding box. In contrary to standard object detectors, the Quasi-Dense method utilizes these multiple regions of interest to learn instance similarity. Once the similarity of an instance is extracted, the resulting feature descriptor is matched to the feature descriptors of previous tracks.

D.3.2. CenterTrack

The CenterTrack method is based on two key components. First, a 2D detector is used to predict the bounding boxes of objects. Each object is then represented by a single point located at the center of its bounding box. The center points of the objects are then tracked over time. A constellation of tracked points is then represented by a heatmap of points. The second component, a tracking-conditioned detector, uses this heatmap to associate detections to previous tracks. The object association relies on point displacement prediction and allows objects in different frames to be linked. Because CenterTrack relies on the prediction of the displacement of objects, it means that re-identification of lost tracks is not possible.

D.3.3. Evaluation of Methods

When evaluating the 2D trackers from the KITTI MOT dataset, we see that the Quasi-Dense method performs slightly better on the MOTA metric than the CenterTrack method and performs comparably

on the MOTP metric. CenterTrack outperforms Quasi-Dense on the Most Tracked (MT), the Identity Switches (IDS), fragments (FRAG), and frame rate (FPS) metrics. Quasi-Dense performs better on the Most Lost (ML). It is interesting to see that CenterTrack outperforms Quasi-Dense on the IDS metric. This metric stands for the number of times a tracked object switches ID. One of the key components of Quasi-Dense is instance similarity matching. One could expect the Quasi-Dense method to excel in the IDS metric, given that CenterTrack does not consider the appearance of objects.

D.4. 3D Trackers

3D object trackers are less frequently used. The field of 3D object detection is emerging fast and opens the doors for the development of competitive 3D trackers. It is much harder for 3D detectors and trackers to re-identify an object after it is lost because, contrary to point clouds, 2D images provide much more information on the appearance of an object. However, the depth information allows 3D trackers to separate objects that might overlap in 2D space. This could especially enhance the performance in cluttered environments.

After removing the methods from the KITTI MOT leaderboard that did not meet the set criteria, only two methods remained, the JRMOT and AB3DMOT methods. The authors of JRMOT [91] state in their paper that the only other open-source real-time 3D object tracker they could find was the AB3DMOT [111] method. Given this information, this thesis only considers these two methods.

D.4.1. JRMOT

The JRMOT method uses both a 2D and 3D object detector. The 2D object detector is used to detect pedestrians as 2D object detectors are faster, more robust, and more accurate. The ReID [123] method is employed to create a feature descriptor based on the appearances of the pedestrians. Next, frustums are extracted from the point cloud based on the 2D detections to downsample the 3D data. A 3D object detector is then used to detect pedestrians in these frustums. Furthermore, the bounding box dimensions of the 3D detections are also used as feature descriptors. Both feature descriptors contain valuable information to associate the detections to previous tracks. The two feature descriptors coming from the 2D and 3D object detectors are fused using a 3-layer fully connected network, resulting in a feature that can be used to associate current detections and previous tracks. Appearance similarity is then calculated, and using a fixed threshold the resulting associations are made. The final step is filtering the noisy tracks using a Kalman filter [48].

D.4.2. AB3DMOT

In contrast to JRMOT, the AB3DMOT method is more straightforward and focuses solely on 3D detections from the point clouds. An off-the-shelf 3D object detector is used to provide public detections. These detections are then used by a 3D Kalman filter (with a constant velocity model), and the Hungarian algorithm [52] for state estimation and data association. Unlike many other 2D MOT methods, the state estimation of the AB3DMOT method is extended to 3D space, including 3D locations, 3D velocity, and object orientation.

D.4.3. Evaluation of Methods

The KITTI dataset does not provide official public detections. This means that the performance of the submitted algorithms depends both on the quality of the object detector's detections and the tracking algorithm's actual performance. Most authors of tracking algorithms decide to use an off-the-shelf object detector to provide the necessary public detections. However, the performance of both object detectors and trackers keeps improving, and state-of-the-art trackers will integrate the best detectors. Both the JRMOT and AB3DMOT methods use public detections in KITTI MOT to eliminate the influence of the object detector on the tracking performance. However, JRMOT uses public detections from the SubCNN [113] object detector, and AB3DMOT uses public detections from the PointRCNN [92] object detector. The performances of these two object detectors is listed in table D.2.

