POLITECNICO DI TORINO

Master's Degree in ICT FOR SMART SOCIETIES

Master's Degree Thesis

Road Elements Identification and LiDAR Integration for Advanced Driver Assistance Systems

Supervisors

Prof. ANDREA TONOLI

Prof. NICOLA AMATI

Prof. ANGELO BONFITTO

PhD Candidate STEFANO FAVELLI

PhD Candidate EUGENIO TRAMACERE

Candidate

MENG XIE

JULY 2023

Abstract

The fusion of data from multiple sensors in real-time is a critical process for autonomous and assisted driving systems, where high level controllers need classification of objects in the surroundings and estimation of relative positions. This paper presents an open-source framework to estimate the distance between a vehicle equipped with sensors and different road objects on its path using the fusion of data from camera and LiDAR. The target application is an Advanced Driver Assistance System (ADAS) which benefits the integration of the sensors' attributes to plan the vehicle speed according to real-time road occupation and distance from obstacles.

Based on geometrical projection, a low-level sensor fusion approach is proposed to map 3D point clouds into 2D camera images. The fusion information is used to estimate the distance of objects detected and labelled by a Yolov7 detector.

The open-source pipeline implemented in ROS consists of a sensors' calibration method, a Yolov7 detector, LiDAR points down-sampling and clustering, and finally a 3D to 2D transformation between the reference frames. The goal of the pipeline is to perform data association and estimate the distance of the identified road objects. The accuracy and performance are evaluated in real-world scenarios with real sensors data. The pipeline running on an embedded Nvidia Jetson AGX Xavier AI Vehicle Computer achieves good accuracy on object identification and distance estimation operating at 5Hz.

The proposed framework introduces a flexible and resource-efficient method for data association using commonly available automotive sensors. It demonstrates its potential as a promising solution for enhancing the environment perception capabilities of assisted driving systems.

Keywords: ADAS, Environment Perception, Object Detection, Sensor Fusion, Camera, LiDAR, ROS, Embedded Linux, C++, Python

Table of Contents

Chapter 1

Introduction

1.1 Background

The automotive industry has been witnessing remarkable advancements in recent years, particularly in the realms of automation and artificial intelligence. This convergence has paved the way for the emergence of Advanced Driver Assistance Systems (ADAS) and autonomous driving, with the potential to revolutionize transportation. Simultaneously, the market share of new energy vehicles (NEVs) has significantly increased, driven by the pressing need for sustainable and environmentally friendly transportation solutions.

A hybrid electric vehicle (HEV) combines an electric motor with an internal combustion engine (ICE), in order to achieve higher fuel economy and reduce greenhouse gas emissions[1]. Due to the rising fuel prices and the implementation of strict emission-related rules, the global market for HEV is expanding quickly[2]. The fuel economy can be improved significantly because electric motors are more efficient than internal combustion engines at lower speeds, and the powertrain efficiency can be improved by optimizing the use of power from engine and battery for fixed speeds, which is known as an Optimal Energy Management Strategy (Optimal EMS). The Optimal EMS is derived from the prediction of future events on the road in the driving scenario [3].

The prediction and decision making are made possible by the ADAS. The ADAS are safety systems designed to remove the human errors when driving the vehicle, and use advanced technologies to assist the driver, and thereby improve the performance of the vehicle. The ADAS platform consists of not only cameras, sensors such as RADAR, LiDAR, and ultrasonic sensors, which gives the vehicle the ability to perceive the surrounding environment, but also interfaces, and a powerful computer processor that integrates all the data and makes decisions in real-time, so the ADAS can provide information to the driver or act when necessary,

and improve the driving safety and experience[4][5].

The ADAS can be further defined as Passive and Active ADAS systems. For the Passive ADAS, the onboard computer merely informs the driver of an unsafe condition. The driver must take action to prevent that condition from resulting in an accident. Examples of Passive ADAS functions include: Forward Collision Warning, Lane Departure Warning, Blind Spot Detection, Parking Assistance, etc. For the Active ADAS, the onboard computer acts directly without the driver's intervention. Examples of Active ADAS functions include: Automatic Emergency Braking, Adaptive Cruise Control, Lane Keeping Assist and Lane Centering, Self-Parking, etc.[4].

The ADAS are one of the first steps towards autonomous driving, although fully autonomous commercial vehicles are still far from being introduced in the automotive market. The existing solutions still have technological limitations, coupled with limited trust from customers regarding the feasibility of autonomous driving in daily applications. In this scenario, the focus of the research work is to enhance the reliability of the existing systems and develop affordable, safety-oriented solutions to gradually improve public perception of autonomous driving technology.

In order to establish a general classification standard for autonomous systems, the Society of Automotive Engineers (SAE) International established its SAE J3016 Levels of Automated Driving[6] standards in 2014. The guidelines, adopted both by the United Nations and the US Department of Transportation, have been updated in 2021 and are now considered as the widely recognized and accepted standard for assessing the capabilities and responsibilities of autonomous driving systems.

As reported in Figure 1.1, the SAE Levels of Driving Automation consist of six distinct levels, each representing a progressive level of autonomy:

- Level 0 - No Automation: The driver has full control of the vehicle. Warnings and momentary assistance such as emergency braking and blind spot warning are provided, but no intervention.
- Level 1 - Driver Assistance: The vehicle incorporates basic driver assistance features of steering or speed management support, such as adaptive cruise control or lane-keeping assistance. However, the driver remains fully responsible for vehicle operation and must monitor the driving environment.
- Level 2 - Partial Automation: The vehicle can control both steering and speed management under specific conditions. The driver is still responsible for monitoring the driving environment and must be ready to take control at any time. This is made possible by ADAS.
- Level 3 - Conditional Automation: The vehicle can manage most aspects of the driving task under specific conditions. However, the driver must be

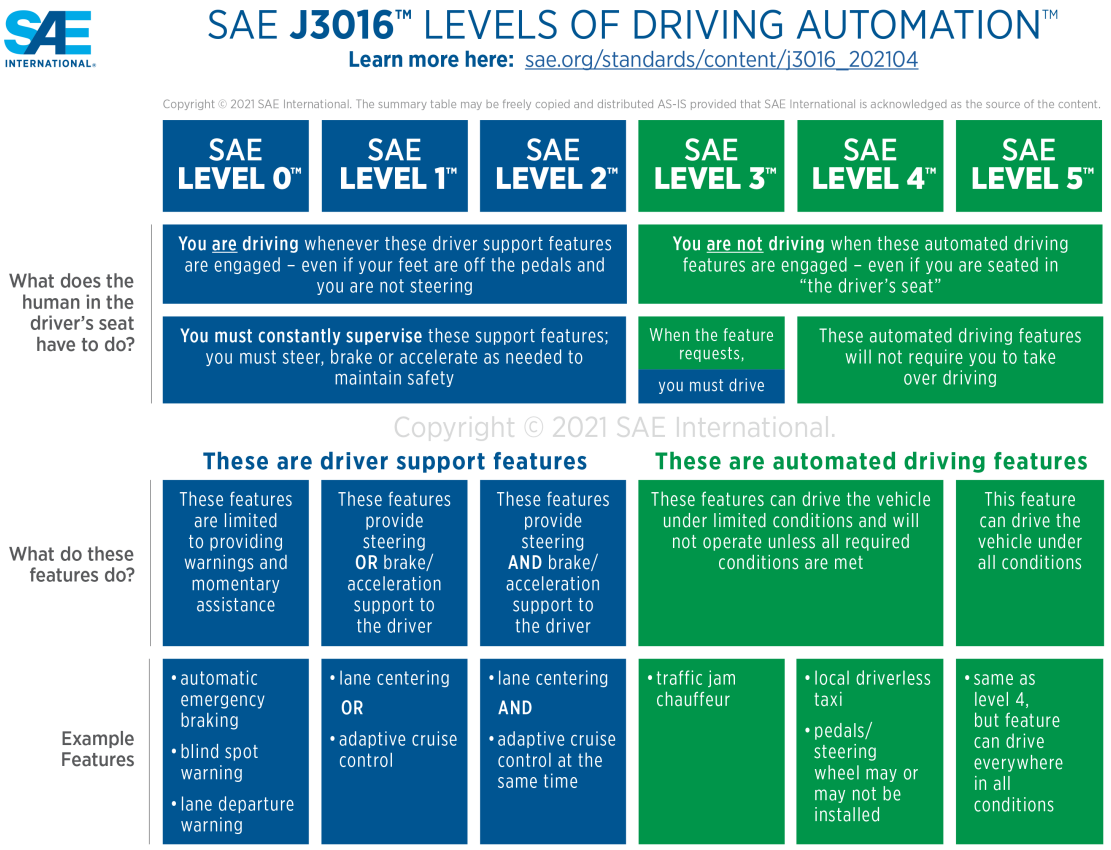

Figure 1.1: SAE Levels of Driving Automation™

prepared to intervene when alerted by the system to resume control.

- Level 4 - High Automation: The vehicle can perform all driving tasks within defined operational domains and conditions without driver intervention. However, the system may require the driver to take over in exceptional circumstances.

- Level 5 - Full Automation: The vehicle is capable of performing all driving tasks under all conditions, and the driver is not required to be involved in the driving process. Level 5 vehicles are fully autonomous and do not require human intervention.

This thesis work is a part of the PITEF – AutoECO project, which aims at building a hybrid light duty vehicle equipped with P1 Electric Motor and ADAS sensors. This thesis focuses on the perception of surrounding environment of the vehicle, in particular the road elements detection and distance measurement. So,

the ADAS can have enough information to make decisions about the optimal energy management.

1.2 Motivation

The main motivation for this thesis work is to design a mechanism to acquire the important information about the surrounding environment, especially the traffic information in front of the vehicle, such as other vehicles, traffic light states, speed limits and other traffic signs. There are several solutions for environment perception in assisted driving applications, but each solution has its own strengths and weaknesses. Some of the most used ones are:

- LiDAR (Light Detection and Ranging): It emits pulsed light waves into the surrounding environment, and these pulses bounce off surrounding objects and return to the sensor, then the sensor uses the time it took for each pulse to return to the sensor to calculate the distance it traveled. Repeating this process millions of times per second creates a precise, real-time 3D map of the environment. This 3D map is called a point cloud, which helps in detecting objects like cars, pedestrians, and road infrastructure, and measure their distance[7].
- Computer Vision: It is a field of artificial intelligence (AI) that enables computers to derive and understand meaningful information from digital images, videos, and other visual inputs in the same way that humans do. It uses cameras and machine learning algorithms to interpret visual data and detect objects in real-time. However, it can be blinded by dirt, sun, rain, snow or darkness[8][9].
- Radar (Radio Detection and Ranging): It uses radio waves to detect objects and measure their distance, speed, and angle relative to the vehicle. It transmits a radio signal and measures the time it takes for the signal to bounce back after hitting an object, then it calculates the distance to the object, and by using the Doppler Effect, according to the difference in frequency between the transmitted and received signals, it calculates the speed of the object[10].
- Ultrasonic Sensor: It emits ultrasonic sound waves, and converts the reflected sound waves into an electrical signal, then measures the time taken for the bounce back to determine the distance to an object. It is particularly useful in low-light or adverse weather conditions, where other sensors, such as cameras and LiDAR, may struggle[11].
- GPS (Global Positioning System): GPS is a satellite-based navigation system that provides accurate and reliable positioning information. It uses satellite

signals to determine the precise location and velocity of the vehicle in real-time[12].

To ensure the accuracy and efficiency of environmental perception, the solution of sensor fusion is employed by the ADAS. Sensor fusion, like how the human brain process information, combines large amounts of data from camera and different sensors with the help of image recognition software, algorithms to process point cloud and range information from LiDAR and RADAR, which is more accurate than the information obtained by individual sensors. This technology provides a more robust and reliable perception of the surroundings, and can physically respond faster than a human driver ever could. It can analyze streaming video in real time, recognize what the video shows, and determine how to react to it[13].

1.3 Overview of the PITEF – AutoECO Project

The PITEF – AutoECO project is a research project funded by the government of Piedmont Region, where PITEF stands for Piattaforma Tecnologica di Filiera, which means Supply Chain Technology Platform[14]. The project aims to promote and validate on a demonstrator vehicle, an integrated system consisting of a hybrid drive module and a control unit that allows to assess the potential benefits and to exploit the information available from the ADAS sensors, in optimizing energy management control and improving energy efficiency.

As shown in Figure 1.2, there are many partners involved in this project. The road map consists of three steps:

- Step 1: Assess the benefits on the fuel economy given by the integration of a P1 Hybrid Architecture.
- Step 2: Assess the further reduction potential introduced by the equipment of ADAS sensors.
- Step 3: Design, Integration and Validation of Control Logics to leverage the Hybrid Architecture and the ADAS information to reduce the fuel consumption.

1.4 Thesis Outline

This thesis work is structured as follows:

- Chapter 2 presents the theoretical background of the topics presented, with a particular focus on computer vision and sensor fusion techniques.

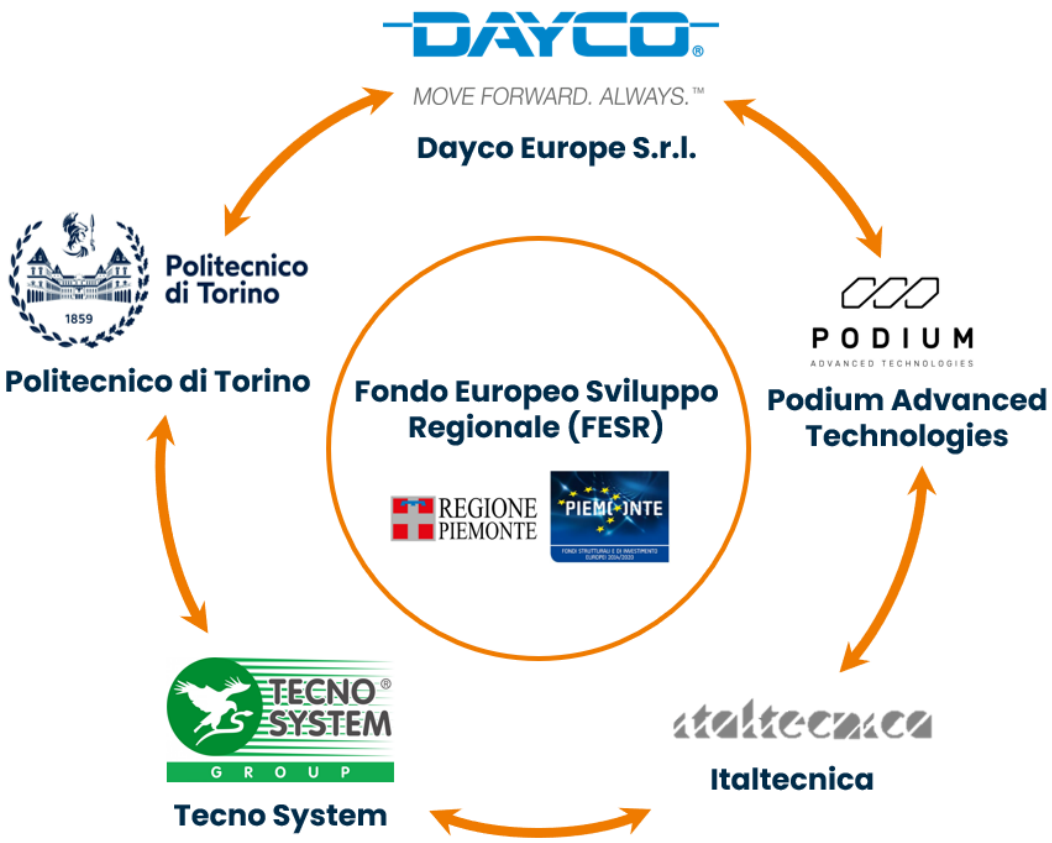

Figure 1.2: PITEF Project

- Chapter 3 is dedicated to the design of system architecture and the implementation of the proposed method. Starting from the hardware architecture overview and then the development of the sensor fusion algorithms is discussed, with particular focus on implementation and integration of hardware and software.
- Chapter 4 presents the setup of experimental validation process and the evaluation of the obtained results.
- Chapter 5 is the final chapter, where conclusions and future works are reported.

Chapter 2

Theoretical Background

Before the discussion of the proposed method, this chapter is focused on the introduction of the theoretical background of this thesis work. With the continuous development of the autonomous driving industry, many research and solutions have sprung up. Although the purpose of our project is not to achieve autonomous driving, the basic principles are very similar in terms of the environmental awareness and data processing. First, we will introduce the common environmental perception solutions in the autonomous driving sector, and define the scenario of our project. Then, we will investigate the working principles of necessary ADAS sensors to acquire the data, such as camera and LiDAR, and the state-of-the-art computer vision algorithms to process the data from camera. Finally, we will discuss about the sensor fusion techniques for the processing methods.

2.1 Environmental perception

To be able to optimize the energy management for a hybrid vehicle driving in a real-world traffic scenario, it is fundamental to have a comprehensive perception of the surrounding environment of the vehicle. An autonomous vehicle acquires knowledge of its surrounding by scanning the road ahead to extract road features, detect road objects and predict their behaviors. The road features consist of the infrastructures such as lanes, pedestrian crossing, barriers, traffic lights, speed limits and stop signs. The road objects are the dynamic roles such as other vehicles and pedestrians[15][16].

The operation scenarios defined in our project have lower requirements than those in the autonomous vehicle industry. What we want to achieve is to identify the road objects such as vehicles, traffic lights and speed limit signs, and detect the relative distances of the objects from our vehicle, and read the information of the traffic lights and road signs. Therefore, the computer vision algorithm for the

object detection and sensor fusion algorithm are the key technologies to realize our goal.

2.2 Object Detection

Over the past two decades, object detection technology has mainly gone through two periods of development. The first period is the traditional methods of target detection using artificially designed features (such as Haar, HOG, LBP, etc.), combined with traditional machine learning algorithms (such as SVM, Boosting, etc.) for target detection. Although these features are not specifically designed for vehicle detection, and the effect of this type of algorithm is average, this period has laid the foundation for target detection. The second period is the introduction of deep learning technology, where convolutional neural network (CNN) is used to learn feature representations and object classifiers. The earliest deep learning target detection algorithm Region-based CNN (R-CNN) uses a combination of CNN network and SVM classifier, which can achieve high detection accuracy.

Due to the different structure of neural networks and the feature learning process, the deep learning object detection algorithms are divided in two directions, two-stage object detection and one-stage object detection.

The two-stage detection algorithm, such as R-CNN, divides the detection process into two stages: region proposal and object classification. The region proposal stage involves extracting a set of regions from the image that may contain objects. These regions are often called regions of interest (ROI) or region proposals. The typical method to generate the ROI is called sliding window-based method, which scans the image at different scales and positions, and generates an object probability for each window to obtain a set of candidate boxes. In the object classification stage, for each candidate box, we need to classify whether it contains an object or not. This usually involves designing a classifier that can take the candidate box as input and output a probability value indicating whether the box contains an object. Common classifiers include Support Vector Machine (SVM), Convolutional Neural Network (CNN), etc.

Fast R-CNN is an upgraded algorithm of R-CNN. Its innovation lies in the ROI pooling layer, which can map candidate regions of different sizes to fixed-size feature maps, thus avoiding the process of convolution feature extraction for each candidate region separately. In addition, Fast R-CNN uses SoftMax layer instead of SVM classifier in R-CNN to improve the speed and accuracy of detection results.

Faster R-CNN further proposed the RPN (Region Proposal Network) module, which replaces the traditional region selection method by extracting candidate boxes on the convolution feature map, and uses shared convolution features and Smooth L1 loss function, which greatly improves the speed and accuracy of the

detection algorithm.

Region-based Fully Convolutional Networks (R-FCN) improved detection performance based on Faster RCNN and FCN with faster speed and less computation. The main contribution of R-FCN is to introduce position-sensitive score maps to replace the fully connected layer in the region-based detection network. The position-sensitive score maps are generated by partitioning the final convolutional feature maps into grids, where each grid cell is associated with a score map that encodes the position information. By using the position-sensitive score maps, the object detection network can efficiently classify object regions with shared computation. This design reduces the computational cost and enhances the detection accuracy at the same time.

In conclusion, the two-stage detection algorithms have been widely used and have accurate detection results in vehicle detection. However, due to the slow and complex region proposal generation stage, they cannot achieve real-time performance[17].

On the other hand, the one-stage detection algorithms directly predict the presence and location of objects in an image without the region proposal step. One-stage detection algorithms typically use a single convolutional neural network (CNN) to predict the class probabilities and bounding boxes for all possible locations in an image. One-stage detection algorithms usually use anchor boxes, which are pre-defined bounding boxes of different sizes and aspect ratios, to handle the variations in object sizes and shapes. The anchor boxes are used to divide the image into a grid of cells, and each cell is responsible for predicting the bounding boxes of the objects that are centered within it, and the network predicts the class probabilities and offsets for each anchor box at each grid cell. The main advantage is the high detection speed. The one-stage detection models are more efficient, easier to optimize, and more suitable for real-time devices. Examples of popular one-stage detection algorithms include YOLO (You Only Look Once), SSD (Single Shot Detector), and RetinaNet. [16]

YOLO uses a deep convolutional neural network (CNN) to extract features from the image, and then applies several convolutional and fully connected layers to generate the final output. YOLO performs convolutional calculation on the entire image, so it has the advantage of a larger field of view (FOV) during the detection, and it is not easy to misjudge the background. The output of YOLO is a set of bounding boxes, each with a corresponding class probability. YOLO is known for its speed and accuracy, achieving state-of-the-art results on several object detection benchmarks, however, it may not perform well on small objects[17][18].