Method	Moderate	Easy	Hard	FPS	Device
SubCNN [113]	66.70 %	79.65 %	61.35 %	0.5	GPU
MMLab-PointRCNN [92]	47.33 %	57.19 %	44.31 %	10	GPU

Table D.2: Performances of the SubCNN and PointRCNN object detectors on the KITTI 2D pedestrian dataset

It is clear that the quality of the public detections generated by the SubCNN method is better than the PointRCNN method's. The higher performance of the JRMOT method compared to the AB3DMOT method can thus not simply be attributed to the quality of the tracking algorithm. Shenoi et al. [91] also introduced a novel tracking dataset (JRDB) on which both methods are evaluated using the same public detections. This allows for the comparison between both methods. The results of the evaluation can be seen in fig. D.1.

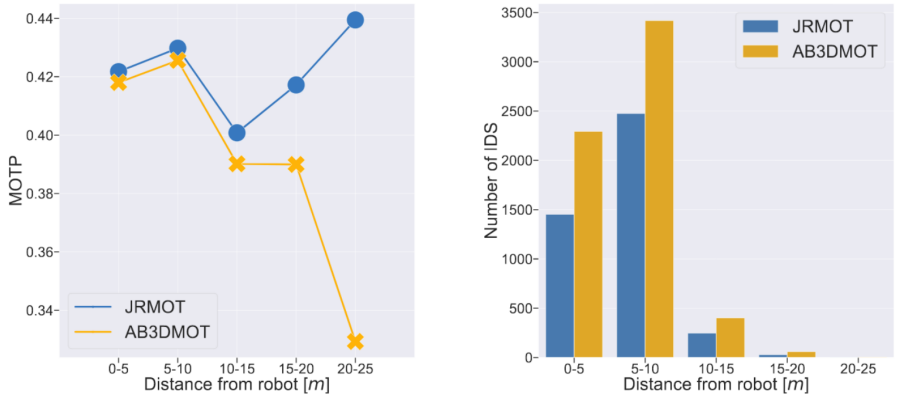


Figure D.1: Comparison of the JRMOT and AB3DMOT methods on the JRDB dataset. Image source [91]

D.5. Conclusion

The KITTI and NuScenes Tracking evaluation leaderboards are impractical when it comes to comparing real-time 3D object trackers. The NuScenes dataset lacks the mandatory submission of inference time, and the KITTI dataset uses controversial metrics to compare both 2D and 3D trackers. It is interesting to see the differences between the two datasets. On the KITTI leaderboard, the two best-performing methods are based on 2D detectors and perform slightly better than the two best 3D methods. Although we are not evaluating the NuScenes dataset in this thesis, it is interesting to see the impact of the different metrics being used. The two best methods on the NuScenes leaderboard rely solely on LiDAR data, with the best method achieving an AMOTA of 0.693. The third and fourth methods follow a multi-modal approach by using both camera and LiDAR data. The first method solely based on camera images is listed the 20th place with an AMOTA of 0.177. The significant performance differences between these two datasets are alarming, and conclusions based solely on the leaderboards should be interpreted carefully.

Because there are several tracking metrics, it depends on the design requirements which method is in favor. In general, a method with both a high MOTA and MOTP is desired. When multiple methods achieve similar performance on these two metrics, one of the other tracking metrics must be considered to make a decision.

When designing a system using only camera images, the Quasi-Dense and CenterTrack methods perform best. It depends on the design requirements which method is preferred. When long persistent tracks are desired, the CenterTrack method is favored since it achieves low FRAG and high MT scores. Although the IDS score of CenterTrack is lower than Quasi-Dense, the Quasi-Dense method is preferred if one is trying to re-identify objects due to the instance similarity matching component.

When the system relies purely on LiDAR data, the AB3DMOT is the best option. It has the fastest

inference time of all trackers and performs reasonably well on the KITTI 2D tracking dataset. It runs on a single CPU core, which is interesting when integrating real-time tracking methods in mobile applications.

If the system relies on multi-modal sensor data and has access to both a CPU and GPU, the JRMOT method is a good option. It relies on state-of-the-art 2D and 3D detectors, computes appearance feature descriptors, and fuses this data to associate detected objects to existing tracks. Its performance is higher than the AB3DMOT method, but its inference time is slower and has more ID switches on the KITTI tracking dataset. When evaluating the performance on the custom JRDB dataset, it appears that the JRMOT method starts outperforming AB3DMOT as object distance increases, see fig. D.1. This can be explained by the sparsity of the point cloud increasing significantly the further away the object stands. The 2D object detector in the JRMOT method suffers less from this phenomenon because of the higher camera resolution compared to LiDAR resolution.

All in all, it is challenging to compare the various real-time tracking algorithms. The problem of not having public detections creates a biased tracking performance. Once the NuScenes tracking dataset starts demanding accurate frame rate submissions and the KITTI tracking dataset embraces the recommendations of Weng et al. [111], a descent inquiry into real-time multi-object trackers can be made.

Bibliography

[1] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. [Online; accessed November 9, 2021]. 2016. URL: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[2] Alexey Bochkovskiy. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. [Online; accessed November 17, 2020]. 2020. URL: https://user-images.githubusercontent.com/4096485/80283279-0e303e00-871f-11ea-814c-870967d77fd1.png.

[3] Alexey Bochkovskiy. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. [Online; accessed November 17, 2020]. 2020. URL: ttps://user-images.githubusercontent.com/4096485/85734112-6e366700-b705-11ea-95d1-fcba0de76d72.png.

[4] Eduardo Arnold et al. "A Survey on 3D Object Detection Methods for Autonomous Driving Applications". In: *IEEE Transactions on Intelligent Transportation Systems* (2019). ISSN: 15580016. DOI: 10.1109/TITS.2019.2892405.

[5] Andrew Bacha et al. "Odin: Team VictorTango's entry in the DARPA Urban Challenge". In: *Journal of Field Robotics* (2008). ISSN: 15564959. DOI: 10.1002/rob.20248.

[6] Hernán Badino, Daniel Huber, and Takeo Kanade. "Integrating LIDAR into stereo for fast and improved disparity computation". In: *Proceedings - 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2011*. 2011. ISBN: 9780769543697. DOI: 10.1109/3DIMPVT.2011.58.

[7] Alejandro Barrera et al. *Birdnet+: End-to-end 3d object detection in lidar bird's eye view*. 2020.

[8] *Benchmark*. URL: http://ai-benchmark.com/ranking_deeplearning_detailed.html.

[9] Keni Bernardin and Rainer Stiefelhagen. "Evaluating multiple object tracking performance: The CLEAR MOT metrics". In: *Eurasip Journal on Image and Video Processing* 2008 (2008). ISSN: 16875176. DOI: 10.1155/2008/246309.

[10] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: (2020). URL: http://arxiv.org/abs/2004.10934.

[11] Daniel Bolya et al. "YOLACT: Real-time instance segmentation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00925.

[12] Markus Braun et al. "The EuroCity Persons Dataset: A Novel Benchmark for Object Detection". In: (2018), pp. 1–18. DOI: 10.1109/TPAMI.2019.2897684. URL: http://arxiv.org/abs/1805.07193%0Ahttp://dx.doi.org/10.1109/TPAMI.2019.2897684.

[13] Holger Caesar et al. "Nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.01164.

[14] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: 2020. DOI: 10.1109/cvpr42600.2020.01164.

[15] Yingfeng Cai et al. "YOLOv4-5D: An Effective and Efficient Object Detector for Autonomous Driving". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021). ISSN: 15579662. DOI: 10.1109/TIM.2021.3065438.

[16] Zhaowei Cai and Nuno Vasconcelos. "Cascade R-CNN: Delving into High Quality Object Detection". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Dec. 2017), pp. 6154–6162. URL: http://arxiv.org/abs/1712.00726.

[17] Jiale Cao et al. “Hierarchical shot detector”. In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2019-October. Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 9704–9713. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00980.

[18] Ping Chao et al. “HarDNet: A Low Memory Traffic Network”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2019-October (Sept. 2019), pp. 3551–3560. URL: http://arxiv.org/abs/1909.00948.