SSD transforms the inspection task into a unified end-to-end regression problem, and obtains both location and classification through only one process. The idea of transforming detection into regression was inherited from YOLO. SSD works by applying a set of predefined anchor boxes to an image to generate multiple bounding

boxes at different scales and aspect ratios, and then performs classification and localization on each bounding box, while YOLO uses a single set of anchor boxes for the entire image. SSD's advantage over other one-stage object detection algorithms is that it can detect objects at multiple scales of an image, better handling objects of different sizes[17][19].

RetinaNet is a state-of-the-art object detection algorithm that was introduced in 2017 by a team of researchers at Facebook AI Research. RetinaNet uses a focal loss function to address the class imbalance problem in object detection, which occurs when there are many more background regions than object regions. This can avoid the bias towards background regions and have higher detection accuracy for smaller objects. In terms of the use of anchor boxes, RetinaNet adopts the same approach as the SSD. RetinaNet may be a better choice for applications that require high detection accuracy, while YOLO may be more suitable for real-time applications on low-power devices.

In this work, since our use case is a real-time object detection application running on a low-power onboard device, we eventually choose Yolo v7, the newest version of the Yolo family, when we started to design our system[17][20].

2.3 Working Principles of ADAS sensors

2.3.1 Camera

Type of sensors

The camera is an optical device that captures visual information with high resolution. Like the human eye, most cameras capture light in the visible spectrum, while some special cameras capture other parts of the electromagnetic spectrum, such as infrared. The camera consists of a lens that captures the light in the environment and focuses it on an image sensor, usually a CMOS (complementary metal oxide semiconductor) or CCD (charge-coupled device) sensor. The sensor converts light into electrical signals, which are then processed by the onboard processing unit of the camera to generate images.

Both CCD and CMOS image sensors convert light into electrons by capturing light photons with millions of photosensitive sites and converting them into an electrical signal. Then the sensors quantify the accumulated charge of each photosensitive site in the image, and from here the technologies start to differ. A CMOS sensor is a digital device, while a CCD sensor is an analog device. The CMOS sensor is made up of an array of photosensitive sites, each contains a photodiode, which absorbs light and generates an electrical charge that is proportional to the amount of light that falls on it. There are several transistors at each photosensitive site to amplify and read out the electrical charge, and the charge is converted to

a voltage signal, then the signal is multiplexed by row and column to multiple on-chip, digital-to-analog converters. On the other hand, the CCD sensor converts light into electrons. It is a silicon chip that contains an array of photosensitive sites. The output is immediately converted to a digital signal by an analog-to-digital converter. The voltage is read from each site to reconstruct an image.

Both technologies are commonly used in digital cameras, each has its own merits. Here are some of the key differences between CMOS and CCD:

- Image quality: CCD sensors produce high-quality, low-noise images with better color accuracy and dynamic range, because they have high sensitivity to light, and each photosensitive site is capable of capturing a large number of electrons, so they produce less noise in low-light conditions, while CMOS sensors have high, fixed-pattern noise and low sensitivity, as there are several transistors located next to each photosensitive site and some photons might hit the transistors instead of the photosensitive site. However, CMOS sensors have made significant progress in recent years and are now capable of producing high-quality images that rival those of CCD sensors.
- Speed: CMOS sensors are generally faster than CCD sensors because each photosensitive site directly converts the electrical charge into a voltage signal, so the signal can be read out in parallel, while CCD sensors require a sequential readout of each photosensitive site. This advantage makes CMOS sensors well-suited for applications that require high-speed imaging, such as sports photography and smart driving.
- Power consumption: CMOS sensors are typically more power-efficient than CCD sensors because they only need to power the photosensitive site that are being read out, which can significantly reduce power consumption compared to CCD sensors, which require constant power to maintain the charge on each photosensitive site. A CCD sensor consumes as much as 100 times more power than an equivalent CMOS sensor.
- Manufacturing cost: CCD sensors rely on specialized fabrication that requires dedicated and costly manufacturing processes, while CMOS sensors can be manufactured on most standard silicon production lines, so CMOS sensors cost much less than CCD sensors.

In conclusion, the choice between CMOS and CCD sensors depends on the specific application and the requirements of the user. However, CMOS sensors have significant advantage over CCD sensors in terms of manufacturing cost, power consumption and processing speed, and they are rapidly improving in image quality. These advantages have led to the wide adoption of CMOS sensors in a variety of imaging applications, including digital cameras, mobile phones, and automotive cameras[15][21][22][23][24].

Pinhole Camera Model

The pinhole camera model is a simple mathematical model that describes the relationship between a point in the 3D world and its projection on the image plane of an ideal pinhole camera. It assumes that light enters the camera through a small aperture or so-called pinhole, and projects an inverted image on the image plane behind the pinhole, where the camera aperture is described as a very small point with no lens to focus the light.

In this model, the camera is represented by a single point, known as the camera center or optical center. The image plane or focal plane is assumed to be perpendicular to the optical axis of the camera, which is a straight line called the principal axis passing through the center of the pinhole and the center of the image plane. The distance between the image plane and the pinhole is the focal length. The plane through the camera center parallel to the image plane is called the principal plane of the camera.

Sometimes, the image plane is placed between the optical center and the 3D object at distance of the focal length from optical center. In this case, it is called the virtual image plane or virtual retinal plane. The pinhole camera model is shown in the following figure 2.1[25]:

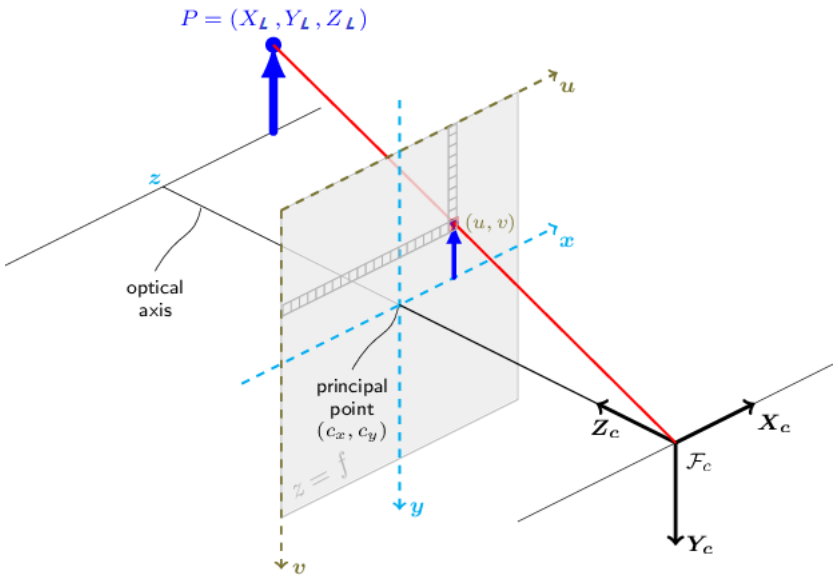

Figure 2.1: Pinhole Camera Model

To capture an image using the pinhole camera model, each point in the 3D world is projected onto the image plane by drawing a line from the point through the pinhole and onto the image plane. By similar triangles, this projection results in an inverted image of the world on the image plane, with objects farther away

from the pinhole appearing smaller than objects closer to the pinhole.

The pinhole camera model can be used to calculate the 2D coordinates of each point in the image, given its 3D coordinates and the intrinsic parameters of the camera, which include the focal length of the lens, the position of the pinhole, and the size of the image sensor. It can also be used to estimate the camera's extrinsic parameters, which describe the position and orientation of the camera relative to the 3D world. The detailed approach will be introduced in the methodology section[26][27].

Monocular Camera vs Stereo Camera

A monocular camera uses a single lens to capture 2D images or videos. The working principle of a monocular camera is introduced above. It is a common type of vision sensor used in autonomous vehicles. It is smaller and cheaper than the stereo camera, and can be easily mounted and integrated into different systems. However, all real scenes encountered by the camera are three-dimensional. Objects at different depths in the real world may appear adjacent to each other in the two-dimensional mapping world of the camera sensor. The human brain has the ability of perspective, which allows us to determine the depth from the two-dimensional scene. For a front camera in the car, the ability to analyze the relative distance is not easy.

In order to estimate the distance of objects in the scene, the stereo camera was invented. A stereo camera uses two lenses to capture images or videos from two different viewpoints. By capturing two images of the same scene from slightly different perspectives, stereo cameras can create 3D images or videos that provide depth information. This allows the camera to simulate human binocular vision and therefore it can sense depth. This is accomplished by analyzing the disparity between the two images captured by the camera, which is the position difference between corresponding pixels in the two images, and then by using triangulation method, the algorithm can calculate the distance of objects in the scene. Although this method is effective in many environments, stereo vision is limited by the baseline distance between the two cameras. Especially when the detected object is far away, the depth estimation is often inaccurate, because even very small triangulation or angle estimation errors can be translated into very large errors in the distance.

To further discuss about the triangulation method, the concept of disparity needs to be clarified in detail. After the left and the right camera of the stereo camera are calibrated, we can calculate the distance to an object by finding the disparity between the images captured by the left and right cameras for that point at the same time. The disparity is defined as the number of pixels that a particular point has moved in the right camera image compared to the left camera image. An

object is projected onto different locations on the image plane of the two cameras, depending on the distance of the object. The stereo disparity varies with object distance, and is inversely proportional to the distance of the object.

The stereo correspondence usually gives a reliable estimate of disparity, unless most of the image are featureless, and no correspondence can be found. The accuracy depends on the baseline distance between the two cameras. In general, for a given baseline distance between cameras, the accuracy decreases as the depth value increases. This is because small errors in disparity can be translated into huge errors in depth estimation. In the range of very distant objects, there is no observable disparity, and depth estimation usually fails. According to experience, when the depth exceeds a certain distance, the depth estimation from the stereo vision often becomes unreliable.

There are some methods to increase the maximum detection range and improve the accuracy, such as physically increase the distance between the two cameras or increase the focal length. However, it is not feasible to change a physical attribute of a well-designed and calibrated stereo camera, and the space allowed to mount a stereo vision system on a vehicle is very limited. Therefore, we can only optimize the system on a software level, such as implement algorithms to calculate stereo disparity at the sub-pixel level, which leads to increased complexity of the system and additional consumption of computing power.

In conclusion, as for our use case, the depth sensing function of our stereo camera can only provide a baseline for our distance estimation task, and we cannot only rely on the stereo camera to measure the precise distances of the detected objects. This is the chance for the LiDAR to be effective[28][29].

2.3.2 LiDAR

LiDAR is short for light detection and ranging. It is a remote sensing technology that works by emitting eye-safe laser beams towards an object or surface, and then measuring the time it takes for the laser pulse to bounce back to the sensor to create a 3D representation of the surveyed environment that can be used to better understand and manage the world around us. LiDAR technology can be used in a wide range of applications, including automotive, infrastructure, robotics, trucking, UAV, industrial, mapping, and many more. Because the light source of LiDAR is itself, the technology offers strong performance in a wide variety of lighting and weather conditions.

The laser emission is in the infrared spectrum, typically with a wavelength of around 905 nanometers or 1550 nanometers. The use of infrared light allows LiDAR systems to penetrate through haze, dust, and other atmospheric conditions that can interfere with visible light. The short pulses of light emitted by LiDAR lasers typically last only a few nanoseconds, and can be repeated many times

per second to gather a large amount of data about the environment. There are differences between the two wavelengths in terms of safety, water absorption, and power consumption.

Water can affect LiDAR signal integrity, which is important for automotive LiDAR systems given the adverse weather conditions vehicles might encounter on the road and the safety of the human eyes. The energy of laser beams begins to be absorbed by water from the wavelength of 1400 nanometers. As a result, 1550 nm waves may experience significant signal degradation under conditions of rain, fog or snow compared to 905 nm waves, but this inherent disadvantage makes it possible for the aqueous liquid of human eyes to filter out the wavelength, making the 1550 nm beams less harmful than the 905 nm beams. However, we need to emphasize that sensors using 905 nm and 1550 nm wavelengths achieve eye-safety certification via compliance with the FDA eye-safety standard IEC 60825. If sensors are designed to meet eye-safety standards, both wavelengths can be used safely. The degradation in rain, snow and fog conditions of 1550 nm wavelength means LiDAR sensors using 1550 nm will need 10 times more power as compared to a similar 905 nm system. This may demand for larger systems, including additional power and cooling components that must be stored in a vehicle. To put it another way, to offset degradation and to achieve longer range, 1550 nm systems need to send out more laser light to achieve performance comparable to 905 nm systems. As a result, 1550 nm systems typically consume more electrical power.

From the engineering point of view, the 905 nm LiDAR technology is a better fit for ADAS and autonomous vehicles.

There are two kinds of LiDAR, mechanical rotation LiDAR and solid-state LiDAR. Mechanical rotation LiDAR uses a rotating mirror or a moving lens to scan the environment with laser pulses. This method has been used for decades and is effective in many applications, but it is relatively bulky and expensive. Additionally, the moving parts are subject to wear and tear, reducing the lifespan of the device. Currently, the common mechanical rotation LiDARs on the market ranging from 16 channels to 128 channels with a horizontal FOV of 360 degrees and vertical FOV between 20 to 45 degrees. In contrast, solid-state LiDAR uses an array of lasers, typically semiconductor lasers, to emit laser pulses without any moving parts. The lasers are mounted on a fixed substrate and directed using an optical beam-steering system, such as micro-electro-mechanical systems (MEMS). This approach eliminates the need for bulky and expensive mechanical components, reducing the size, weight, and cost of the device. Additionally, solid-state LiDAR can emit laser pulses at a much higher rate, allowing for more detailed and accurate scanning of the environment, and it has a lower power consumption and can operate at higher temperatures. Solid-state LiDAR has a smaller FOV than the mechanical rotation LiDAR, typically of 120 degrees. The trend in perception system is replacing the current mechanical rotation LiDAR by a set of solid-state

LiDARs integrated around the vehicle[7][15][30][31][32][33][34][35][36].

There are some terminologies need to be clarified when we evaluate the performance of a LiDAR. This is very important for us to choose the appropriate LiDAR for different application scenarios, as the industry has not yet established a unified standard.

The first concept is related to the distance. In the ADAS or autonomous driving scenario, the farther a LiDAR can see, the more time is allowed for the system to prepare and make decisions. However, there are differences between the concepts of detection distance and maximum range. The maximum range is the farthest detection distance without restrictions. On the other hand, the detection distance usually refers to the ranging capability of a LiDAR, which means the farthest detection distance under standard working conditions.

To define the standard working conditions, one of the essential standards is the 10% reflectivity target, which leads us to the second concept, the reflectivity. It is defined as the degree to which an object reflects light. For an object made of the same material, the color and smoothness of their surface can affect reflectivity. White has a higher reflectivity than black, and smooth surfaces have a higher reflectivity than rough surfaces. Objects with higher reflectivity are more easily perceived. However, in the real-world scenario, the ability to perceive object with low reflectivity is more meaningful. There are certain materials with a known reflectivity of 10%, meaning that it reflects approximately 10% of the light incident on its surface. For example, a black tire is a typical 10% reflectivity target. During the calibration or performance testing of the LiDAR system, a target with a 10% reflectivity is placed at a known distance from the sensor. Then, the sensor emits laser pulses toward the target, and captures and analyzes the reflected signal. The accuracy and consistency of the measured distance and signal strength from the target can help evaluate the performance of the LiDAR system and can be used to adjust and fine-tune the system parameters if necessary. When evaluating the performance of LiDAR systems from different manufacturers or applications, the 10% reflectivity target can be used as a standardized reference to ensure comparability and consistency, providing a reliable benchmark for evaluating the system's capabilities.

Normally, the maximum range of the same LiDAR is greater than the ranging capability. For example, a LiDAR that claims to have a maximum detection distance of up to 400 meters may only have a ranging capability of 200 meters when the reflectivity is limited to 10%. In conclusion, the ranging capability is a more meaningful indicator for assisted driving.

Finally, the concept of resolution is introduced to evaluate if the LiDAR can see things clearly. Like cameras, an image is composed of a matrix of pixels. The denser the pixels in an image, the higher the resolution. LiDAR can be seen as a three-dimensional camera, and the resolution is evaluated by the density of the

three-dimensional pixels. The density is the number of point clouds generated by the LiDAR per second, i.e., the point frequency. The higher the point frequency, the higher the resolution, providing a clearer view and bringing safety to the ADAS system.

2.3.3 Sensor Fusion Techniques

Using a single type of sensor to achieve environmental awareness has proven to be insufficient and unreliable. Therefore, in order to overcome these limitations, it is necessary to adopt sensor fusion technology. Due to the integration of information from multiple sensors, sensor fusion improves the reliability and accuracy of measurement, and reduces the uncertainty of results.

Sensor fusion between LiDAR and camera refers to the process of combining data from both LiDAR and camera sensors to obtain a more accurate and robust perception of the environment. The camera is responsible for identifying and classifying objects, while the LiDAR is responsible for measuring the exact distance of objects. The combination of the information from the camera and the LiDAR adds redundancy and certainty to the decision-making stage.

There are many studies on different implementation ideas for sensor fusion, and here we review these approaches through the levels of sensor fusion, depending on the complexity and degree of integration of the sensors. In fact, when we make decision on the implementation of sensor fusion, we need to answer the following three questions:

When should the fusion occur?

The answer to this question is focused on the manipulation of the data. From this point of view, the fusion is divided by abstraction level. In the industry, it is further divided into low-level, mid-level, and high-level fusion.

The low-level fusion, also known as sensor level fusion, combines the raw data from multiple sensors. The most common approach is to project the LiDAR point cloud onto the 2D camera image, then to check whether the projected points belong to 2D bounding boxes detected with the camera. The advantage is that it takes all the data into consideration, so it has the potential to support algorithms that utilize information more comprehensively, making detection results more accurate. The disadvantage is that it requires very high computational resources, as it needs to project hundreds of thousands of points onto a 2D plane in a few milliseconds.

The mid-level fusion, also known as feature level fusion, combines the outputs of objects detected independently by each sensor. The idea is to detect objects from the 3D point cloud directly to get 3D bounding boxes, and then fuse them with the 2D bounding boxes detected with the camera. The advantage is that the working

process of each sensor is more intuitive and easier to understand. The disadvantage is that it relies heavily on the performance of each detector. If the detection of one sensor is too noisy or even failed, the entire fusion might fail. Although it is possible to adopt algorithms such as Kalman Filter to reduce the noise and improve accuracy, it still requires high computational resources and increases the complexity of the system.

The high-level fusion, also known as decision level fusion, uses the resulting data to make decisions and predictions about the environment, which involves the tracking of the detected objects, i.e., the prediction of their trajectories. The advantage is the same as the above-mentioned mid-level fusion. The disadvantage is the output result of each sensor is more abstract as most of the information is hidden in the process. If one tracking is wrong, the entire tracking task is failed.

Where should the fusion happen?

The answer to this question is mainly focused on the design of hardware architecture and dataflow. In theory, the information from all the sensors must be combined together to make a final decision, but there are different physical and logical arrangements of data processors in different architectures. In this case, the fusion can be further categorized into centralized fusion, decentralized fusion, and distributed fusion.

The centralized fusion adopts a central control system to process data from all the sensors and realize the fusion. The sensors either process the raw data on its own computer and send the abstract information to the central control system, or send raw data to the central control unit directly, and the system is responsible for the processing of different types and formats of data. For example, Aptiv, an Irish-American automotive technology supplier, developed a sensor fusion system called the Satellite Architecture, which treats the sensors mounted on different positions as satellites, and the main computer fuses the data to realize a 360° detection. It is a highly flexible and scalable approach, and reduces the weight of the vehicle and the complexity of the system at the same time.

As opposed to the centralized fusion concept, the decentralized fusion and distributed fusion let sensors to fuse data from other sensors locally, then forward the fused data to the next sensors. For example, if a car has LiDAR, Radar and camera, it first fuses the data between LiDAR and camera, and between Radar and camera separately, then combine the two fused data together to get the final detection. This is an approach adopted by the classic architecture, because it is easier to fuse data between two sensors in terms of complexity of algorithm and computation, rather than deal with data of different format and rate at the same time.

What should the fusion do?

This question is mainly about the objective of the sensor fusion. If the sensors are used for the same purpose, for example, when we use a Radar and a LiDAR at the same time for distance measurement, we introduce redundancy to improve the accuracy.

If we want to build a panorama with multiple cameras, the sensors look at different directions to get the full picture. Since the sensors complete each other, this can be called a complementary fusion.

If we want to do a 3D reconstruction or 3D Scan with multiple 2D sensors, we are using two or more sensors to produce a new scene, and it can be called a coordinated fusion[17][37][38][39][40][41][42][43][44][45][46].

Chapter 3

System Architecture Design and Methodology

This chapter focuses on the overall architecture design and actual implementation of our road environment awareness system. As introduced in the theoretical background reviewed in the previous chapter, there are solid scientific theories and mathematical basis to support the working principles of sensors for data acquisition and the workflow of sensor fusion for environmental perception. In order to successfully achieve the objectives of this project, we need to first define the core requirements of the task, then design the overall architecture of the system, and select appropriate hardware and software solutions, and finally deploy the entire system on the vehicle for testing.

The first section is dedicated to the requirement analysis. As the project studied in this thesis is a part of a complete ADAS system, we need to define the task objectives specifically applicable to our use case based on upstream and downstream constraints and requirements in the workflow, rather than proposing a general sensor fusion solution.

The second section introduces the architecture design of the hardware. While ensuring that the project requirements are met, we fully weighed the feasibility, efficiency, scalability of the system, as well as the differences between the laboratory and the industrial environment, then we choose the most suitable hardware equipment from the available resources at hand. The key parameters, operating environment, dependency libraries, instructions, and workflow of the selected hardware will be emphasized.

The third part introduces software architecture design and algorithm development, which is also the core of this project. Firstly, we will briefly introduce the working principle of ROS environment, the creation and usage of ROS packages, followed by the working principle and training process of the YOLO algorithm,

including the joint calibration method of the camera and LiDAR, and finally the design and implementation of the sensor fusion algorithm.

The fourth part introduces the deployment plan of the entire system on the testing vehicle, the selection of testing environment, and the collection of key data.

Finally, an overview of the whole pipeline of the system is presented to explain its integration with the complete ADAS system.

3.1 Requirement Analysis

Requirement analysis is the process of gathering and defining the functional and non-functional requirements of a software system. It is a critical step in the software development process, as it forms the basis for all subsequent development activities, including design, implementation, and testing. This process includes collecting and documenting requirements, analyzing, and prioritizing them, and defining the scope and objectives of the system. It helps to ensure that the system meets the needs and expectations of its stakeholders, providing a clear understanding of what the system must do and how it should perform.

As the system is developed to realize the environmental awareness function in an ADAS system, it must be capable of accurately detecting and interpreting the surroundings of the vehicle in real-time to provide necessary information to the ADAS system, and even advanced warning to the driver in case of potential dangers.

Given that our main function is to detect objects in front of our own vehicle in the current lane and adjacent lanes, such as other vehicles, pedestrians, traffic lights, and road signs related to speed control, etc., and to measure the relative distance between the detected objects and our own vehicle. We decided to utilize camera and LiDAR to detect and analyze the environment surrounding the vehicle. In addition, the system must be capable of processing large volumes of data in real-time, meanwhile, it must have low energy consumption and compact size.

As the system will be installed on a vehicle and tested in actual road scenarios, the robustness is also very important. The system must have a high level of accuracy and precision, and be able to function reliably, in detecting and interpreting objects in the environment, even in challenging weather conditions.

3.2 Hardware Architecture Design

Given the constraints that the whole system will be mounted on a hybrid light duty vehicle and will be integrated with the existing vehicle's system, the hardware equipment needs to be powerful in computation, efficient in power consumption, and compact in size, including the ECUs (Electronic Control Unit) for central

control and environment perception, the sensors such as LiDAR and camera, as well as other devices such as the monitor for visualization, and the batteries for power supply during the testing phase.

3.2.1 NVIDIA Jetson AGX Xavier

NVIDIA Jetson AGX Xavier is a high-performance computing platform designed specifically for autonomous machines and AI applications. It is an embedded system-on-module that combines a powerful NVIDIA GPU, CPU, and deep learning accelerators, making it one of the most advanced and capable embedded edge devices available.

The Jetson AGX Xavier platform is based on the NVIDIA Xavier SoC (system-on-chip) which integrates an 8-core NVIDIA Carmel ARM® v8.2 64-bit CPU, a 512-core NVIDIA Volta™ GPU with 64 Tensor Cores, two NVDLA (NVIDIA Deep Learning Accelerator) engines and two PVA (Programmable Vision Accelerator). It also includes 32 GB of LPDDR4x memory, 32 GB of eMMC 5.1 storage, and support for high-speed connectivity such as HDMI, USB 3.1, PCIe, and Gigabit Ethernet interfaces.

While it offers an excellent workstation performance with up to 32 TOPS (Tera Operations Per Second) of AI performance, it has a size of only 10 percent of a normal workstation, and a configurable power mode at 10W, 15W, and 30W according to application needs, which makes it an ideal onboard device for a vehicle.

Figure 3.1: AI Vehicle Computer RSL A3

To ensure a long-term reliable use, we choose the AI Vehicle Computer RSL A3, as shown in Figure 3.1, from Syslogic, a leading tech company in the embedded industry, which combines the NVIDIA Jetson AGX Xavier module with its own

carrier board and a specifically designed robust housing made of aluminum and stainless steel, resistant to shock and vibration. The device adopts a fanless passive cooling system and is capable of operating in temperatures ranging from –25 to +65 degrees Celsius.

The operating system of the platform is NVIDIA Linux for Tegra (L4T 35.1.0) Ubuntu 20.04, JetPack 5.0.2 and ROS Noetic, which have been chosen to ensure the compatibility of various dependencies required by the state-of-the-art Yolo algorithm, the sensor fusion algorithm, and the ROS packages of the LiDAR and camera sensors. One thing to be noticed is that the CPU architecture is aarch64, so most of the libraries installed to support the hardware and software that will be introduced in the following chapters are specifically designed for this architecture, not the x86_64 or amd64 version.

3.2.2 ZED2

The ZED2 camera is a stereo camera that provides high-definition 3D video and neural depth perception of the environment. It has been designed for the most challenging applications, from autonomous navigation and mapping to augmented reality and 3D analytics. It is manufactured by Stereolabs, a company that specializes in producing cameras and software for 3D sensing applications.

The ZED2 camera is designed to replicate the way human vision works. As shown in Figure 3.2, by using its two CMOS image sensors and triangulation, it captures stereo images and generate depth maps with a high-resolution video output up to 2.2K, and provides a three-dimensional understanding of the observed scene. It has a wide FOV of up to 120 degrees, which enables it to capture a large area in a single shot. It also uses neural networks to reproduce human vision, bringing stereo perception to a new level.

Figure 3.2: ZED2

The ZED2 has the most complete built-in sensor stack. Featuring next-generation

IMU, barometer, magnetometer, and temperature sensors, it captures elevation and magnetic field data in real-time. As camera heating induces changes in focal length and motion sensors biases, it adopts a more robust all-aluminum enclosure with thermal control to monitor temperature and compensate these drifts, allowing it to be capable of operating in temperatures ranging from –10 to +50 degrees Celsius.

The ZED2 is a USB-powered video camera with low level access to the device. It provides control over all the camera parameters such as exposure, gain, sharpness, etc. Thanks to its comprehensive and well-documented API, it can be interfaced with multiple third-party libraries and environments. In our case, we use the ZED ROS wrapper to use the camera with ROS Noetic environment.

However, despite the ZED2 camera has such powerful functions, we decided to use it as a monocular camera, because its depth detection ability using dual vision is limited by its size, and is not enough for a vehicle in actual road scenarios. Not to mention the complexity of performing the joint calibration between the stereoscopic view of the camera and the LiDAR.

3.2.3 PandarXT-32

The PandarXT-32 is a high-performance LiDAR sensor developed by Hesai Technology, a leading Chinese manufacturer of advanced 3D sensing solutions. It is a 32-channel LiDAR sensor designed for autonomous driving, robotics, and other high-precision sensing applications. It adopts the mechanical rotation scanning method to generate a 360-degree horizontal FOV. The channels of the LiDAR are uniformly distributed in the vertical direction, with an interval of 1 degree between adjacent wire harnesses, as shown in Figure 3.3:

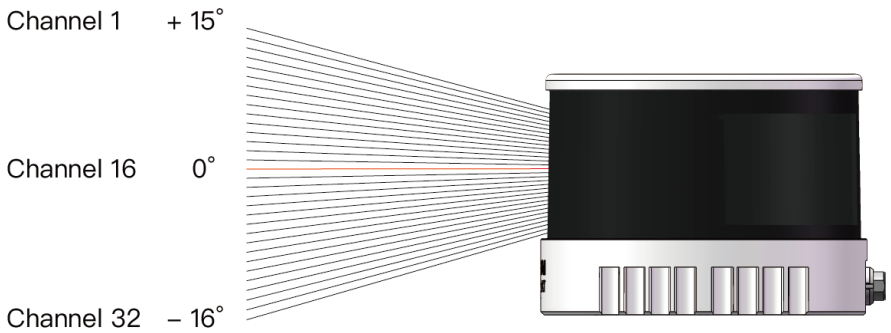

Figure 3.3: LiDAR's Channel Vertical Distribution

The vertical FOV is 31 degrees (-16 degrees to +15 degrees). The instrument

range is from 0.05m to 120m, and the range capability is 80m at 10% reflectivity for the channels in the middle of the vertical FOV (Channels 9 24), and 50m at 10% reflectivity for the channels on both sides of the vertical FOV (Channels 1 8, 25 32). The vertical resolution is 1 degree, and the horizontal resolution is 0.18 degree at 10 Hz frame rate typically. The range accuracy is ±1cm and the precision is 0.5cm.

There are two return modes to choose, single return and dual return. The single return means that for each laser beam, the LiDAR receives either the strongest or the last return signal, while dual return means that for each laser beam, both return signals are received by the LiDAR. In this project, we choose the single return mode to receive the strongest return signal, so the data rate is 640,000 points per second at 10 Hz frame rate. The wavelength of the laser beam is 905nm, and it satisfies the Class 1 Eye Safe standard. The LiDAR is capable of operating in temperatures ranging from –20 to +65 degrees Celsius.

The point cloud data is transmitted using UDP/IP Ethernet protocol. The data is composed of distance, azimuth angle, and intensity of each returned point. The LiDAR uses GPS or PTP as the clock source to define the timestamp of each frame of point cloud data.

3.3 Algorithms Development

3.3.1 ROS environment

ROS (Robot Operating System) is an open-source software development kit for robotics applications. Although ROS is not a real operating system, but a set of software frameworks for robot software development, it has the services that an operating system can provide, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. ROS offers a standard software platform to developers from various industries, providing full support from research and prototype design to deployment and production. It also provides a flexible and distributed architecture that enables communication, coordination, and collaboration among various components and modules of a robotic system.

Figure 3.4: The ROS Ecosystem

The core of ROS is a messaging system, often called middleware or plumbing. Middleware is a type of software that bridges the gap between applications and operating systems. It provides a method of data management and communication, and enables the interaction between software and hardware. The ROS runtime is a peer-to-peer network of loosely coupled processes using the ROS communication infrastructure. ROS implements several different communication methods, including synchronous RPC-style request–response communication over services, asynchronous streaming of data over topics based on a publish-subscribe messaging model, and storage of data on a Parameter Server. Although reactivity and low latency are important in robot control, ROS is not a real-time framework. However, it is possible to integrate ROS with real-time computing code.

ROS follows a modular approach, where a robotic system is divided into individual software modules called nodes. A node is an executable that uses ROS to communicate with other nodes. Nodes can perform specific tasks, such as controlling actuators, processing sensor data, or implementing algorithms. Nodes can communicate with each other by publish messages or subscribe to predefined topics, and can also provide or use a Service. ROS organizes software components into packages, which are the basic unit of software distribution in ROS. Packages contain nodes, libraries, configuration files, and other resources related to a specific functionality or application. For example, the package must contain a catkin compliant package.xml file, which is called manifest and is a description of a package. It serves to define dependencies between packages and to capture meta information about the package like version, maintainer, and license.

The catkin is the official build system of ROS, which combines CMake macros and Python scripts to provide more functionality on top of CMake's normal workflow. The build system is responsible for generating targets that end users can use from raw source code. The targets can be libraries, executables, generated scripts, exported interfaces such as C++ header files, or any other non-static code. A package typically consists of one or more targets during the building process. ROS is designed to support cross platform code reuse in robotics research and development, so the language-independent and platform-independent tools are included for building and distributing ROS-based software, and Python and C++ are fully supported. All the mentioned features make ROS highly scalable and suitable for large-scale runtime systems and development processes.

3.3.2 Yolo v7

The YOLO algorithm, as the most typical representative of one stage object detection algorithm, is based on deep neural networks for object detection, with fast running speed and can be used in real-time systems. YOLOv7 was released by Chien-Yao Wang and Alexey Bochkovskiy in July 2022. At that time, it was the

most advanced algorithm in the YOLO series, surpassing previous YOLO series and other object detectors in both accuracy and speed in the range from 5 FPS to 160 FPS. It also had the highest accuracy 56.8% AP among all known real-time object detectors with 30 FPS or higher on GPU V100.

Compared with the most advanced real-time object detectors at that time, YOLOv7 reduced the parameter count by about 40% and the computational complexity by about 50%. Its optimization mainly involves two aspects: model architecture optimization and training process optimization. They proposed the extend and compound scaling methods that effectively utilize parameters and computational complexity for model architecture optimization. For the optimization of the training process, the concept of bag-of-freebies was proposed in YOLOv4, which is a module or method that improves accuracy at the cost of increasing training costs but does not increase inference costs. In YOLOv7, the re-parameterized technology is used to replace the original module, and the dynamic label assignment strategy is used to allocate labels more efficiently to different output layers.

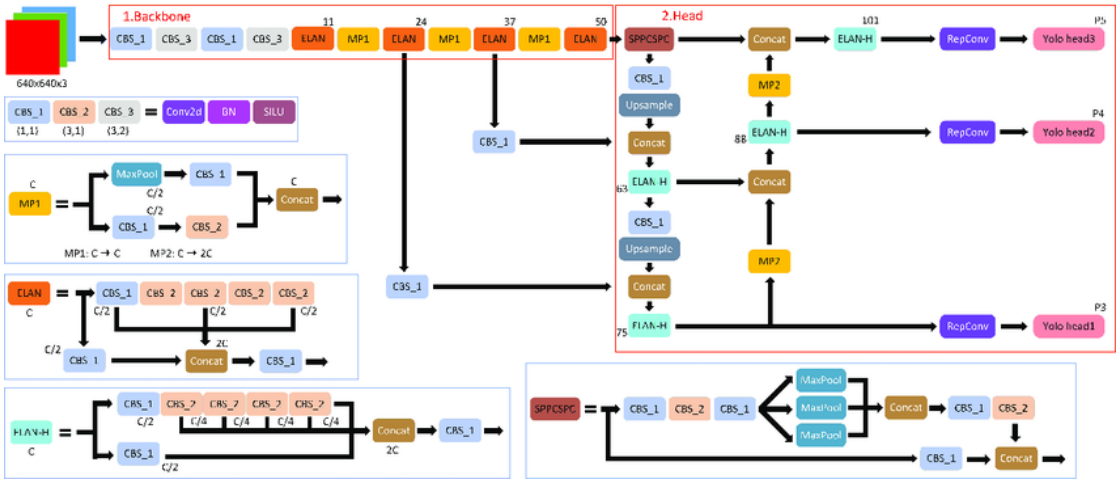

Figure 3.5: The structure of YOLOv7

The network structure of YOLOv7 is divided into three parts: input, backbone network, and head network, as shown in Figure 3.5 [47]. The backbone uses convolutional neural networks to effectively extract image features. It consists of several CBS (Conv+BN+SiLU), ELAN and MP (Maxpooling) layers. The head is responsible for detecting targets and generating bounding boxes and class predictions. It consists of a SPPCSPC layer, several Conv and MP layers, and then three Rep+Conv layers for output three unprocessed prediction results of different sizes. The three output layers are used to detect targets of different scales. Each output layer will generate a set of bounding boxes and category predictions, and then use the Non-Maximum Suppression (NMS) algorithm to eliminate duplicate

bounding boxes to obtain the final detection result.

The workflow of Yolov7 can be briefly summarized as follows: first, the input image is resized to 640x640, and features are extracted by the backbone network. Then, feature maps in three different sizes are output through the head network. The Rep and Conv layers predict the three tasks of image detection, namely the classification, foreground and background separation, and bounding box. Finally, the prediction results are output. It is worth mentioning that a threshold can be set on the confidence score, so that the Yolov7 only output the detections with a confidence score above the threshold. In our use case, we set the threshold to 50%.

The detailed working principle of the Yolov7 neural network is not the focus of this study, for in depth analysis on the Yolov7, please refer to[48].

3.3.3 Training of Yolo v7

To fulfill the task of object detection in this project, it is crucial to define the target objects that we want to identify. According to the requirement of the entire project, there are 25 classes of objects that need to be detected, including vehicles, traffic light status, speed limits, traffic signs that may affect speed, pedestrians, and bikes. The detailed classes are listed in the Table 7.1.

In order to train the neural network to detect all the required targets, we need to prepare our own dataset. Many annotated datasets for autonomous driving research can be found on the internet, but their annotation does not meet our needs. So, we decided to use the images from these datasets, but manually annotated them according to the task requirements.

The datasets we used to train the Yolov7 are BDD100K from Berkeley DeepDrive Industry Consortium[49], The German Traffic Sign Recognition Benchmark (GTSRB)[50], The German Traffic Sign Detection Benchmark (GTSDB)[51], and Road Sign Detection[52]. The BDD100K dataset consists of 100,000 videos. Each video is about 40 seconds long, 720p, and 30 fps. The videos were collected from diverse locations in the United States, covering different weather conditions, including sunny, overcast, and rainy, as well as different times of day including daytime and nighttime. We used the 100K Images subset from the BDD100k, which consists of the frames at the 10th second in the videos. The GTSRB was introduced as a multi-class, single-image classification challenge which was held at the IEEE International Joint Conference on Neural Networks (IJCNN) 2011. As a successor to the GTSRB, the GTSDB was introduced on the IEEE IJCNN 2013. They both provide a single-image detection assessment dataset for researchers with interest in the field of computer vision, pattern recognition and image-based driver assistance. The Road Sign Detection dataset was published on Kaggle.com and it contains 877 images of 4 distinct classes for the objective of road sign detection.

After careful examination of the datasets, we select in total 3,108 images to train

the Yolov7, which contains 1,451 images from BDD100k, 99 images from GTSRB, 706 images from GTSDB, and 852 images from the Road Sign Detection.

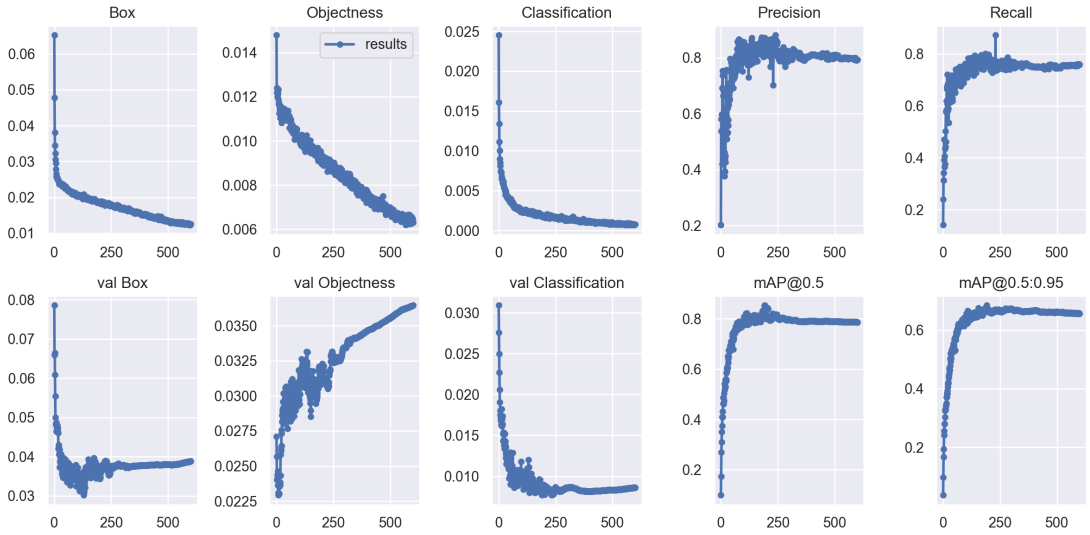

Figure 3.6: Yolov7 Training Results

To manually annotate the images, we use the Yolo_mark tool provided by Alexey Bochkovskiy[53], which is a GUI for marking bounding boxes of objects in images for training Yolo neural network. It outputs label files directly in Yolo format. The label file is a txt file with the same name as the corresponding image, and the content is the object number and object coordinates on this image. For each object, it takes up a line of <object-class> <x_center> <y_center> <width> <height>.

The training of the Yolov7 was conducted locally on the workstation in our laboratory, with an Intel i7 16 core CPU and a NVIDIA GeForce RTX 3080Ti GPU. As for the main parameters, we used all 16 workers (CPU cores), 640x640 image input size, 4 batch size, and trained for 600 epochs. The whole training process lasted for 13.085 hours. The training results are shown in Figure 3.6 and 3.7.

3.3.4 LiDAR and Camera Joint Calibration

In order to get the LiDAR and the camera to work together, the first step is to perform the joint calibration between the two sensors, so that they can work in a common coordinate system. The joint calibration typically involves determining the relative position and orientation between the two sensors, which is described by the extrinsic parameters. The basic problem in calibration is to calculate the

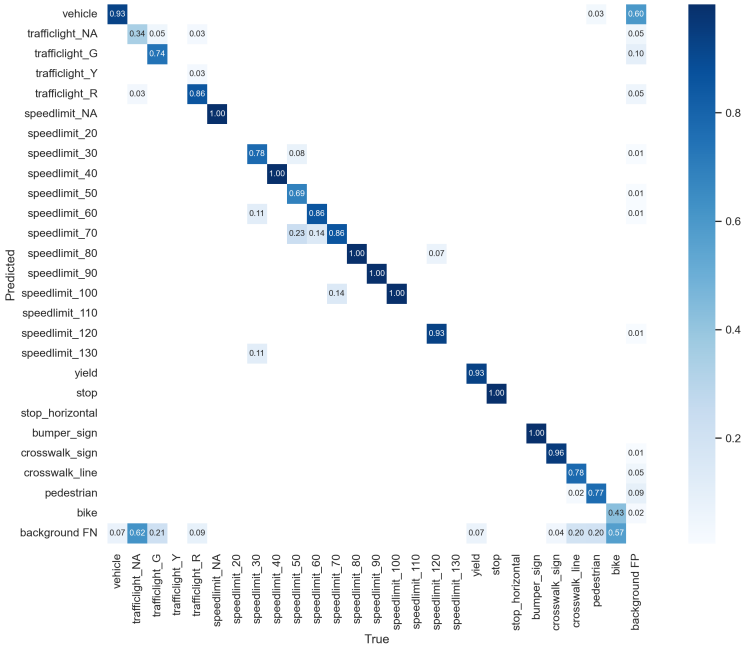

Figure 3.7: Yolov7 Confusion Matrix

transformation between the two frames of the camera and the LiDAR, which has been theoretically solved, but still challenging and often underestimated in applications. The main challenge is to achieve robust and accurate estimation of calibration in practice, which is often affected by human operations, environmental factors, and sensor errors.