[19] Hao Chen et al. “Blendmask: Top-down meets bottom-up for instance segmentation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.00860.

[20] Xiaozhi Chen et al. “3D object proposals for accurate object class detection”. In: *Advances in Neural Information Processing Systems*. Vol. 2015-January. 2015.

[21] Xiaozhi Chen et al. “Multi-view 3D object detection network for autonomous driving”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua (2017), pp. 6526–6534. DOI: 10.1109/CVPR.2017.691.

[22] Jiyu Cheng et al. “Autonomous Navigation by Mobile Robots in Human Environments: A Survey”. In: *2018 IEEE International Conference on Robotics and Biomimetics, ROBIO 2018*. 2018. DOI: 10.1109/ROBIO.2018.8665075.

[23] Wongun Choi. “Near-online multi-target tracking with aggregated local flow descriptor”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.347.

[24] Gioele Ciaparrone et al. “Deep learning in video multi-object tracking: A survey”. In: *Neurocomputing* (2020). ISSN: 18728286. DOI: 10.1016/j.neucom.2019.11.023.

[25] Bogdan Cramariuc, Moncef Gabbouj, and Jaakko Astola. “Clustering based region growing algorithm for color image segmentation”. In: *International Conference on Digital Signal Processing, DSP*. Vol. 2. 1997. DOI: 10.1109/icdsp.1997.628490.

[26] Ankit Dhall et al. “LiDAR-Camera Calibration using 3D-3D Point correspondences”. In: *arXiv* (May 2017). URL: http://arxiv.org/abs/1705.09785.

[27] Donghyeop Shin. *Deep Neural Network-Based Scene Graph Generation for 3D Simulated Indoor Environments*. [Online; accessed November 9, 2021]. 2019. URL: http://ktsde.kips.or.kr/digital-library/full-text/view?doi=10.3745/KTSDE.2019.8.5.205.

[28] Kaiwen Duan et al. “CenterNet: Keypoint Triplets for Object Detection”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2019-October (Apr. 2019), pp. 6568–6577. URL: http://arxiv.org/abs/1904.08189.

[29] Richard O. Duda and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures”. In: *Communications of the ACM* 15.1 (1972). ISSN: 15577317. DOI: 10.1145/361237.361242.

[30] Martin Ester et al. “A Density-Based Clustering Algorithms for Discovering Clusters”. In: *KDD-96 Proceedings* 96.34 (1996). ISSN: 09758887.

[31] Mark Everingham et al. “The pascal visual object classes (VOC) challenge”. In: *International Journal of Computer Vision* 88.2 (2010). ISSN: 09205691. DOI: 10.1007/s11263-009-0275-4.

[32] Di Feng et al. “Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges”. In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–20. ISSN: 1524-9050. DOI: 10.1109/tits.2020.2972974.

[33] Cheng-Yang Fu, Mykhailo Shvets, and Alexander C. Berg. “RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free”. In: *arXiv* (Jan. 2019). URL: http://arxiv.org/abs/1901.03353.

[34] Keinosuke Fukunaga and Larry D. Hostetler. “The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition”. In: *IEEE Transactions on Information Theory* 21.1 (1975). ISSN: 15579654. DOI: 10.1109/TIT.1975.1055330.

[35] S. Garrido-Jurado et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. In: *Pattern Recognition* 47.6 (2014). ISSN: 00313203. DOI: 10.1016/j.patcog.2014.01.005.

[36] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the KITTI vision benchmark suite”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2012. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248074.

[37] Golnaz Ghiasi, Tsung Yi Lin, and Quoc V. Le. “NAS-FPN: Learning scalable feature pyramid architecture for object detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. IEEE Computer Society, June 2019, pp. 7029–7038. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00720.

[38] Ross Girshick. “Fast R-CNN arXiv1504.08083v2-1”. In: *arXiv* (2015).

[39] *GPU Specs Database*. URL: https://www.techpowerup.com/gpu-specs/.

[40] Kaiming He et al. “Mask R-CNN”. In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2017-October. 2017. DOI: 10.1109/ICCV.2017.322.