In our case, what we want to achieve is to project 3D point of the LiDAR to the 2D image plane of the camera. The extrinsic parameters are the translation and rotation vectors that convert LiDAR's coordinate system to the camera's coordinate system. The image frame is the left camera color rectified image from the ZED2. The distortion-free projective transformation given by a pinhole camera model is described by the function:

$$p_C = A \left[R|t \right] P_L \tag{3.1}$$

where

- P_L is a 3D point expressed in the LiDAR coordinate system
- p_C is a 2D pixel in the image plane in the camera coordinate system
- A is the intrinsic parameter matrix of the camera
- R and t are the rotation and translation vector mentioned above

The camera intrinsic matrix A projects 3D points given in the camera coordinate system P_C to 2D pixel coordinates:

$$p_C = AP_C \tag{3.2}$$

The A is composed of the focal lengths f_x and f_y, which are expressed in pixel units, and the principal point (c_x, c_y) is usually at the image center:

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, the projection equation can be expressed as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.3}$$

The intrinsic parameters do not depend on the scene viewed. They are estimated before leaving the factory, and provided in the calibration file.

The joint rotation-translation matrix $[R|t]$ is the matrix product of a projective transformation and a homogeneous transformation. The 3×4 projective transformation maps 3D points in camera coordinates to 2D points in the image plane and represented in normalized camera coordinates $x' = X_c/Z_c$ and $y' = Y_c/Z_c$:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

The homogeneous transformation is encoded by the extrinsic parameters R and t and represents the change of basis from the LiDAR's coordinate system L to the camera's coordinate system C. R is a 3×3 rotation matrix, and t is a 3×1 translation vector:

$$\begin{bmatrix} R|t \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

Therefore, if we combine the above mentioned parameters together, we can write out 3.1 as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_L \\ Y_L \\ Z_L \\ 1 \end{bmatrix} \tag{3.4}$$

The objective of calibration is to acquire the R and t vectors[25].

Extrinsic calibration parameters are very sensitive to noise during the feature extraction process, as small errors in rotation or translation estimation can greatly affect the usability of the calibration result. For example, the uncertainty in plane-fitting or line-fitting of edges caused by measurement errors of sensors. Tsai et al.[54] proposed a well-designed method to perform the calibration using multiple sets of 3 poses of chessboard, to obtain a robust estimate of the calibration parameters with uncertainty. In our project, we adopted their method to calibrate our LiDAR and camera system. The calibration pipeline is shown in the Figure 3.8.The main idea of their method will be briefly explained in the following paragraphs.

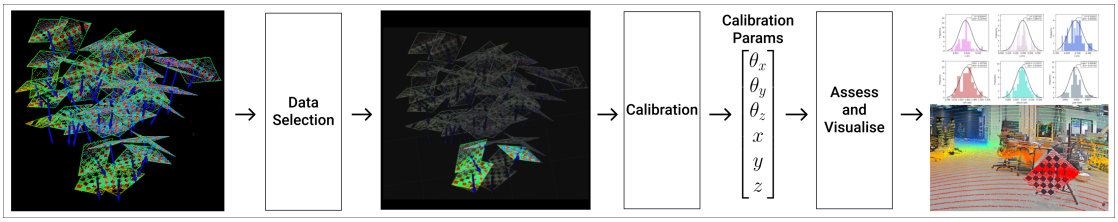

Figure 3.8: Calibration Pipeline

The rotation matrix R_C^L represents a rotation transformation between two coordinate system in a three-dimensional space. It is used to rotate points around a specified axis or a combination of axes. In their calibration method, the algorithm extracts the center point and plane normal of a chessboard target from the LiDAR point cloud and the image, which are denoted as N_C and N_L. Then, from the equation:

$$R_C^L N_C = N_L \tag{3.5}$$

where

- $N_C = \begin{bmatrix} n_C^0 & n_C^1 & n_C^2 \end{bmatrix}^T$
- $N_L = \begin{bmatrix} n_L^0 & n_L^1 & n_L^2 \end{bmatrix}^T$

the rotation matrix can be calculated. The equation is modified by Verma et al.[55] to include more plane normals in N_C and N_L to improve the robustness of the algorithm. However, in practice, if too many poses are included, the algorithm tends to be over-fitted. In this method, instead of using more poses for a single calibration, it uses multiple sets of 3 poses as illustrated in Figure 3.9 to obtain a robust estimation of the rotation matrix with some uncertainty.

Similarly, the translation matrix represents a translation transformation between two coordinate system in a three-dimensional space. It is used to shift points by a specified distance in different directions, which is more straight forward to calculate.

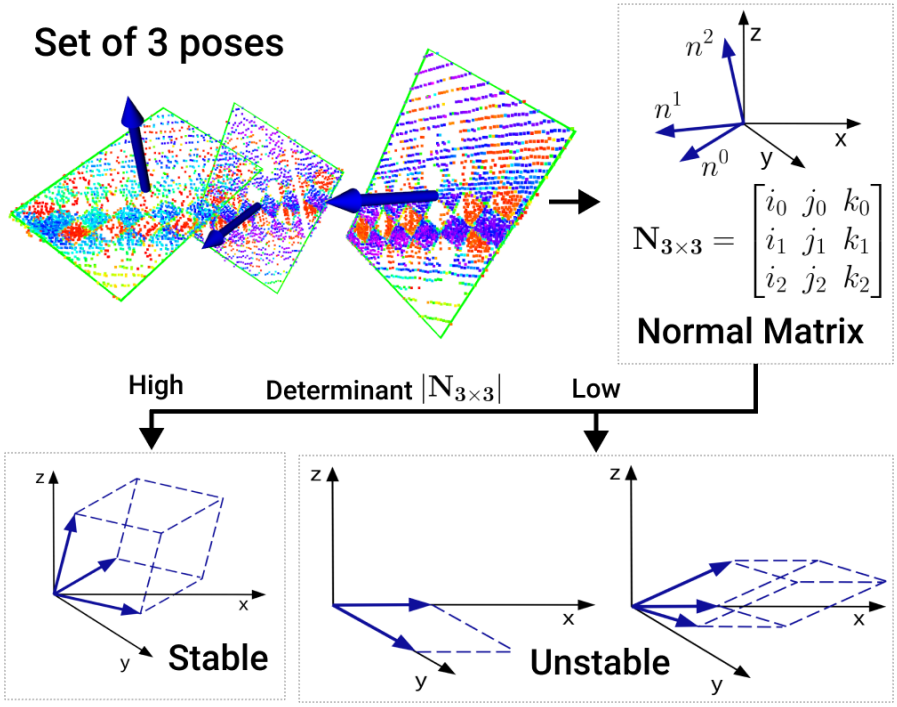

Figure 3.9: Set of 3 Poses

The translation matrix is computed from aligning the center of the chessboard in the LiDAR and the camera domain. So, the accuracy of the center estimation by the LiDAR and the camera greatly impacts the reliability of the calibration. Since the height and width of the chessboard is known, it is easy to find the center in the camera view by counting the pixels in the image. The tricky part lies in the LiDAR domain.

To perform a line-fitting or plane-fitting in the LiDAR cloud points, the Random Sample Consensus (RANSAC)[56][57] algorithm is a popular choice. RANSAC is an iterative algorithm to estimate parameters of a mathematical model from a set of data points that contains outliers. It works by estimating the parameters of the desired model that best fit the inlier data points while ignoring the outliers. The pseudocode of RANSAC is described in Algorithm 1. It is a non-deterministic algorithm, because it only produces a reasonable result with a certain probability, and this probability will increase as more iterations are allowed. It accomplishes this by iteratively selecting a random subset of the data points, known as a minimal sample, and fitting a model to this subset. The model is then tested against the remaining data points to determine how well it fits. Data points that are consistent with the model within a specified threshold are considered inliers, while those that deviate beyond the threshold are considered outliers. By repeating this process

for enough iterations, RANSAC aims to find the best model that maximizes the number of inliers and minimizes the impact of outliers.

For example, if the task is to estimate the parameters of a plane in a point cloud, RANSAC will sample three points that are the minimum needed to define a plane (provided they are not all co-linear to each other), then calculate the Euclidean distance from all points in the point cloud to such a plane, and then calculate the percentage of inliers based on a certain threshold. After a certain number of iterations, RANSAC will return the plane with the highest support, thereby returning the plane with more points closer to it.

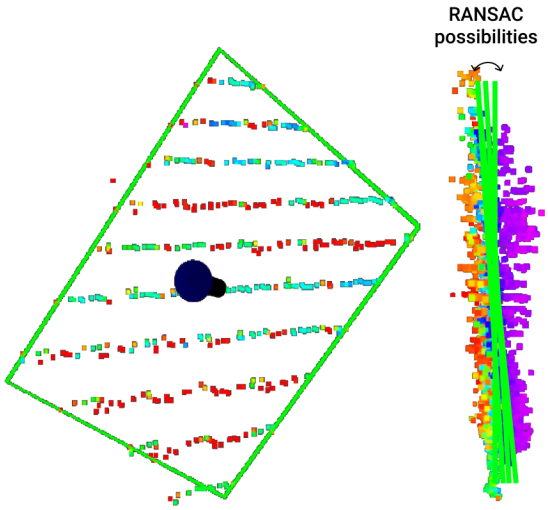

Figure 3.10: LiDAR Range Error

In our case, the method first determines the edge lines of the chessboard using the RANSAC algorithm, then it finds the intersection of edge lines to identify the corners. This estimation is heavily influenced by the number of points that lie on the board, and the range accuracy of those points due to that LiDAR has errors in the range measurement as illustrated in Figure 3.10. It has been found empirically that the error in chessboard measurement is greater at certain distances from the sensor pair, such as when the distance is too close or too far, and when the chessboard is facing certain directions. By calculating the measurement error of the chessboard, we can quantitatively evaluate the accuracy of the chessboard center estimation.

For a set of three poses, the quality of the rotation parameters can be evaluated using the rank of the matrix, and the quality of the translation parameters can be evaluated using errors in chessboard measurements. These metrics can be combined into a single equation called Variability of Quality (VOQ).

$$VOQ = k_{LC} + e_{be} \tag{3.6}$$

where

$$k(N) = \| N \| \| N^{-1} \|$$

$$k_{LC} = max\{k(N_L), k(N_C)\}$$

$$e_{dim} = \sum_{i=0}^{3} |l_{L,i} - l_{M,i}|$$

$$e_{be} = \frac{1}{3} \sum_{j=0}^{2} e_{dim,j}$$

- k_{LC} is the condition number of the matrix, which represents the linear dependency of the matrix, and gives an indication of the stability of its inverse. If the normal matrix tends to be linear dependent, the computation of the inverse becomes unstable and can negatively affect calibration results. As shown in Figure 3.9, the determinant $|N_{3\times3}|$ reflects the stability of the inverse computation, and a high determinant means a stable inverse. However, it's hard to be used as a metric, because it's difficult to set a boundary. So they decided to use the condition number as a metric, as $k = 1$ means the most stable result, and $k \to +\infty$ means the least stable result.

- the sum of dimension errors e_{dim} represents the accuracy of sensor measurements, where $l_{L,i}$, i = 0, 1, 2, 3 are the LiDAR's measurement of the chessboard's edge lengths, and $l_{M,i}$ are the actual edge lengths. For each set of 3 poses, we have 3 errors $e_{dim,j}$, j = 0, 1, 2, and the mean error is e_{be} in millimetres.

A low VOQ score means better generalization of the calibration results, while a high VOQ score leads to overfitting and a higher standard deviation of the calibration results.

The calibration procedure consists of preparing the chessboard, fixing the relative position between radar and camera, capturing at least 20 poses, and run the algorithm to estimate the parameters. Our chessboard has 7x5 inner vertices with 90mm square length, and the board dimension is 565x793mm. The setup of our sensor pair is shown in the picture. We captured 33 poses. The output parameters are listed in Table 3.1. The histogram with a Gaussian fit is shown in the Figure 3.11.

The output calibration parameters describe the transformation from the camera frame to the LiDAR frame, which means that the camera coordinate system is the parent coordinate system, and that is the rule defined in the OpenCV library. However, in the fusion algorithm we will introduce in detail in the following paragraphs, the rotation vector required by OpenCV is in Rodrigues rotation representation, that is, the Axis-Angle representation, while the rotation vector

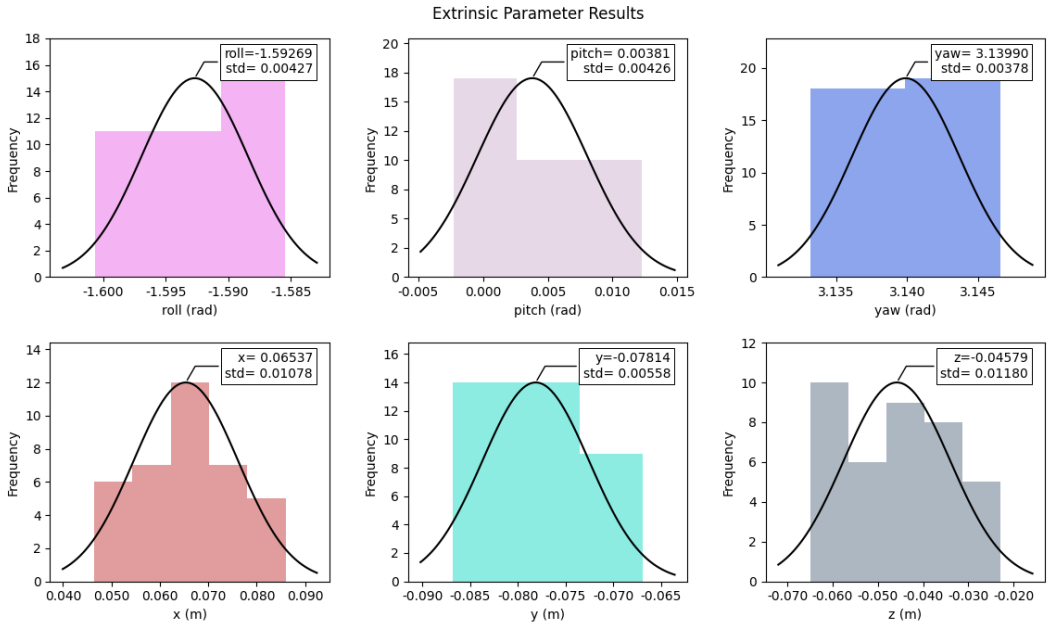

Figure 3.11: Extrinsic parameter results

roll	-1.59269
pitch	0.00381
yaw	3.13990
x	0.06537
y	-0.07814
z	-0.04579

Table 3.1: Calibration output parameters

given by the calibration algorithm is in Euler Angle representation, so an additional conversion is needed. We can use the equation introduced in [58] to compute the conversion. The final rotation vector is [0.0061, 2.2445, -2.1959], and the translation vector is [0.0654, -0.0781, -0.0458].

3.3.5 Sensor Fusion Algorithm

At this stage, we have a calibrated camera LiDAR sensor set, a properly trained object detector, and a ROS environment running on our NVIDIA Jetson AGX Xavier. Now, our goal is to fuse these data from the sensors to extract the distance information for the detected objects. We have proposed two versions of the sensor fusion algorithms:

- The first version utilizes the point cloud data after preprocessing, which includes down-sampling and clustering. This approach isolates the objects of interest and filters out noisy background, thereby enhancing the accuracy of distance measurement.

- The second version involves using point cloud data with a simpler preprocessing step. It includes filtering out points at ground height and applying restrictions on the FOV to focus solely on objects in front of us. This version does not employ down-sampling and clustering.

The main objective of this design is to conduct a comparative study between the two approaches. We aim to evaluate differences in terms of processing time, precision in distance measurement, and the impact of different parameter settings in the preprocessing step on overall sensor fusion performance. We chose to use raw data because LiDAR distance measurements are highly precise. In simple and controlled scenarios, we can consider these measurements as a baseline or even as ground truth to some extent, because obtaining precise ground truth distance values during tests on real vehicles in real-world environments is impractical.

The sensor fusion process consists of six steps: time synchronization of the data-flows, preprocessing of the LiDAR cloud points, distance calculation of the cloud points, projection of the 3D cloud points to 2D Yolov7 detection images, the distance measurement of the detected object, and finally the output of the detection information.

Time synchronization of the data-flows

Our fusion algorithm needs to subscribe to two topics, the point cloud data from the LiDAR and the image data with bounding box information of the detections from the Yolov7. The two dataflows are at different rates. The LiDAR outputs point cloud data at 10 Hz (10 samples per second), while the Yolov7 outputs image at about 3 to 4 Hz. It means the algorithm receives about 2 or 3 frames of point cloud data between two image frames. The purpose of sensor fusion is to combine information from the two sensors at the same time to extract the useful information in real-time. However, if the rates of the two data streams are different, it will greatly impact the real-time performance, especially in real-world traffic scenarios. In order to solve this problem, we need to design a strategy to synchronize the two data streams. The main idea of the synchronization is, firstly, to drop some frames of the faster sensor which are in between the intervals of the frames of the slower sensor to reduce the complexity of the data processing, and secondly, to make sure the data we kept is collected at approximately the same time and represents the same state of the world.

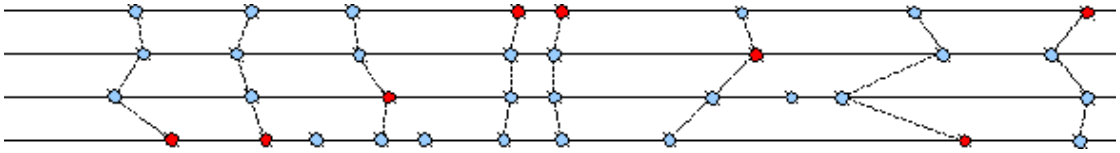

Figure 3.12: ApproximateTime policy

ROS provides a time synchronizer in the "message_filters" library, which takes in messages of different types from multiple sources, and outputs them only if it has received a message on each of those sources with the same timestamp. Both the data frames from the LiDAR and the Yolov7 have a timestamp in their headers. We adopt the Policy-Based Synchronizer filter to synchronize the dataflows received from the two topics. There are two policies: the ExactTime and the ApproximateTime. The ExactTime policy requires messages to have exactly the same timestamp in order to match, while the ApproximateTime policy uses an adaptive algorithm to match messages based on their timestamps. In our case, since it's not realistic to expect that there could be enough frames from the two data flows to arrive at exactly the same time, we choose the ApproximateTime policy, since it finds the best match by allowing some time difference.

The adaptive algorithm works as following: firstly, for a new set of frames from the two topics, it finds the latest message among the heads of the two queues, and we call that the pivot. For each message older than the pivot in the other queue, we create a set of frames as candidate. Then, it goes through the queue until the end in order of time in search of a better candidate. Finally, it outputs the optimal candidate as the new frame set to the callback function. The process can be shown intuitively in the Figure 3.12 [59][60].

In our algorithm, we first create two Subscriber filter to subscribe to the LiDAR and Yolov7, respectively. Then, we create a synchronizer with the ApproximateTime policy, and register the callback function for the synchronizer.

Preprocessing of the LiDAR cloud points

After the time synchronization strategy is defined, we need to design the callback function. The input data of the callback function are the point cloud data of the $sensor\_msgs :: PointCloud2$ message type, and the image data with detection information of the $vision\_msgs :: Detection2DArray$ message type. The message structures of the two types are shown in List 8.1 and 8.2.

In our solution, we use the PCL library to process the point cloud data, and the OpenCV to process the image data. The PCL stands for the Point Cloud Library, which is a standalone, large scale, open project for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the art

algorithms including noise filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. The OpenCV stands for the Open-Source Computer Vision Library, which is an open-source computer vision and machine learning software library. It has more than 2500 optimized algorithms, which has a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms, including object identification, face detection and recognition, tracking camera movement and moving objects, producing 3D point clouds from stereo camera, etc.

The first task of our callback function is to convert the message of $sensor\_msgs :: PointCloud2$ data type to the $PointCloud < pcl :: PointXYZRGB >$ data type in PCL. In this way, we can get a point cloud data with the coordinate information of x, y, z, and create a RGB field for each point. which can be used for visualization in the next steps.

The second task is to apply a filter on the point cloud data by x-axis to limit the FOV of the LiDAR. The purpose of this step is focus on detecting the distance of the objects in front of our vehicle and on the adjacent lanes on both sides.

The second task is to filter out the points on the ground, because in the task of environment perception, we are only interested in the objects on the ground, not the ground itself. The points on the ground account for a considerable portion of the cloud points, and they can affect the accuracy of the subsequent clustering task. There are many methods commonly used to determine if a point belongs to the ground, such as using RANSAC to fit a plane model to the points and identifying the ground points based on their proximity to the estimated ground plane, or using the morphological approach to classify ground and non-ground points by comparing the height difference between neighboring points, or even the machine learning-based methods classify points as ground or non-ground based on features derived from the point cloud. However, those methods are very complex and resource intensive.

As the main purpose of this step is to preliminary filter out some ground points without affecting the computational efficiency, in order to be more accurate in the clustering step, we chose a relatively simple method, which is to set a threshold on the z-axis to filter out points on the ground according to the height of the LiDAR from the ground. We use the $pcl :: PassThrough$ function to implement this operation.

Then, we need to down-sample the point cloud data, which reduces the number of points while preserving important features and structures. For each frame of point cloud data, there are 64,000 points, even if we only focus on the 180 degrees of FOV in the front after the ground filtering, there are still about 20,000 points. Down-sampling can significantly reduce the number of points by about 50%, and the clustering can further reduce 50% to 80% of points depending on the road scene, surrounding environment, methods and parameters, therefore reducing the

computational complexity, memory requirements, and processing time in future operations.

The down-sampling process involves selectively retaining a subset of points from the original point cloud based on certain criteria. Here we adopt voxel grid down-sampling method, which divides the 3D space of the input point cloud data into a grid of uniform-sized cubic voxels. Then, the points falling within the same voxel are grouped together, and only one representative point is retained for that voxel. The representative point can be the centroid of the points inside the voxel or the center of the voxel.

We apply the $pcl :: VoxelGrid$ function to realize the voxel grid down-sampling, and it uses the centroid as the representative point, because though it is a bit slower than approximating the points with the center of the voxel, but it represents the underlying surface more accurately. We set the voxel grid leaf size to 10cm, because in our planned application scenario, the common detection distance is between 5 to 20 meters. The error introduced by a 10cm leaf size is acceptable, and its performance is the best when we tried different leaf sizes ranging from 5cm to 30cm. This process effectively reduces the number of points by preserving the overall distribution and shape of the original point cloud.

The next task is to perform a Euclidean cluster extraction from the down-sampled point cloud. It is often used in segmentation tasks, where groups of points that are spatially close to each other are grouped into separate clusters based on their Euclidean distance. It assumes that points belonging to the same object or surface are in proximity to each other. By clustering points based on their spatial proximity, we can segment objects or structures within a point cloud. However, the segmentation of the point cloud is not the goal of our project. The purpose of the clustering is to separate the points belonging to the detected objects from the noisy points in the background, because the depth information is lost when the 3D point cloud is projected onto the 2D image plane.

If from the camera's point of view, there is obstruction or overlap between objects, or there is a background far behind the detected objects, and at the same time, if we experience some fluctuations in message transmission due to the hardware problems such that the two data streams are misaligned, then the projection of 3D points within the bounding box may be wrong, resulting in incorrect distance measurement. Due to the difficulty of directly addressing hardware issues, we choose to avoid or at least mitigate the impact of this phenomenon from a software perspective. By down-sampling, the point cloud of objects farther away from us becomes sparser, while objects closer to us are less affected. Then we cluster the down-sampled point cloud, and by setting appropriate parameters, we can filter out most of the background noise, such as houses, trees, etc. In this way, even if there is a deviation in the projection, at least the distance of the detected object will not be wrongly measured as the distance of the background noise.

In our algorithm, we perform the extraction of the Euclidean clusters with the *pcl* :: *EuclideanClusterExtraction* class. The class provides various parameters that can be adjusted to control the clustering behavior, including the minimum and maximum cluster sizes, the distance threshold for considering points as neighbors, and the search method to use. The first step is to define a search method for the extraction. PCL provides many search methods, such as KdTree, Octree, and FlannSearch:

- The Octree is a tree-based data structure used for voxel-based spatial partitioning, which divides 3D space into smaller cubic voxels. The root of the Octree represents the entire 3D space, and each level of the tree further subdivides the space into eight sub-voxels.
- The Flann stands for Fast Library for Approximate Nearest Neighbors, which focuses on performing efficient approximate nearest neighbor searches rather than exact searches.
- The KdTree, or k-dimensional tree, is a data structure used for organizing data points in a space with k dimensions. It is particularly useful for fast and efficient nearest neighbor searches and range queries in high-dimensional spaces. It is a binary tree where each node represents a k-dimensional point and recursively splits the data space into two halves based on a splitting criterion.

In terms of our application scenario, KdTree is the most suitable one. The point clouds are in three dimensions, so our k-d tree is a three-dimensional tree. At each level of the tree, all the data are divided along a specific dimension, using a hyperplane that is perpendicular to the corresponding axis. For example, at the root of the tree, the tree splits based on the x axis, at the next level, the tree splits based on the next axis, and so on. At each level of the k-d tree, the data points are partitioned into two subsets based on the splitting criterion. Points on one side of the split are assigned to the left sub-tree, and points on the other side are assigned to the right sub-tree. Then it repeats this procedure on both the left and right sub-trees until each leaf node represents a single data point.

The second step is to define the other three main parameters. The first one is the Cluster Tolerance, which is the distance threshold used to determine if points in a point cloud belong to the same cluster. It specifies the maximum distance between two points for them to be considered part of the same cluster. This parameter has a significant impact on the quality of clustering, as a larger tolerance value allows for greater distances between points, resulting in larger clusters, while a smaller tolerance value restricts the distance between points, leading to smaller clusters.

In our case, we set the tolerance to 30cm, which means if the distance between two points is larger than 30cm, they will not be regarded as the same object. The

reason is that we set the voxel size to 10cm during the down-sampling stage. It is reasonable to assume that for the two adjacent voxels in the diagonal direction after down-sampling, the maximum distance between the farthest two points is less than 35cm, so 30cm is a reasonable choice.

The other two parameters are Minimum Cluster Size and Maximum Cluster Size, which represent the minimum and maximum number of points required for a cluster to be considered valid and included in the output. Clusters with points less than the specified minimum or greater than the maximum will be discarded. We set the minimum to 50 and maximum to 20000, because the main target for the down-sampling and clustering is to filter out the noisy background, and this set of parameters worked best in the test.

Once we set all the parameters, we can perform extract the clusters. The algorithmic steps are listed in the pseudo code in Algorithm 2. At this point, the preprocessing of the LiDAR cloud points is completed.

Distance calculation of the cloud points

There are different ways to define the distance between vehicles and objects, each with its own advantages and considerations. One approach is to utilize point cloud data to calculate the three-dimensional Euclidean distance. By measuring the Euclidean distance in this three-dimensional space, we can obtain an accurate measure of the physical distance between the LiDAR and an object.

To calculate the Euclidean distance between the points and the LiDAR is straightforward. It can be done using the distance function with the x, y, z values of the coordination information.

$$d = \sqrt{(x_P - x_O)^2 + (y_P - y_O)^2 + (z_P - z_O)^2}$$

On the other hand, a simpler alternative is to directly use the distance information along the y-axis to represent the object's distance in front of the vehicle, which is also called longitudinal distance. This approach assumes that the vehicle's forward direction aligns with the y-axis of the LiDAR. By considering only the longitudinal distance, we simplify the calculation and reduce the dimensionality of the problem. This method can be particularly useful in certain scenarios where the precise three-dimensional location of objects might not be necessary, such as when estimating distances within a constrained region directly in front of the vehicle.

Since our LiDAR is mounted on top of the vehicle, calculating the Euclidean distance using point cloud data inherently includes the height of our vehicle in the measurement. However, when defining the distance between two vehicles, we typically focus on the distance from our vehicle's front bumper to the rear bumper or the nearest part of the other vehicle. In this context, considering the longitudinal distance becomes a more appropriate and practical representation. By utilizing the

longitudinal distance, we can effectively capture the relative distance between the relevant parts of the vehicles, providing a more meaningful measure in our specific scenario.

Here we also want to emphasize on the order of the data process. After the down-sampling and clustering, we still obtain a point cloud data. However, the projection function requires a vector<Point3f> data type as input, so we need to convert the point cloud data to a vector<Point3f> before the projection. Since we need to traverse the point cloud and reorganize it into an ordered vector, we can extract or calculate the distance information during this process, and create a distance vector in the same order as the point cloud vector, so that we can search for the distance information in future operations based on the order of the elements in the vector.

3D to 2D Projection

The projection of the 3D cloud points onto 2D Yolov7 detection images is the core of our algorithm. This operation correctly assigns the LiDAR points to the corresponding objects in the image, so that we can extract the distance information of the detected objects. In this step, we use the $cv :: projectPoints$ function to implement the projection. This function computes the 2D projections of 3D points to the image plane given intrinsic and extrinsic camera parameters. As mentioned in the previous step, the input is an array of object points, the array can be a 3xN/Nx3 1-channel array or a 1xN/Nx1 3-channel array, where N is the number of points, or it can be a vector of vector<Point3f> datatype, which is what we use in our solution.

Then the function needs the rotation vector in Rodrigues rotation representation and the translation vector as the extrinsic parameters, which define the relative position and orientation between the camera's coordinate system and the LiDAR's coordinate system. The function also needs the camera intrinsic matrix and the distortion coefficients as the intrinsic parameters. The intrinsic matrix describes the focal length and the optical center of the camera, and the distortion coefficients describes the distortion of the lens. Since the ZED2 camera is well calibrated in the factory, and it offers a rectified image output, so the distortion coefficients can be set to all zeros, and the intrinsic matrix is provided by the manufacture of the camera. The extrinsic parameters are calculated from the calibration step. The parameters we were using are listed in the chapter 7 Parameters.

After we input the vector of points and the parameters, the function outputs an array of image points, the array can be a 1xN/Nx1 2-channel array, or a vector of or vector<Point2f> datatype. The output image points are pixels in the Yolov7 detection image. Each pixel represents a corresponding LiDAR point.

The distance measurement of the detected objects

After the projection step, we iterate through the detection results to calculate the distance of each detected object. As mentioned before, each object contains information such as its object class, the coordinates of the center point of the bounding box, as well as its height and width. To determine if the 2D points from the projected LiDAR data fall within the bounding box, we traverse the vector of the projected LiDAR points. As the bounding box is a rectangular box surrounding the object, and the shapes of the detected objects are diverse, there may be some blank spaces at the corners of the bounding box, particularly for the most common targets like vehicles. To avoid mistakenly selecting background points as part of the detected objects, we need a method to eliminate the corners of the bounding boxes.

Here we propose a method that applies a re-scale factor to both the height and width of the bounding box, effectively shrinking it. This allows us to focus more on the points around the center of the object. In our algorithm, we set the re-scale factor to 90%.

For the 2D points within the bounding box, we utilize their position information to find the corresponding distance information. Since we already have a vector of distances in the same order as the LiDAR points, we create a new vector for each detected object to store its distance information.

Once we have extracted the distance of each LiDAR point within the bounding box of the detected object, there are two approaches to represent the relative distance. We can either use the minimum value or calculate the average value of the distance vector. However, even after shrinking the bounding box, in extreme cases, there may still be a few outliers in the distance vector. To address this issue, we employ another method. We calculate the average value using a truncated mean method, which involves disregarding obvious outliers based on a certain percentage threshold. For example, if a vehicle is approximately 5 meters away in front of us, most of the points reflected by the vehicle should have distances around 5 meters. However, if there are some points reflected by the background, such as the road, walls, or other vehicles in front of the target vehicle, the distances might be around 10 meters, indicating they are obviously the outliers. In our algorithm, we set the truncate threshold to 10%.

The minimum distance measurement provides an indication of the closest point between the two cars, while the average distance offers a more generalized representation based on the overall distances observed. By comparing the minimum distance and average distance, we can evaluate which calculation method is more suitable for representing the relative distance between two cars from the perspective of the driver. As the sensor set is positioned on top of the vehicle, its FOV differs from that of the driver. Consequently, the distance measured by the LiDAR may

not align precisely with the driver's perception. Therefore, it becomes crucial to find a representation that better corresponds to the driver's perception of distance. This evaluation helps ensure that the system provides distance measurements that are intuitive and more driver friendly, enhancing the overall usability and safety of the system in real-world driving situations.

The output of the detection information

In this step, we can choose to output any necessary information according to the different requirements of the project at different development stages. For the testing stage, we need to publish point cloud after the down-sampling and clustering to tune the parameters, and visualize the projected LiDAR points on the image to check if the projection is correct. We also need to print the detection information, such as object class, confidence score, and distance information, and save those information to a local CSV file for post-processing to evaluate the performance of the algorithm. When the algorithm is integrated into the ADAS system, we will need to output the required information according to the downstream requirements of the pipeline.

Figure 3.13: LiDAR and camera sensor set

3.4 Hardware Deployment

In order to assess the viability of installing the sensor set on an actual vehicle and evaluate the real-time performance of the entire system on the road, we have selected a FIAT QUBO as our test vehicle. The configuration of the LiDAR and camera sensor set can be seen in Figure 3.13, with the sensors mounted above the front windshield of the vehicle as depicted in Figure 3.14. To utilize the outlet of the air conditioning system as a cooling mechanism, we have positioned the NVIDIA Jetson AGX Xavier AI Vehicle Computer and the battery for powering the system on the floor of the co-pilot seat.

Figure 3.14: LiDAR and camera mounting position

3.4.1 Testing Plan

Since our system is designed for the implementation on a real vehicle, it is important to perform the tests in a real-world environment. There are several indicators that we want to focus on in the testing:

- The first one is to assess the precision of the distance measurement. This involves evaluating the two different approaches for calculating relative distance (Euclidean distance and longitudinal distance) and two different representations for evaluating the distance (minimum value and truncated mean value). To achieve this, we gather data in a simple and controlled environment. Our test platform consists of our vehicle equipped with the sensor set, while another vehicle serves as the target object. These tests are conducted in an empty

parking lot, allowing us to create a scenario where we can precisely control the variables and obtain accurate measurements.

- The second one is to assess the stability of object detection and distance measurement. This test involves simulating a typical driving scenario where we follow another vehicle on a highway or suburban road while maintaining a relatively fixed distance to the target vehicle in front of us. By conducting this test, we can evaluate the stability and potential of our sensor set for future project developments, such as object tracking and Adaptive Cruise Control (ACC). The aim is to ensure that our system can reliably detect and measure distances to objects under real-world driving conditions, demonstrating its effectiveness and suitability for practical applications.

- The third one is to assess the robustness of multiple object detection and distance measurement. This test involves driving in a busy and crowded urban neighborhood characterized by a rapidly changing and complex environment. In addition to vehicles parked on both sides of the road and approaching vehicles from the opposite direction, the surroundings may include various traffic signs, bicycles, and pedestrians. Since this test is conducted in a real-world neighborhood, obtaining ground truth data for the relative distances to each detected object is not feasible. Therefore, the key performance indicators in this scenario are the accuracy of the Yolov7 detection results and the reasonableness of the distance measurements. The aim is to evaluate how well our system performs in detecting multiple objects in such dynamic and challenging environments. We assess the accuracy of object detection using the Yolov7 model and examine the reliability and rationality of the distance measurements. This test will provide valuable insights into the system's ability to handle complex real-world scenarios and will aid in further refining and improving our project's capabilities.

3.5 Overview of the Integration Pipeline

This system incorporates multiple hardware and software components with scalable potential. It handles the real-time processing of data flows from various sensors, extracting valuable information, and transmitting it to the upper-level control unit. The system architecture is illustrated as Figure 3.15.

The hardware components include sensors such as LiDAR and cameras, which capture the surrounding environment. These sensors generate continuous data streams that need to be processed in real-time to extract relevant information.

The software components consist of algorithms and processing modules that handle the data from the sensors. These components are responsible for performing

tasks such as object detection, sensor fusion, distance measurement, and data interpretation. The algorithms and modules are designed to efficiently process the incoming sensor data and extract useful information from it.

The system is designed with scalability in mind, meaning that it can handle an increasing number of sensors and adapt to different configurations. This scalability allows for the integration of additional sensors or the expansion of the system's capabilities as needed.

One of the key challenges is the real-time processing requirement, as the system needs to handle and analyze data streams in a timely manner to ensure the driving safety. This necessitates efficient algorithms and hardware resources capable of handling the computational load.

Overall, the system's primary objective is to process data from multiple sensors in real-time, extract meaningful information, and transmit it to the upper-level control unit for decision-making and further actions. Its scalable design allows for future enhancements and adaptability to different applications and scenarios.

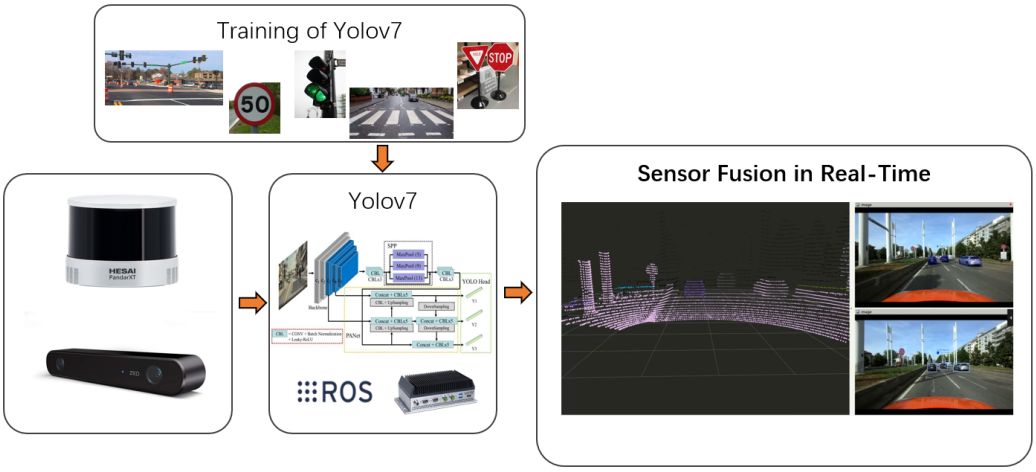

Figure 3.15: Integration Pipeline

Chapter 4

Experiments and Results

This chapter is dedicated to the implementation of the test plan and the subsequent evaluation of the test results. The implementation of the test plan involves carrying out the predefined test scenarios and procedures. This includes setting up the necessary hardware and software components, configuring the system accordingly, and executing the planned tests in a controlled and consistent manner. Following the completion of the test execution, the evaluation of the test results is performed. This evaluation phase involves analyzing the collected data, assessing the system's performance, and comparing the obtained results against the predefined success criteria or benchmarks. The test results are thoroughly examined to determine the system's strengths, weaknesses, and areas for further improvement.

4.1 Data collection

As per the test plan, we conducted the tests in three distinct scenarios and obtained raw data from the LiDAR and camera sensors. To facilitate further analysis and processing, the acquired data were stored in rosbag files. Although the fusion algorithm is capable of real-time execution, the offline fusion process primarily serves the purpose of testing various parameter combinations and conducting data analysis. By performing offline fusion, we have the flexibility to experiment with different parameter settings and evaluate their impact on the fusion results. This allows us to fine-tune the algorithm and optimize its performance. Additionally, conducting offline fusion enables us to thoroughly analyze the data, examine the accuracy of the fusion process, and gain insights into the overall performance of the system.

4.1.1 Precision of the Distance Measurement

In the first testing scenario, we focused on assessing the distance measurement capabilities of the system. This test was conducted in an empty parking lot that featured standard parking spaces, each measuring 2.5 meters in width. For the first test, as shown in Figure 4.1, the setup involved a target vehicle parked in a fixed position, while our test vehicle, equipped with the sensor set, parked behind the target vehicle at an initial distance of 2.5 meters. It should be noted that in the context of this chapter, the distance refers to the distance from the LiDAR to the rear of the target vehicle. To evaluate the distance measurement accuracy, our test vehicle slowly moved backward, increasing the distance between the test vehicle and the target vehicle by 2.5 meters. Once the movement was completed, we stopped and repeated the process until the Yolov7 object detection system stopped detecting the target vehicle. By systematically increasing the distance and observing the point at which the target vehicle was no longer detected, we were able to assess the system's ability to accurately measure distances.

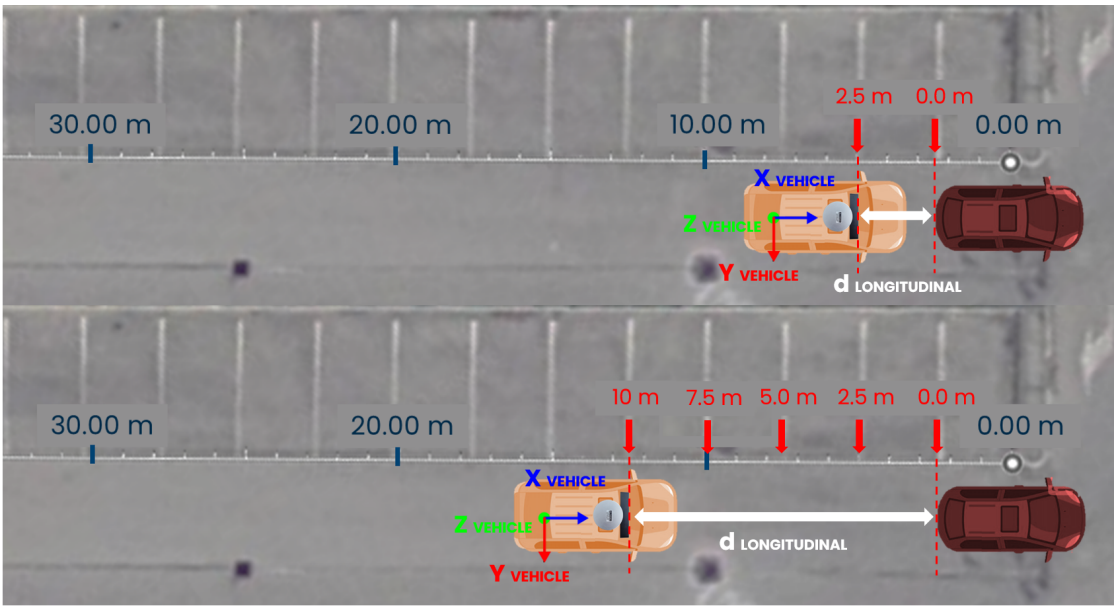

Figure 4.1: Precision evaluation in the parking lot (Vehicle)

For the second test, as depicted in Figure 4.2, we conducted an experiment where we gradually approached a yield sign. The aim was to determine the maximum distance at which the system could recognize the yield sign and to observe the differences between the Euclidean distance representation and the longitudinal distance representation.