[41] Tong He and Stefano Soatto. “Mono3D++: Monocular 3D vehicle detection with two-scale 3D hypotheses and task priors”. In: *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*. 2019. DOI: 10.1609/aaai.v33i01.33018409.

[42] Mahmoud Al Henawy and Martin Schneider. “Integrated antennas in eWLB packages for 77 GHz and 79 GHz automotive radar sensors”. In: *European Microwave Week 2011: ”Wave to the Future”, EuMW 2011, Conference Proceedings - 41st European Microwave Conference, EuMC 2011*. 2011. ISBN: 9782874870224. DOI: 10.23919/eumc.2011.6102012.

[43] Takehiro Horiuchi et al. “Pedestrian tracking from a mobile robot using a laser range finder”. In: *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*. 2007. DOI: 10.1109/ICSMC.2007.4413964.

[44] Alexandru Iloie, Ion Giosan, and Sergiu Nedevschi. “UV disparity based obstacle detection and pedestrian classification in urban traffic scenarios”. In: *Proceedings - 2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing, ICCP 2014*. 2014. DOI: 10.1109/ICCP.2014.6936963.

[45] Licheng Jiao et al. “A survey of deep learning-based object detection”. In: *IEEE Access* (2019). ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2939201.

[46] John Wilson. *Could you explain me how instance segmentation works?* [Online; accessed November 9, 2021]. 2019. URL: https://ai-pool.com/d/could-you-explain-me-how-instance-segmentation-works.

[47] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. [Online; accessed November 22, 2021]. 2018. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[48] R. E. Kalman. “A new approach to linear filtering and prediction problems”. In: *Journal of Fluids Engineering, Transactions of the ASME* (1960). ISSN: 1528901X. DOI: 10.1115/1.3662552.

[49] Puneet Kohli and Anjali Chadha. “Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash”. In: *Lecture Notes in Networks and Systems*. Vol. 69. 2020. DOI: 10.1007/978-3-030-12388-8{_}19.

[50] Hyong-Keun Kook et al. *Parallel Feature Pyramid Network for Object Detection Seung-Wook Kim [0000−0002−6004−4086]*. Tech. rep.

[51] Jason Ku et al. “Joint 3D Proposal Generation and Object Detection from View Aggregation”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2018. ISBN: 9781538680940. DOI: 10.1109/IROS.2018.8594049.

[52] H. W. Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* (1955). ISSN: 00281441. DOI: 10.1002/nav.3800020109.

[53] R. Labayrade, D. Aubert, and J.-P. Tarel. “Real time obstacle detection in stereovision on non flat road geometry through ”v-disparity” representation”. In: 2003. DOI: 10.1109/ivs.2002.1188024.

[54] Alex H. Lang et al. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2019. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.01298.

[55] Hei Law and Jia Deng. “CornerNet: Detecting Objects as Paired Keypoints”. In: *International Journal of Computer Vision* 128.3 (Aug. 2018), pp. 642–656. URL: http://arxiv.org/abs/1808.01244.

[56] Laura Leal-Taixé et al. “MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking”. In: (2015), pp. 1–15. URL: http://arxiv.org/abs/1504.01942.

[57] Youngwan Lee and Jongyoul Park. “CenterMask: Real-time anchor-free instance segmentation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.01392.

[58] John Leonard et al. “A perception-driven autonomous urban vehicle”. In: *Springer Tracts in Advanced Robotics*. 2009. ISBN: 9783642039904. DOI: 10.1007/978-3-642-03991-1{_}5.

[59] Bo Li, Tianlei Zhang, and Tian Xia. “Vehicle detection from 3D lidar using fully convolutional network”. In: *Robotics: Science and Systems*. Vol. 12. 2016. DOI: 10.15607/rss.2016.xii.042.

[60] Shuai Li et al. *Dynamic Anchor Feature Selection for Single-Shot Object Detection*. Tech. rep.

[61] Yanghao Li et al. “Scale-aware trident networks for object detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2019-October. Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 6053–6062. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00615.

[62] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (Aug. 2017), pp. 318–327. URL: http://arxiv.org/abs/1708.02002.

[63] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2015). ISSN: 10636919.