In the offline fusion step, we implemented the two versions of the sensor fusion

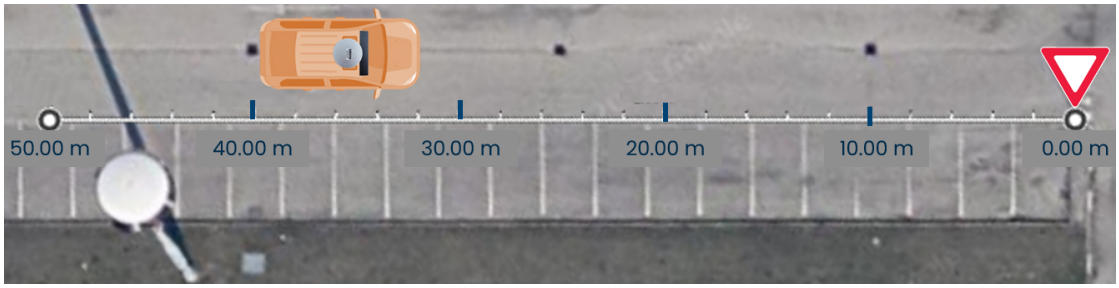

Figure 4.2: Precision evaluation in the parking lot (Yield Sign)

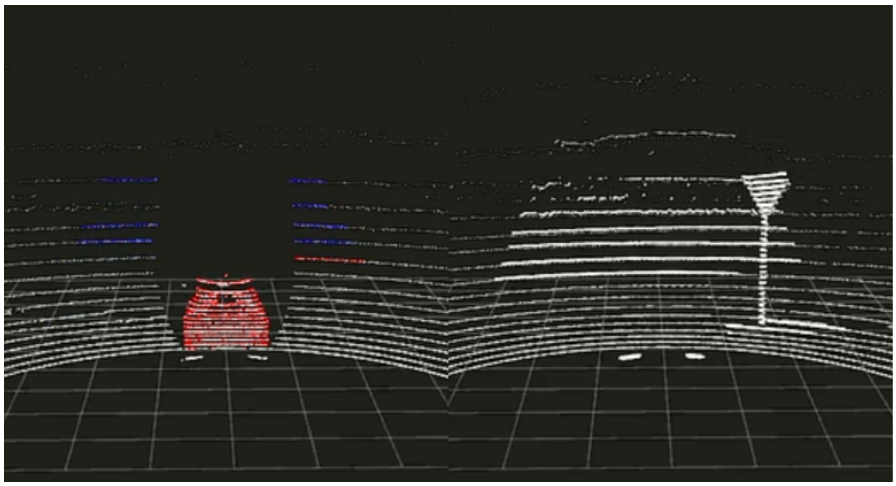

Figure 4.3: Point cloud visualization (left: vehicle, right: yield sign)

algorithm introduced earlier in Section 3.3.5, the point cloud visualization of the two tests is shown as 4.3. In this controlled environment, our primary objective was to investigate the feasibility of using the raw data as the baseline for distance measurement. To achieve this, we compared the measurements obtained from the raw data with manually measured distances. Furthermore, we compared the distance measurements derived from the raw data with those obtained from the preprocessed point cloud. This comparison allowed us to evaluate the differences between the two approaches, particularly at longer distances. By conducting these comparisons, we aimed to assess the accuracy and reliability of the sensor fusion algorithms in capturing distance information. We specifically focused on understanding how well the raw data approach aligned with the manually measured distances, as well as examining any disparities between the raw data and preprocessed point cloud approaches. By comparing and evaluating the results from both approaches, we gained a better understanding of their respective strengths, limitations, and potential applications in real-world scenarios.

Additionally, we utilized this opportunity to evaluate the differences between the two distance calculation methods, considering the combination of the test driver's perception. By comparing the results obtained from the minimum distance calculation and the average distance calculation, we could assess which method better aligned with the test driver's perception of distance. This evaluation allowed us to understand the system's performance in determining the precise distance to the target vehicle in various positions and distances.

4.1.2 Stability of Object Detection and Distance Measurement

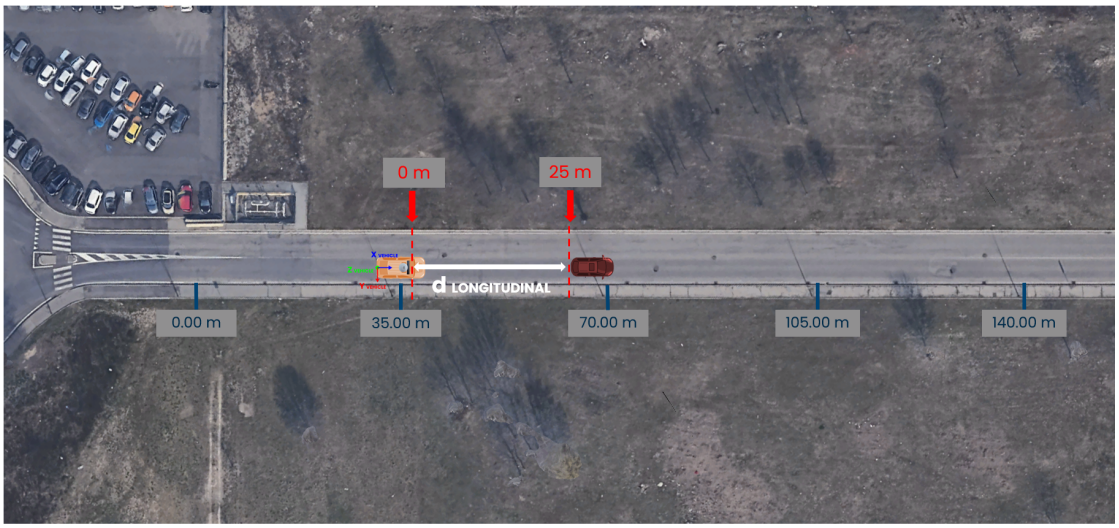

Figure 4.4: Car following scenario

In the second testing scenario, our focus was on evaluating the stability of object detection and distance measurement. This test took place on an empty suburban road to simulate real-world driving conditions. The test began with two cars parked on the road, facing the same direction, with the target vehicle positioned in front and the test vehicle positioned behind it. The target vehicle gradually started to move away, fading into the distance. Once the Yolov7 object detection system stopped detecting the target vehicle, the test vehicle began to chase after it. As the two cars eventually met again, they both resumed normal driving while maintaining a relatively fixed distance between them, as shown in Figure 4.5 and Figure 4.4. This scenario allowed us to assess the system's stability in detecting objects, as well as its capability to maintain a steady distance from the target vehicle.

In the offline fusion step, similar to the first testing scenario, we implemented the two versions of the sensor fusion algorithm to assess the impact of preprocessing the

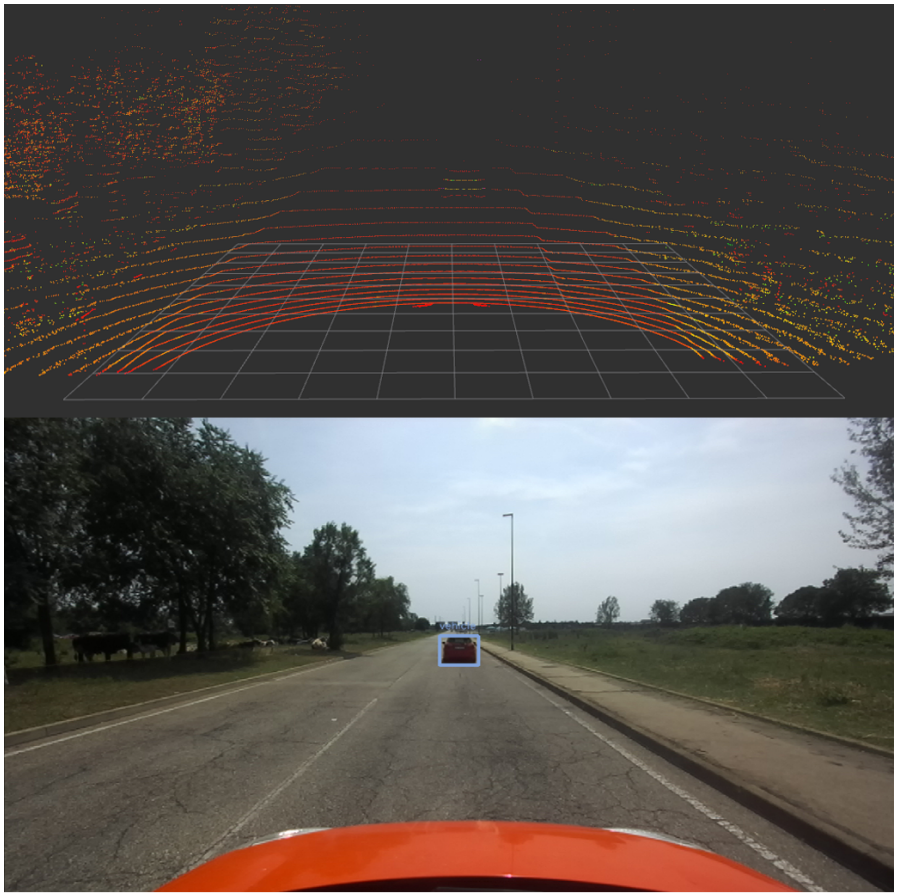

Figure 4.5: Car following visualization

point cloud data during dynamic driving conditions. This evaluation allowed us to understand how different preprocessing techniques affect the system's performance in detecting and measuring distances to objects in real-time. By conducting these evaluations in the offline fusion step, we aimed to gain insights into the effectiveness and practicality of the preprocessing techniques in enhancing the system's performance. Furthermore, by aligning the distance calculation methods with the test driver's perception, we could determine the most suitable representation of relative distance for the specific driving scenario.

The results obtained from this test provided insights into the system's reliability and accuracy in real-world driving scenarios, demonstrating its potential for applications such as object tracking and Adaptive Cruise Control (ACC).

Figure 4.6: Typical Neighborhood in Torino

4.1.3 Robustness of Multiple Object Detection and Distance Measurement.

In the third testing scenario, our focus was on assessing the robustness of multiple object detection and distance measurement. This test was conducted in three different locations, each representing various real-world driving scenarios:

- Typical Neighborhood in Torino: As depicted in 4.6, this location simulated a typical neighborhood environment with parked cars on both sides of a narrow road. Additionally, there were occasional pedestrians, bicycles, and traffic signs indicating speed limits or pedestrian lines. This scenario aimed to evaluate the system's ability to detect and measure distances accurately in a complex and dynamic neighborhood setting.

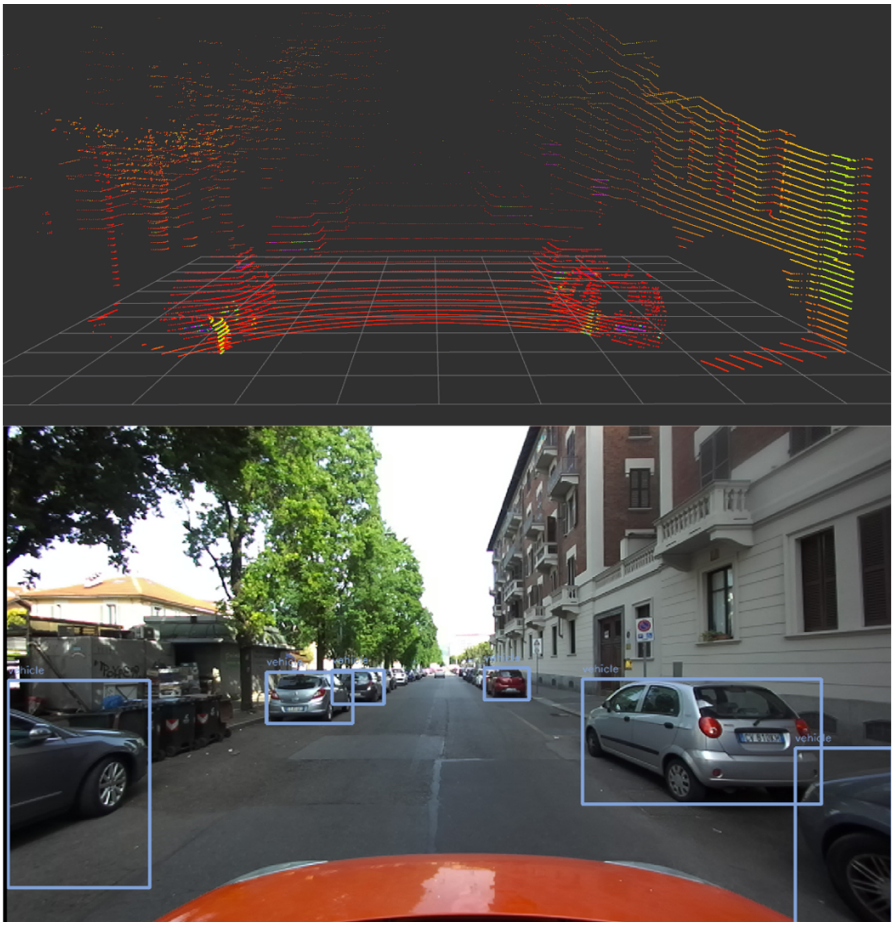

Figure 4.7: Two-Lane Road without Traffic Light

- Two-Lane Road without Traffic Light: In this scenario, we conducted tests on a two-lane road with low traffic flow. This scenario aimed to simulate a typical urban driving environment, as shown in 4.7. While there were no traffic lights present, we encountered occasional appearances of pedestrians and loading trucks, adding complexity to the driving situation. The purpose of this scenario was to evaluate the system's performance in a realistic urban driving scenario where traffic lights were absent. The evaluation helped identify strengths, weaknesses, and potential areas for improvement, ultimately enhancing the system's performance and ensuring its suitability for use in typical urban driving scenarios.

- Three-Lane Road with Traffic Light: In this scenario, we started from approaching a traffic light and waiting for it to turn green. This test involved driving on a three-lane road alongside other vehicles, with instances of lane

switching performed by both the test vehicle and other vehicles, as shown in 4.8. This particular situation represents a common use case for ADAS. This scenario aimed to evaluate the system's robustness in detecting and measuring distances accurately and ability to handle traffic light detection in a dynamic traffic situation with lane changes and intersections.

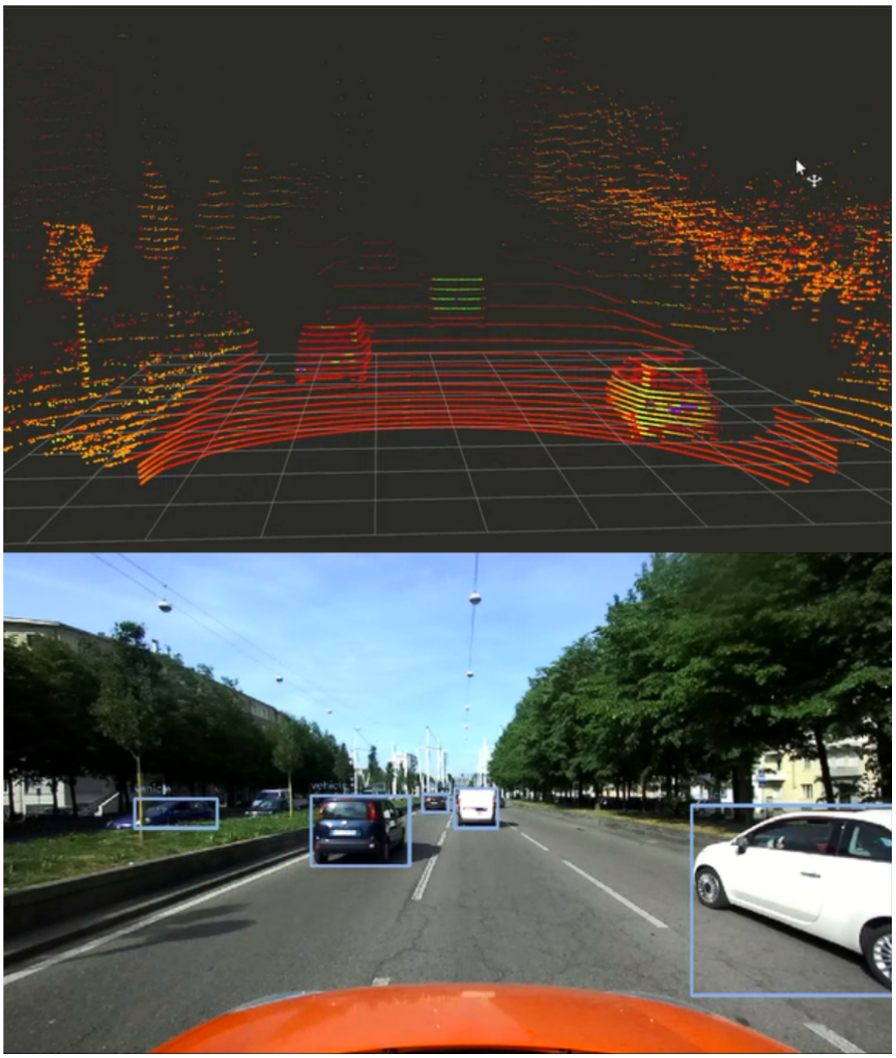

Figure 4.8: Three-Lane Road with Traffic Light

By conducting the tests in these three different locations, we aimed to assess the system's performance in detecting multiple objects and accurately measuring their distances in real-world driving conditions. The tests helped us evaluate the system's reliability, robustness, and its ability to provide accurate and timely information to assist drivers in various driving scenarios.

In the offline fusion step, in contrast to the first and second testing scenarios, we solely implemented the preprocessed version of the sensor fusion algorithm. This decision was made based on our understanding from the algorithm's design phase that using raw point cloud data in crowded scenarios, where objects overlap with each other, could lead to incorrect distance measurements. Therefore, we introduced clustering of the raw point cloud data to differentiate objects from each other and the noisy background, ensuring more accurate distance measurement.

Nevertheless, we still conducted a comparison between the two distance calculation methods to determine which representation is more reasonable in an urban driving scenario. This evaluation aimed to identify the most appropriate approach for representing relative distances in the context of complex urban environments. By focusing on the preprocessed version of the sensor fusion algorithm and comparing the distance calculation methods, we aimed to assess the system's performance in accurately measuring distances between objects in crowded urban driving scenarios. Through this analysis, we sought to provide a reference basis for future improvements and upgrades to enhance the system's reliability, and ensure that it provides distance measurements that are reasonable and applicable to real-world urban driving scenarios.

Additionally, since we have applied a filter on the x-axis of the point cloud data to limit the LiDAR's FOV and focus only on the objects in front of us and the adjacent lanes, we can select a scenario to evaluate the impact of this filter on the fusion time compared to the algorithm without any limitation on FOV.

4.2 Accuracy Evaluation

In this section, we present and discuss the results of the offline fusion from the three testing scenarios. For each scenario, we have developed specific criteria to evaluate the system's capabilities and performance in different real-world driving situations.

4.2.1 Precision of the Distance Measurement

In the first scenario, our focus was on comparing the measurements obtained from the sensor fusion algorithm with manually measured distances. The purpose of this evaluation was to verify the precision of the distance measurement performed by the system. Since LiDAR is renowned for its high precision in measuring distances, it is crucial to determine the most suitable distance representation in order to fully leverage the potential of the LiDAR sensor.

Firstly, we need to assess which distance calculation method is more reasonable to be used as the default method in the subsequent tests. Figure 4.9 demonstrates

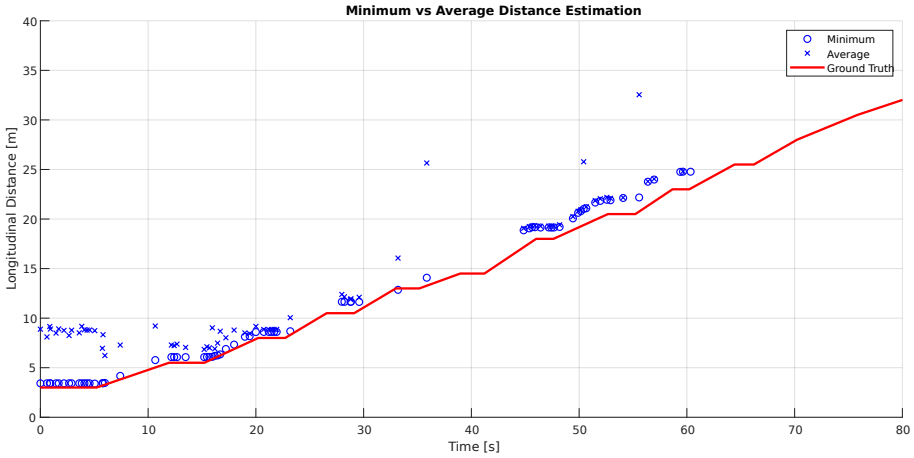

Figure 4.9: Minimum vs Average Distance Estimation

that the distance calculated using the minimum value provides a more stable measurement compared to the distance calculated using the average value, and Figure 4.10 shows a significantly higher root mean square error (RMSE) for the average value, where $RMSE_{avg} = 5.1545$ and $RMSE_{min} = 2.4878$. This observation is significant as it indicates that relying on the minimum value as a representative distance yields more consistent and reliable results. Using the minimum value helps mitigate the influence of outliers or irregularities in the measured distances, leading to a more robust distance measurement.