[64] Timm Linder and Kai O. Arras. “People detection, tracking and visualization using ROS on a mobile service robot”. In: *Studies in Computational Intelligence* 625 (2016). ISSN: 1860949X. DOI: 10.1007/978-3-319-26054-9{_}8.

[65] Songtao Liu, Di Huang, and Yunhong Wang. “Learning Spatial Fusion for Single-Shot Object Detection”. In: *arXiv* (Nov. 2019). URL: http://arxiv.org/abs/1911.09516.

[66] Songtao Liu, Di Huang, and Yunhong Wang. “Receptive Field Block Net for Accurate and Fast Object Detection”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11215 LNCS (Nov. 2017), pp. 404–419. URL: http://arxiv.org/abs/1711.07767.

[67] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9905 LNCS (Dec. 2015), pp. 21–37. DOI: 10.1007/978-3-319-46448-0{_}2. URL: http://arxiv.org/abs/1512.02325%20http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[68] Zhe Liu et al. “TANet: Robust 3D Object Detection from Point Clouds with Triple Attention”. In: (2019). ISSN: 2374-3468. DOI: 10.1609/aaai.v34i07.6837. URL: http://arxiv.org/abs/1912.05163.

[69] J MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1. 1967.

[70] Will Maddern and Paul Newman. “Real-time probabilistic fusion of sparse 3D LIDAR and dense stereo”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2016. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759342.

[71] Roberto Martin-Martin et al. “JRDB: A Dataset and Benchmark of Egocentric Robot Visual Perception of Humans in Built Environments”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). ISSN: 19393539. DOI: 10.1109/TPAMI.2021.3070543.

[72] Anton Milan, Stefan Roth, and Konrad Schindler. “Continuous energy minimization for multitarget tracking”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014). ISSN: 01628828. DOI: 10.1109/TPAMI.2013.103.

[73] Shervin Minaee et al. “Image Segmentation Using Deep Learning: A Survey”. In: (2020), pp. 1–20. URL: http://arxiv.org/abs/2001.05566.

[74] Takahiro Nakada, Satoshi Kagami, and Hiroshi Mizoguchi. “Pedestrian detection using 3D optical flow sequences for a mobile robot”. In: *Proceedings of IEEE Sensors*. 2008. DOI: 10.1109/ICSENS.2008.4716556.

[75] Bodam Nam, Sung Il Kang, and Hyunki Hong. “Pedestrian detection system based on stereo vision for mobile robot”. In: *2011 17th Korea-Japan Joint Workshop on Frontiers of Computer Vision, FCV 2011*. 2011. DOI: 10.1109/FCV.2011.5739758.

[76] Jing Nie et al. “Enriched feature guided refinement network for object detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2019-October. Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 9536–9545. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00963.

[77] Jiangmiao Pang et al. “Libra R-CNN: Towards balanced learning for object detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. IEEE Computer Society, June 2019, pp. 821–830. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00091. URL: https://github.com/OceanPang/.

[78] Jiangmiao Pang et al. *Quasi-Dense Instance Similarity Learning*. 2020.

[79] *Puck Lidar Sensor, High-Value Surround Lidar*. URL: https://velodynelidar.com/products/puck/.

[80] Charles R. Qi et al. “Frustum PointNets for 3D Object Detection from RGB-D Data”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2018), pp. 918–927. ISSN: 10636919. DOI: 10.1109/CVPR.2018.00102.

[81] Zheng Qin et al. “ThunderNet: Towards Real-time Generic Object Detection”. In: (Mar. 2019), pp. 1–10. URL: http://arxiv.org/abs/1903.11752.

[82] R. H. Rasshofer and K. Gresser. “Automotive radar and lidar systems for next generation driver assistance functions”. In: *Advances in Radio Science* (2005). ISSN: 16849965. DOI: 10.5194/ars-3-205-2005.

[83] Joseph Redmon and Ali Farhadi. “YOLO v.3”. In: *Tech report* (2018), pp. 1–6. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[84] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem (2016), pp. 779–788. ISSN: 10636919. DOI: 10.1109/CVPR.2016.91.

[85] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017). ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031.