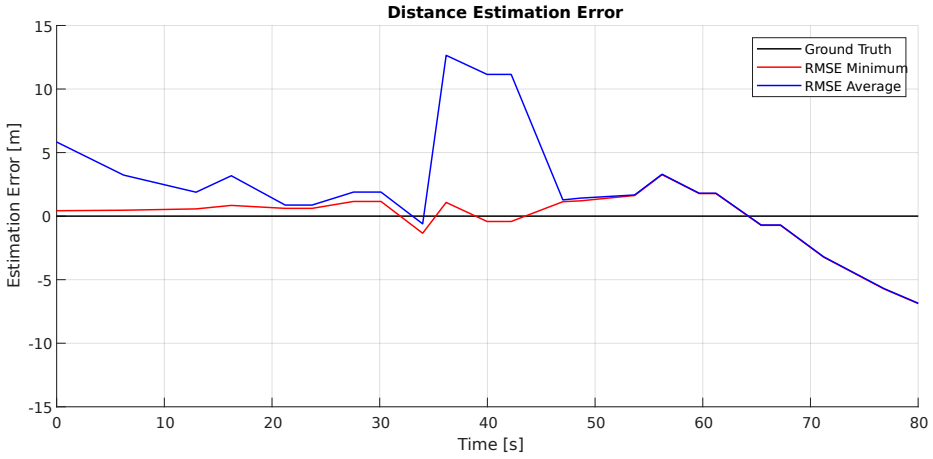

Figure 4.10: Minimum vs Average Distance Estimation Error

As mentioned in 3.3.5, we employed the Euclidean distance ($d_{euclidean}$) and the

longitudinal distance ($d_{longitudinal}$) as distance representations, aiming to determine which one is more appropriate through this test.

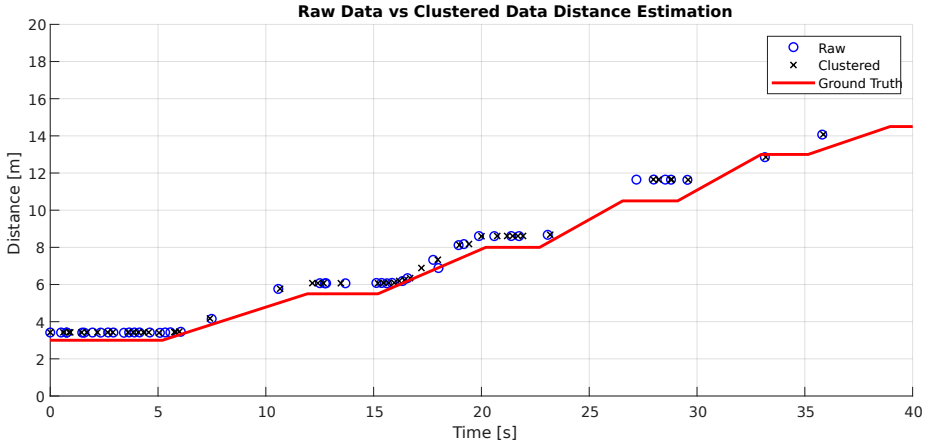

Figure 4.11: Raw Data vs Clusterd Data Distance Estimation

In Figure 4.11, we can observe that the measured distances closely align with the ground truth, indicating that the LiDAR's ranging capability is reliable and precise when measuring a single target directly in front of the vehicle, and there is not much disparity in the two distance representations.

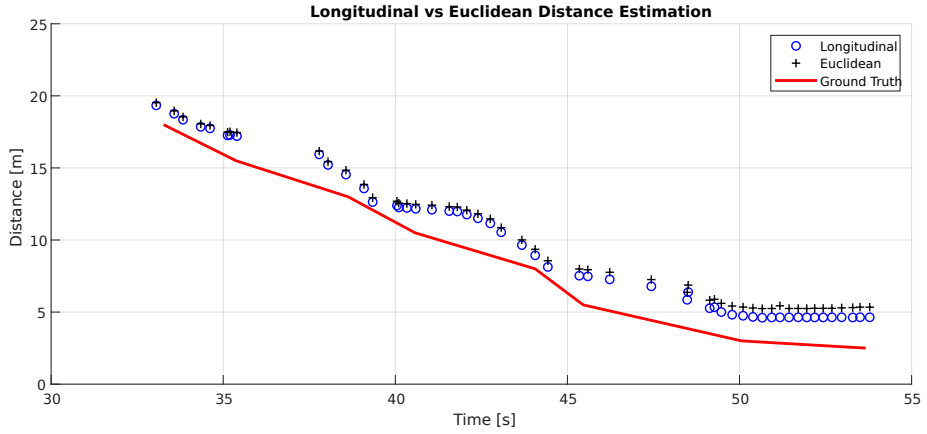

Figure 4.12: Longitudinal vs Euclidean Distance Estimation

However, as shown in Figure 4.12, in the test where we gradually approach a yield sign, the difference between the distance representations becomes more pronounced. This disparity arises because the Euclidean calculation takes into account the height (z-axis) and lateral distance (x-axis) of the detected object, as depicted in Figure 4.13, resulting in a larger Euclidean distance compared to the

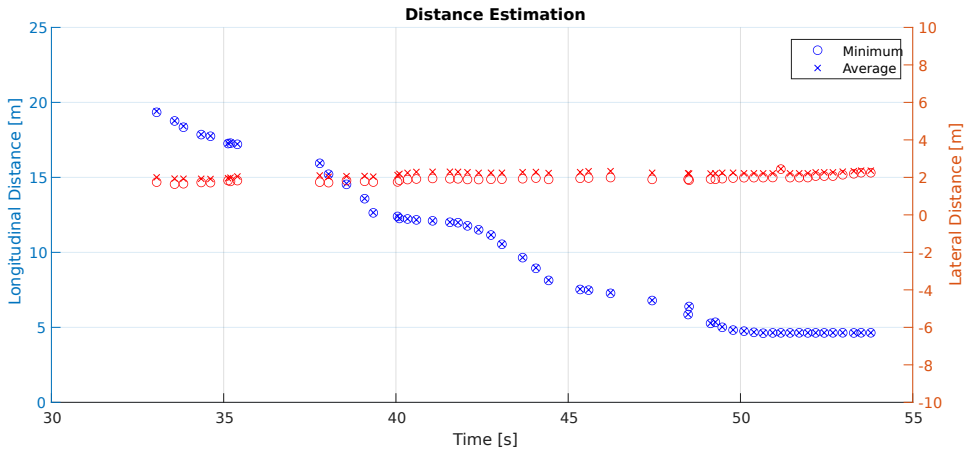

Figure 4.13: Longitudinal vs Lateral Distance Estimation

longitudinal distance. Based on this test, we can confidently conclude that, in driving scenarios, the longitudinal distance representation is more aligned with the driver's perception of distance.

Additionally, we observe a difference in the maximum detection range between the two preprocessing approaches. The measurement from the raw data exhibits a longer detection range compared to that from the preprocessed data. This disparity arises due to the minimum number of reflected points threshold applied during the preprocessing of point cloud data. As the distance between the two vehicles increases, the point cloud becomes sparser, resulting in fewer points being reflected back by the target vehicle. Once the number of points falls below the threshold, the target vehicle is filtered out, causing the algorithm to fail in recognizing it. However, this situation represents a trade-off between ranging ability and measurement stability, which will be further elucidated in the subsequent test.

4.2.2 Stability of Object Detection and Distance Measurement

In the second scenario, the main objective is to assess the stability of object detection and distance measurement using both raw data and preprocessed data. To achieve this, we concentrate on determining the maximum effective detection distance and evaluating the stability of distance measurements within this range while the target vehicle is maintaining a relatively fixed distance from the test vehicle.

Since we have already established that the longitudinal distance representation is more reasonable, we plot the variation of the longitudinal distance to assess

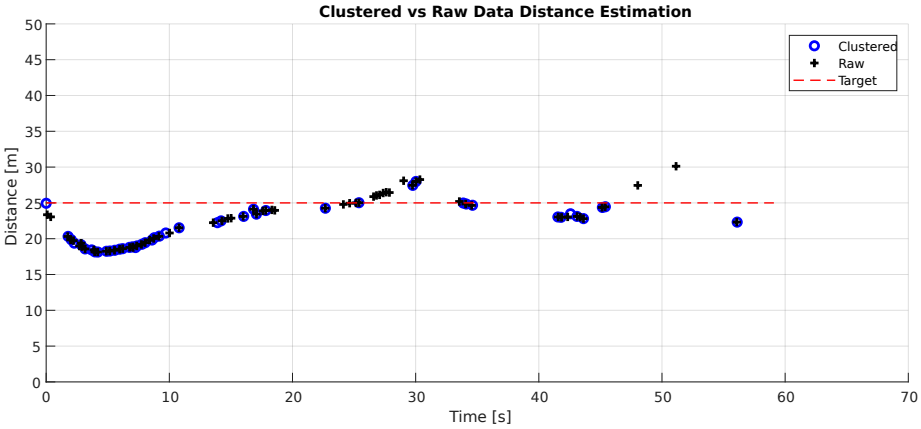

Figure 4.14: Clustered vs Raw Data Stability Estimation

its stability when maintaining a relatively constant distance from the test vehicle. From the Figure 4.14, we can see that the maximum effective and stable detection distance of clustered data is 25 meters, while the detection range using the raw data can reach up to 30 meters, which aligns with the driver's perception. In urban driving scenarios, drivers typically focus more on the environment within a range of approximately 30 meters. Therefore, for the current sensor set and system, the detection range meets the requirements of the ADAS system's use case.

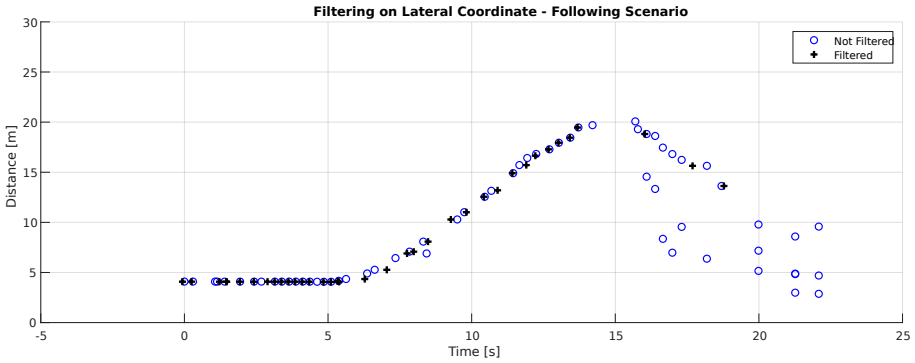

Figure 4.15: Filtering on Lateral Coordinate

Regarding the ranging stability, the Figure 4.14 illustrates that the distances measured from both the preprocessed data and the raw data exhibit stability. This observation suggests that both the preprocessed data, obtained after applying downsampling and clustering, and the raw data provide consistent and reliable distance measurements. The stability in the measured distances indicates that the fusion algorithm effectively processes the data and maintains accuracy throughout

the car following scenario. Additionally, we implemented a filter on the lateral coordinate to restrict the FOV and focus specifically on the current lane. Figure 4.15 presents a comparison between the filtered and unfiltered situations. Figure 4.16 illustrates a typical urban driving scenario, where we follow a car in front of us. The distance measurements captured during this scenario provide valuable insights into the system's performance. The minimum distance recorded in the figure is 5 meters, indicating that the system successfully maintains a safe distance from the car in front of us when coming to a stop. On the other hand, the maximum distance observed is 30 meters, representing the system's maximum ranging capability. The stability of the measured distances is a crucial aspect in ADAS systems, as it directly impacts the system's ability to make informed decisions and provide timely warnings to drivers, demonstrating its potential for applications such as object tracking and ACC.

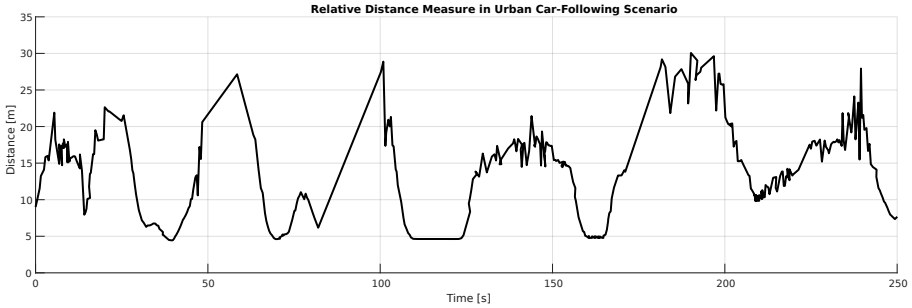

Figure 4.16: Filtering on Lateral Coordinate in Urban Scenario

4.2.3 Robustness of Multiple Object Detection and Distance Measurement

In the third scenario, the main objective is to evaluate the robustness of multiple object detection and distance measurement in the typical urban driving situations. Due to the lack of a ground truth value for the relative distances between the detected objects and the test vehicle, our focus shifts towards assessing the accuracy of the Yolov7 by comparing the detected objects with the actual objects present in the scene, and the reasonableness of the measured distances based on the relative positions and sizes of the detected objects within a specific section of the road.

Therefore, we proposed some criteria to evaluate the performance of the algorithm in this scenario:

- Accuracy of the Yolov7 detection: This criterion involves comparing the number of correct detections and false detections generated by the Yolov7 with the total number of objects present in the scenario. This analysis provides

insights into the system's ability to correctly identify and classify objects, thus understanding its reliability in real-world scenarios.

- Distance representations: In this criterion, we calculate the average distance of all the measured objects based on their respective classes, which helps us understand the average proximity maintained between the test vehicle and various objects encountered during the driving scenarios. This analysis allows us to compare the representation of the Euclidean distance and the longitudinal distance and assess the differences between them, and provides insights into the effectiveness and applicability of each distance representation method in accurately reflecting the distances between objects and the test vehicle.

- Valid ranging ratio: In this criterion, we calculate the ratio of detections with a valid measured distance to all the detections, considering their respective object classes. This metric provides insights into the system's ranging ability for different objects encountered during the testing scenarios.

Typical Neighborhood in Torino

As shown in Figure 4.17 and Table 4.1, the Yolov7 demonstrates a high level of precision in detecting vehicles within a crowded environment. However, for less frequently encountered objects, there may be instances of false detections. For objects with valid distance measurements, the average distances obtained are found to be reasonable. However, it is worth mentioning that the measurements for traffic signs may not always be valid. In terms of distance representation, it is observed that the Euclidean distance tends to be larger than the longitudinal distance.

Class	Detection	False Detection	Total	$\overline{d}_{euclidean}$[m]	$\overline{d}_{longitudinal}$[m]	Valid Ranging[%]
Vehicle	82	0	82	7.59	6.82	91.4
Crosswalk Line	4	1	4	6.79	5.99	30.8
Pedestrian	4	1	5	6.25	5.54	75
Bike	1	1	2	5.67	4.70	100
Bumper Sign	1	0	1	N/A	N/A	0
Speedlimit 40	1	1	1	N/A	N/A	0

Table 4.1: Typical Neighborhood in Torino

Two-Lane Road without Traffic Light

As shown in Figure 4.18 and Table 4.2, the Yolov7 performs even better in a less crowded environment. For objects with acceptable valid distance measurements,

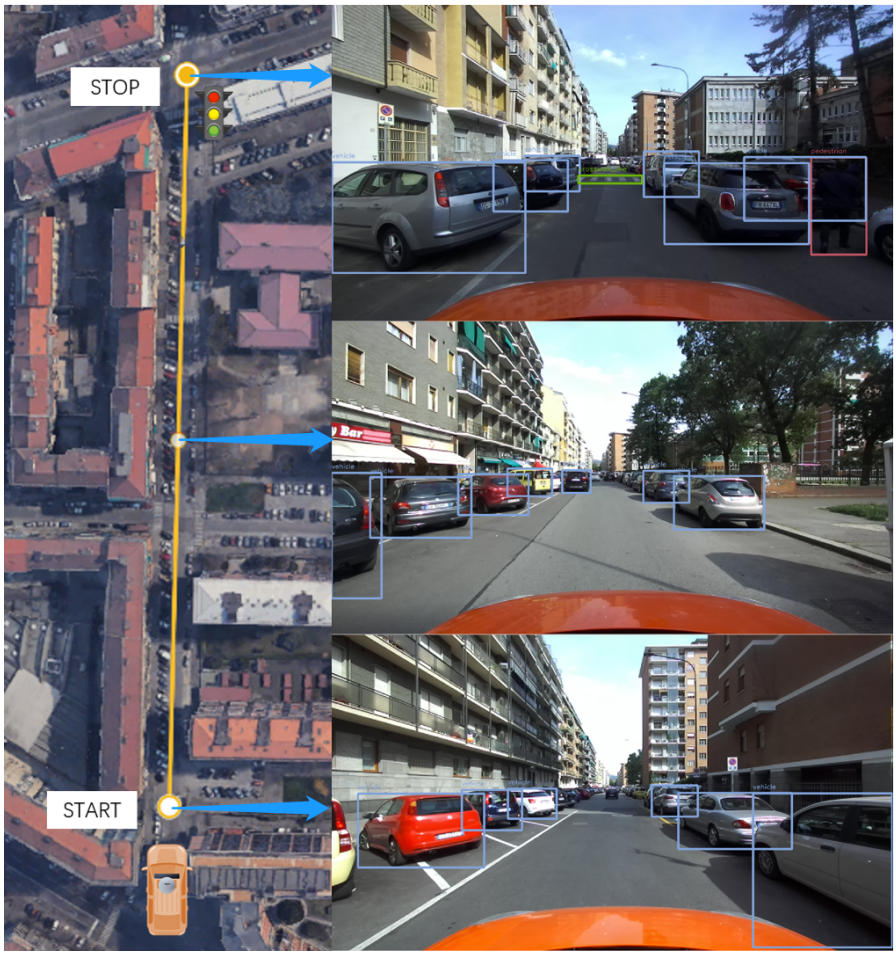

Figure 4.17: Typical Neighborhood in Torino

the average distances are also reasonable. The measurements for traffic signs are still not reliable. As for the distance representation, the Euclidean distance is also larger than the longitudinal distance.

Three-Lane Road with Traffic Light

As shown in Figure 4.19 and Table 4.3, there are less objects since we are driving in a traffic flow, the performance of Yolov7 is stable, but there are still some false detections in the speed limit signs. The average distances are also reasonable, however, the measurements for traffic lights are not available. As for the distance representation, the difference is similar to the previous cases.

Class	Detection	False Detection	Total	$\overline{d}_{euclidean}$[m]	$\overline{d}_{longitudinal}[m]$	Valid Ranging[%]
Vehicle	61	0	61	9.83	9.05	75.5
Crosswalk Line	4	0	4	8.71	8.34	76.9
Pedestrian	6	0	6	7.51	7.19	14.3
Bike	1	0	2	6.55	5.09	25
Crosswalk Sign	5	0	5	14.23	13.95	3.8
Speedlimit 30	1	0	1	N/A	N/A	0
Stop Sign	1	0	1	14.36	13.79	100

Table 4.2: Two-Lane Road without Traffic Light

Figure 4.18: Two-Lane Road without Traffic Light

Summary of Robustness

Through the analysis of the three urban driving scenarios mentioned above, we can draw the following conclusions:

Class	Detection	False Detection	Total	$\overline{d}_{euclidean}$[m]	$\overline{d}_{longitudinal}$[m]	Valid Ranging[%]
Vehicle	23	0	23	11.24	10.88	62.7
Red Light	2	0	2	N/A	N/A	0
Green Light	5	0	5	N/A	N/A	0
trafficlight NA	2	0	2	N/A	N/A	0
Crosswalk Line	3	0	3	8.33	7.63	66.7
Pedestrian	4	0	4	5.16	3.86	24
Bike	2	0	2	9.82	8.98	18.2
Yield Sign	2	0	2	12.88	12.59	6.67
Speedlimit 30	0	2	0	N/A	N/A	0

Table 4.3: Three-Lane Road with Traffic Light

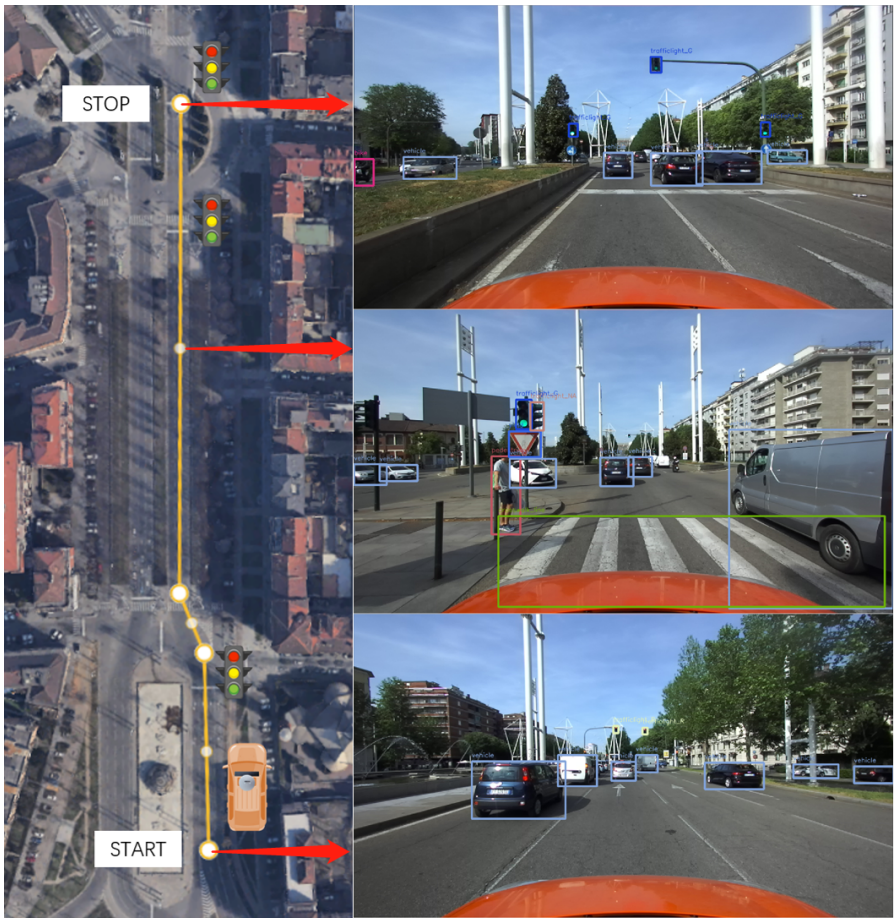

Figure 4.19: Three-Lane Road with Traffic Light

- Accuracy of the Yolov7 detection: The current training of Yolov7 demonstrates reliability in detecting common objects such as vehicles, traffic lights, and

pedestrians. For example, the detection success rate of vehicle and traffic lights is 100%. However, there are some instances of false detection when it comes to traffic signs. This discrepancy suggests that the current training data may not be as comprehensive or specific for accurately detecting and recognizing traffic signs. To address this issue and improve the detection performance of traffic signs, further training of the Yolov7 model with additional dedicated datasets specifically focused on traffic signs is recommended.