[86] Ergys Ristani et al. “Performance measures and a data set for multi-target, multi-camera tracking”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9914 LNCS.c (2016), pp. 17–35. ISSN: 16113349. DOI: 10.1007/978-3-319-48881-3{_}2.

[87] Cristina Romero-González et al. “InLiDa: A 3D lidar dataset for people detection and tracking in indoor environments”. In: *VISIGRAPP 2017 - Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. Vol. 6. 2017. DOI: 10.5220/0006148704840491.

[88] SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016*. 2018.

[89] R E Schapire. “A Short Introduction to Boosting”. In: *Society* (2009). DOI: 10.1.1.112.5912.

[90] Joel Schlosser, Christopher K. Chow, and Zsolt Kira. “Fusing LIDAR and images for pedestrian detection using convolutional neural networks”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2016. ISBN: 9781467380263. DOI: 10.1109/ICRA.2016.7487370.

[91] Abhijeet Shenoi et al. *JRMOT: A Real-Time 3D Multi-Object Tracker and a New Large-Scale Dataset*. 2020.

[92] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. “PointRCNN: 3D object proposal generation and detection from point cloud”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2019. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00086.

[93] Shaoshuai Shi et al. “From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). ISSN: 0162-8828. DOI: 10.1109/tpami.2020.2977026.

[94] Shaoshuai Shi et al. “PV-RCNN: Point-voxel feature set abstraction for 3D object detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.01054.

[95] Kiwoo Shin and Masayoshi Tomizuka. *Improving a quality of 3D object detection by spatial transformation mechanism.* 2019.

[96] Martin Simon et al. “Complex-YOLO: An euler-region-proposal for real-time 3D object detection on point clouds”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2019. ISBN: 9783030110086. DOI: 10.1007/978-3-030-11009-3{_}11.

[97] Sayanan Sivaraman and Mohan Manubhai Trivedi. “Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis”. In: *IEEE Transactions on Intelligent Transportation Systems* (2013). ISSN: 15249050. DOI: 10.1109/TITS.2013.2266661.

[98] J.R.van der Sluis. “3D Object Detection For Intelligent Vehicles”. In: 2020.

[99] Pei Sun et al. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: 2020. DOI: 10.1109/cvpr42600.2020.00252.

[100] Pei Sun et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.00252.

[101] Mingxing Tan, Ruoming Pang, and Quoc V. Le. “EfficientDet: Scalable and Efficient Object Detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Nov. 2019), pp. 10778–10787. URL: http://arxiv.org/abs/1911.09070.

[102] R. J. Valkenburg and A. M. McIvor. “Accurate 3D measurement using a structured light system”. In: *Image and Vision Computing* (1998). ISSN: 02628856. DOI: 10.1016/s0262-8856(97)00053-x.

[103] Jessica Van Brummelen et al. *Autonomous vehicle perception: The technology of today and tomorrow*. 2018. DOI: 10.1016/j.trc.2018.02.012.

[104] Carl Vondrick et al. “Efficiently Scaling up Crowdsourced Video Annotation A Set of Best Practices for High Quality, Economical Video Labeling”. In: *Int J Comput Vis* 101 (2013), pp. 184–204. DOI: 10.1007/s11263-012-0564-1. URL: http://mit.edu/vondrick/vatic..

[105] Chien Yao Wang et al. “CSPNet: A new backbone that can enhance learning capability of CNN”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. Vol. 2020-June. IEEE Computer Society, June 2020, pp. 1571–1580. ISBN: 9781728193601. DOI: 10.1109/CVPRW50498.2020.00203. URL: https://github.com/WongKinYiu/CrossStagePartialNetworks..

[106] Chien-Yao Wang et al. *Enriching Variety of Layer-wise Learning Information by Gradient Combination*. Tech. rep.

[107] Jiaqi Wang et al. “Region Proposal by Guided Anchoring”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (Jan. 2019), pp. 2960–2969. URL: http://arxiv.org/abs/1901.03278.

[108] Shaofei Wang and Charless Fowlkes. “Learning Optimal Parameters For Multi-target Tracking”. In: 2015. DOI: 10.5244/c.29.4.

[109] Shaoru Wang et al. “RDSNet: A New Deep Architecture for Reciprocal Object Detection and Instance Segmentation”. In: *arXiv* (Dec. 2019). URL: http://arxiv.org/abs/1912.05070.

[110] Tiancai Wang et al. *Learning Rich Features at High-Speed for Single-Shot Object Detection*. Tech. rep. URL: https://github.com/vaesl/LRF-Net..

[111] Xinshuo Weng et al. “AB3DMOT: A Baseline for 3D Multi-Object Tracking and New Evaluation Metrics”. In: (2020). URL: http://arxiv.org/abs/2008.08063.

[112] Bo Wu and Ram Nevatia. “Tracking of multiple, partially occluded humans based on static body part detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 1 (2006), pp. 951–958. ISSN: 10636919. DOI: 10.1109/CVPR.2006.312.

[113] Yu Xiang et al. “Subcategory-Aware convolutional neural networks for object proposals & detection”. In: *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*. 2017. ISBN: 9781509048229. DOI: 10.1109/WACV.2017.108.

[114] Yan Yan, Yuxing Mao, and Bo Li. “Second: Sparsely embedded convolutional detection”. In: *Sensors (Switzerland)* 18.10 (2018), pp. 1–17. ISSN: 14248220. DOI: 10.3390/s18103337.

[115] Ze Yang et al. “RepPoints: Point Set Representation for Object Detection”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2019-October (Apr. 2019), pp. 9656–9665. URL: http://arxiv.org/abs/1904.11490.

[116] Zetong Yang et al. “3DSSD: Point-based 3d single stage object detector”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2020. DOI: 10.1109/CVPR42600.2020.01105.

[117] Zetong Yang et al. “STD: Sparse-to-dense 3D object detector for point cloud”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00204.

[118] Lewei Yao et al. “SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection”. In: *arXiv* (Nov. 2019). URL: http://arxiv.org/abs/1911.09929.

[119] Shanshan Zhang, Rodrigo Benenson, and Bernt Schiele. “CityPersons: A diverse dataset for pedestrian detection”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. 2017. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.474.

[120] Shifeng Zhang et al. “Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Dec. 2019), pp. 9756–9765. URL: http://arxiv.org/abs/1912.02424.

[121] Shifeng Zhang et al. “Single-Shot Refinement Neural Network for Object Detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Nov. 2017), pp. 4203–4212. URL: http://arxiv.org/abs/1711.06897.

[122] Xiaosong Zhang et al. “FreeAnchor: Learning to Match Anchors for Visual Object Detection”. In: (Sept. 2019). URL: http://arxiv.org/abs/1909.02466.

[123] Xuan Zhang et al. *Alignedreid: Surpassing human-level performance in person re-identification*. 2017.

[124] Qijie Zhao et al. “M2Det: A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network”. In: *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019* (Nov. 2018), pp. 9259–9266. URL: http://arxiv.org/abs/1811.04533.

[125] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. “Tracking Objects as Points”. In: Figure 1 (2020). URL: http://arxiv.org/abs/2004.01177.

[126] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. *Bottom-up Object Detection by Grouping Extreme and Center Points*. Tech. rep. URL: https://github.com/xingyizhou/.

[127] Yin Zhou and Oncel Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018. DOI: 10.1109/CVPR.2018.00472.

[128] Chenchen Zhu, Yihui He, and Marios Savvides. “Feature Selective Anchor-Free Module for Single-Shot Object Detection”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (Mar. 2019), pp. 840–849. URL: http://arxiv.org/abs/1903.00621.

[129] Chenchen Zhu et al. “Soft Anchor-Point Object Detection”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12354 LNCS (Nov. 2019), pp. 91–107. URL: http://arxiv.org/abs/1911.12448.

[130] Walter Zimmer, Akshay Rangesh, and Mohan Trivedi. *3D BAT: A Semi-Automatic, Web-based 3D Annotation Toolbox for Full-Surround, Multi-Modal Data Streams*. Tech. rep. URL: https://github.com/walzimmer/3d-bat..

[131] Zhengxia Zou et al. “Object Detection in 20 Years: A Survey”. In: (2019), pp. 1–39. URL: http://arxiv.org/abs/1905.05055.