- Distance representations: It is observed that when an object is detected by the Yolov7 and has a valid distance measurement, both the Euclidean and longitudinal representations provide reasonable and consistent distance measurements that align with the corresponding scenarios. Based on the tests conducted and the analysis performed, a cautious conclusion can be drawn that, considering the nature of driving scenarios and the need to match the driver's distance perception, the longitudinal distance emerges as a more suitable representation in real-world driving situations.

- Valid ranging ratio: The ranging ability of the system demonstrates variations across different classes of objects. Objects that are relatively large and have a clear line of sight from the LiDAR's perspective, such as vehicles, bikes, and pedestrians near the test vehicle, consistently yield valid distance measurements. For example, In the first two scenarios, where the objects are not obstructed by other objects, the valid ranging ratio exceeds 75%. This indicates that the system successfully provides valid distance measurements for a significant portion of the detected objects. However, in the third scenario, where there are more objects present in parallel lanes, the valid ranging ratio is slightly lower, reaching approximately 65%. This stability in distance measurement validity can be attributed to the prominent presence and unobstructed visibility of these objects. However, for smaller objects, such as traffic lights and traffic signs, the validity of distance measurements may be less reliable. As for the crosswalk lines on the road, they might pose challenges for the ground filtering function, leading to potential difficulties in obtaining valid distance measurements. Additionally, certain vehicles and pedestrians that are obstructed by other objects from the LiDAR's perspective may also exhibit a lower likelihood of obtaining valid distance measurements. These findings indicate that the ranging ability of the system is influenced by the size, visibility, and potential obstructions of the objects being detected. Larger and more visible objects tend to yield more consistent and reliable distance measurements, while smaller or obstructed objects may present challenges in obtaining valid measurements.

- FOV Limitation vs Fusion Time: From Figure we can see that the limitation

of FOV has significantly improved the fusion speed and shortens the fusion time. After limiting the x-axis to cover a range of four meters on each side to focus on the adjacent lanes, we observed a 50% reduction in the number of points that needed to be processed compared to the unrestricted scenario. Additionally, the fusion time was only 10% of the original processing time. This indicates that by applying the FOV limitation, we successfully reduced the number of point cloud data to be processed and significantly improved the fusion processing speed. This FOV filter allows us to balance the trade-off between system reliability and computational efficiency. It enables us to lower the 'min_cluster_size' parameter to detect more clusters and improve the detection capability for distant objects, while maintaining an appropriate balance between system reliability and processing speed. This filtering technique plays a crucial role in the system, providing an effective and reliable solution to enhance the detection of distant objects.

It is important to note that these conclusions are drawn based on the tests conducted in specific driving scenarios. Further validation and testing in various real-world scenarios are necessary to solidify these findings and ensure their general applicability.

Chapter 5

Conclusions and Future Works

With the increasing convergence of artificial intelligence and the automotive industry, the capability of vehicles to perceive their surroundings, process data in real-time, and integrate information has become a crucial aspect of ADAS. This work aims to investigate the feasibility of meeting project requirements for the environmental awareness task in a real ADAS project and propose a practical solution.

The study begins by investigating the theoretical background of each related field, which involves computer vision, sensor technology, and real-time sensor fusion techniques. By gaining a deep understanding of these concepts, the groundwork is laid for the subsequent development of an effective solution. Then the focus shifts to the design of hardware and software architectures, leveraging existing hardware equipment available. Next, we build a reliable software working environment that enables seamless integration between the hardware and software components. Meanwhile, due to specific requirements for the targets to be detected, Yolov7 was trained using a customized dataset prepared by ourselves. Then an algorithm is developed to realize the real-time sensor fusion, which aims to effectively combine data from various sensors, such as LiDAR and camera, with potential scalability, and output meaningful result according to system requirements, as depicted in Figure 5.1, enabling a comprehensive perception of the vehicle's surrounding environment.

Once the software and hardware components are interconnected and functional, the system is installed in a test vehicle. Real-world testing is conducted in diverse application scenarios to evaluate the performance and capabilities of the system. Throughout the testing phase, data is collected and analyzed to assess the system's performance, reliability, and adherence to project requirements. This evaluation serves as a foundation for identifying areas of improvement, refining algorithms, enhancing the overall performance of the system, and proving the validity and

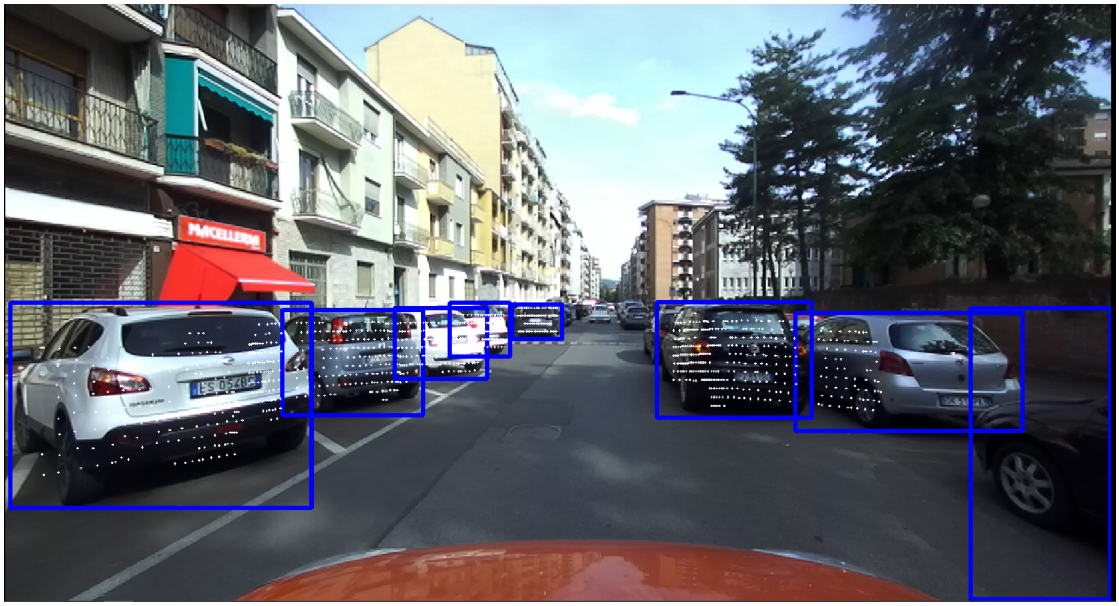

Figure 5.1: Visualization of the sensor fusion result

reliability of integrating the system to the ADAS system.

Based on the test results, several key conclusions can be drawn. First, the Yolov7 object detector demonstrates impressive performance and is well-suited for real-time and customized object detection tasks. Its accuracy and reliability make it a valuable choice for ADAS applications. Additionally, the preprocessing of point cloud data proves to be essential in ensuring accurate and stable detection and distance measurement in real-world driving scenarios. While this preprocessing operation may reduce the effective detection distance slightly and increase processing time, it is crucial for achieving reliable results in complex environments. The trade-off between accuracy and efficiency is well-justified in this context. Regarding distance representations, the longitudinal distance proves to be more aligned with the driver's perception of distance in real-world driving scenarios. This finding suggests that using the longitudinal distance, which represents the distance in front of the vehicle along the y-axis, provides a more intuitive and reasonable measure for the driver. Furthermore, when representing an object's distance, using the nearest distance rather than the average distance appears to be more appropriate. This approach ensures that the closest point to the vehicle, which is more likely to reflect the object's position, is considered as the representative distance. This choice enhances the overall accuracy and reliability of the distance measurement.

This study outlines the process of exploring, designing, implementing, and evaluating an ADAS project. By establishing the theoretical basis, developing hardware and software architectures, and conducting real-world testing, the study aims to

demonstrate the feasibility and effectiveness of the environmental awareness system in meeting project objectives. The collected data and analysis provide valuable insights for further development of the ADAS system, offering an opportunity to enhance the safety and capabilities of vehicles in real-world driving scenarios.

This study has laid a strong foundation for future development and improvements in the ADAS system. There are several potential areas for further enhancements from both a hardware and software perspective.

On the hardware side, upgrading the sensors can improve the system's ranging ability and extend the detection range. This can be achieved by incorporating more advanced LiDAR sensors, radars, or additional cameras to provide a more comprehensive view of the vehicle's surroundings. The integration of these sensors can enhance the accuracy and reliability of the system's perception capabilities.

From a software standpoint, further development of the fusion algorithm is crucial. The inclusion of an object tracking function can greatly enhance the system's performance by enabling the tracking and prediction of object movements. This functionality can contribute to improved decision-making and proactive responses to evolving traffic situations. Additionally, leveraging the image stream and point cloud data to generate a visually appealing and intuitive visualization for the driver can enhance situational awareness. By presenting the surrounding environment in a clear and informative manner, drivers can make more informed decisions and better understand potential hazards.

In conclusion, this study provides a solid platform for future development in the ADAS system. Upgrading the hardware components, improving the fusion algorithm, and incorporating advanced visualization techniques can further enhance the system's performance, safety, and user experience. Continuous innovation and advancements in this field will contribute to the realization of more sophisticated and effective ADAS systems in the future.

List of Tables

List of Figures

Chapter 6

Algorithm

Algorithm 1 RANSAC algorithm

$N_{data} \leftarrow$ Number of data points
$N_p \leftarrow$ Minimum number of points required by the model
$N_i \leftarrow$ Number of iterations
N_{inlier} ▷ Number of inlier points
$t \leftarrow$ maximum distance threshold of a point to the model to be an inlier
$Best_{inlier} \leftarrow 0$ ▷ Most number of inlier points
$Best_{Model} \leftarrow NULL$ ▷ Best model so far
while not all iterations done **do**
    Draw N_p points randomly;
    Fit a model M to those points;
    $N_{inlier} \leftarrow 0$;
    for each point in the data **do**
        $d \leftarrow$ distance from the point to the model;
        if $d < t$ **then**
            $N_{inlier} \leftarrow N_{inlier} + 1$;
        end if
    end for
    if $N_{inlier} > Best_{inlier}$ **then**
        $Best_{inlier} \leftarrow N_{inlier}$;
        $Best_{Model} \leftarrow M$
    end if
end while
return $Best_{Model}$ ▷ The best model is returned

Algorithm 2 Euclidean Cluster Extraction

```
1: function EuclideanClusterExtraction(P, r, d_th)
2:     Initialize an empty Kd-tree data structure
3:     Initialize an empty list of clusters C
4:     Initialize an empty queue Q
5:     for each point p_i in P do
6:         Create a new cluster C_i
7:         Add p_i to Q
8:         Add p_i to C_i
9:         while Q is not empty do
10:            Remove the first point p_i from Q
11:            Search for the set P_i^k of point neighbors of p_i within a sphere of
    radius r < d_th
12:            for each neighbor p_i^k in P_i^k do
13:                if p_i^k has not been processed then
14:                    Add p_i^k to Q
15:                    Add p_i^k to C_i
16:                end if
17:            end for
18:        end while
19:        Add C_i to C
20:    end for
21:    return C
22: end function
```

Algorithm 3 Sensor Fusion Function

```
1: procedure SensorFusion(Pointcloud msg, Detection msg)
2:     if Detection msg is not empty then
3:         Filter Pointcloud by x-axis to limit the FOV
4:         Filter Pointcloud by z-axis to remove ground points
5:         Downsample Pointcloud using VoxelGrid
6:         Cluster Pointcloud using EuclideanClusterExtraction
7:         for each point p in preprocessed Pointcloud do
8:             Calculate distance and save to vector distances
9:             Save (x, y, z) as cv :: Point3f and save to vector points_3d
10:        end for
11:        Project points_3d to points_2d using cv :: projectPoints
12:        for each detection in Detection msg do
13:            for each p_i^2d in points_2d do
14:                if p_i^2d is inside the bounding box then
15:                    find the corresponding distance in distances according to the
   position i and save to a new vector distance
16:                end if
17:            end for
18:            if distance is not empty then
19:                Do calculations on the distance vector and output necessary data
20:            end if
21:        end for
22:    end if
23: end procedure
```

Chapter 7

Parameters

$$intrinsic = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 521.517 & 0 & 638.569 \\ 0 & 521.517 & 356.64 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

$$rvec = [0.0061, 2.2445, -2.1959] \tag{7.2}$$

$$tvec = [0.0654, -0.0781, -0.0458] \tag{7.3}$$

0	vehicle
1	trafficlight_NA
2	trafficlight_G
3	trafficlight_Y
4	trafficlight_R
5	speedlimit_NA
6	speedlimit_20
7	speedlimit_30
8	speedlimit_40
9	speedlimit_50
10	speedlimit_60
11	speedlimit_70
12	speedlimit_80
13	speedlimit_90
14	speedlimit_100
15	speedlimit_110
16	speedlimit_120
17	speedlimit_130
18	yield
19	stop
20	stop_horizontal
21	bumper_sign
22	crosswalk_sign
23	crosswalk_line
24	pedestrian
25	bike

Table 7.1: Label Classes of Yolov7

Chapter 8

Data Structure

Listing 8.1: sensor_msgs::PointCloud2

```
sensor_msgs/PointCloud2
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
sensor_msgs/PointField[] fields
  uint8 INT8=1
  uint8 UINT8=2
  uint8 INT16=3
  uint8 UINT16=4
  uint8 INT32=5
  uint8 UINT32=6
  uint8 FLOAT32=7
  uint8 FLOAT64=8
  string name
  uint32 offset
  uint8 datatype
  uint32 count
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Listing 8.2: vision_msgs::Detection2DArray

```
vision_msgs/Detection2DArray
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
vision_msgs/Detection2D[] detections
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  vision_msgs/ObjectHypothesis[] results
    int64 id
    float64 score
  vision_msgs/BoundingBox2D bbox
    geometry_msgs/Pose2D center
      float64 x
      float64 y
      float64 theta
    float64 size_x
    float64 size_y
  sensor_msgs/Image source_img
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    uint32 height
    uint32 width
    string encoding
    uint8 is_bigendian
    uint32 step
    uint8[] data
```

Bibliography

[1] *Hybrid electric vehicle*. URL: https://en.wikipedia.org/wiki/Hybrid_electric_vehicle. (accessed: 25.06.2023) (cit. on p. 1).

[2] *Custom Market Insights*. URL: https://www.globenewswire.com/en/news-release/2022/11/14/2555167/0/en/Latest-Global-Hybrid-Vehicle-Market-Size-Share-Worth-USD-1670-Billion-by-2030-at-a-30-CAGR-Custom-Market-Insights-Analysis-Outlook-Leaders-Report-Trends-Forecast-Segmentation-Growt.html. (accessed: 25.06.2023) (cit. on p. 1).

[3] Jordan Tunnell, Zachary Asher, Sudeep Pasricha, and Thomas Bradley. «Toward Improving Vehicle Fuel Economy with ADAS». In: *SAE International Journal of Connected and Automated Vehicles* 1 (Oct. 2018). DOI: 10.4271/12-01-02-0005 (cit. on p. 1).

[4] *What is ADAS (Advanced Driver Assistance Systems)?* URL: https://dewesoft.com/blog/what-is-adas. (accessed: 25.06.2023) (cit. on p. 2).

[5] *Automotive electronics revolution requires faster, smarter interfaces*. URL: https://www.embedded.com/automotive-electronics-revolution-requires-faster-smarter-interfaces/. (accessed: 25.06.2023) (cit. on p. 2).

[6] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104. URL: https://doi.org/10.4271/J3016_202104 (cit. on p. 2).

[7] *LIDAR 101: What is lidar?* URL: https://velodynelidar.com/what-is-LiDAR/. (accessed: 25.06.2023) (cit. on pp. 4, 16).

[8] *What is computer vision?* URL: https://www.ibm.com/topics/computer-vision. (accessed: 25.06.2023) (cit. on p. 4).

[9] *What Is Sensor Fusion?* URL: https://www.aptiv.com/en/insights/article/what-is-sensor-fusion. (accessed: 25.06.2023) (cit. on p. 4).

[10] *HOW DOES RADAR AND THE DOPPLER SYSTEM WORK?* URL: https://www.radars.com.au/articles.php/how-radar-works-t-9. (accessed: 25.06.2023) (cit. on p. 4).

[11] *What is an Ultrasonic Sensor?* URL: https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor. (accessed: 25.06.2023) (cit. on p. 4).

[12] *Global Positioning System.* URL: https://en.wikipedia.org/wiki/Global_Positioning_System. (accessed: 25.06.2023) (cit. on p. 5).

[13] *What is ADAS?* URL: https://www.synopsys.com/automotive/what-is-adas.html. (accessed: 25.06.2023) (cit. on p. 5).

[14] *Piattaforma tecnologica di Filiera Pi.Te.F.* URL: https://www.regione.piemonte.it/web/temi/fondi-progetti-europei/fondo-europeo-sviluppo-regionale-fesr/programmazione-2014-2020/piattaforma-tecnologica-filiera-pitef. (accessed: 25.06.2023) (cit. on p. 5).

[15] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. «A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research». In: *Sensors* 19.3 (2019). ISSN: 1424-8220. DOI: 10.3390/s19030648. URL: https://www.mdpi.com/1424-8220/19/3/648 (cit. on pp. 7, 11, 16).

[16] Luiz G. Galvao, Maysam Abbod, Tatiana Kalganova, Vasile Palade, and Md Nazmul Huda. «Pedestrian and Vehicle Detection in Autonomous Vehicle Perception Systems—A Review». In: *Sensors* 21.21 (2021). ISSN: 1424-8220. DOI: 10.3390/s21217267. URL: https://www.mdpi.com/1424-8220/21/21/7267 (cit. on p. 7).

[17] Chaochao Meng, Hong Bao, and Yan Ma. «Vehicle Detection: A Review». In: *Journal of Physics: Conference Series* 1634.1 (Sept. 2020), p. 012107. DOI: 10.1088/1742-6596/1634/1/012107. URL: https://dx.doi.org/10.1088/1742-6596/1634/1/012107 (cit. on pp. 9, 10, 19).

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection.* 2016. arXiv: 1506.02640 [cs.CV] (cit. on p. 9).

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: *Computer Vision – ECCV 2016.* Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0 (cit. on p. 10).

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV] (cit. on p. 10).

[21] *Key differences between CCD and CMOS imaging sensors*. URL: https://www.flir.eu/support-center/iis/machine-vision/knowledge-base/key-differences-between-ccd-and-cmos-imaging-sensors/. (accessed: 25.06.2023) (cit. on p. 11).

[22] *CMOS vs CCD: Why CMOS Sensors Are Preferred for Machine Vision Cameras*. URL: https://www.phase1vision.com/blog/difference-between-cmos-and-ccd. (accessed: 25.06.2023) (cit. on p. 11).

[23] *Introduction to Machine Vision*. URL: https://datasensor.in/machine-vision/introduction-to-machine-vision/. (accessed: 25.06.2023) (cit. on p. 11).

[24] *CMOS ADVANTAGES OVER CCD*. URL: http://www.siliconimaging.com/ARTICLES/cmos_advantages_over_ccd.htm. (accessed: 25.06.2023) (cit. on p. 11).

[25] *Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c. (accessed: 25.06.2023) (cit. on pp. 12, 32).

[26] *Pinhole camera model*. URL: https://en.wikipedia.org/wiki/Pinhole_camera_model. (accessed: 25.06.2023) (cit. on p. 13).

[27] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685 (cit. on p. 13).

[28] Ashutosh Saxena, Jamie Schulte, and Andrew Y. Ng. «Depth Estimation Using Monocular and Stereo Cues». In: IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2197–2203 (cit. on p. 14).

[29] Aish Dubey. «Stereo vision-Facing the challenges and seeing the opportunities for ADAS applications». In: *Texas Instruments Technical Note* (2016) (cit. on p. 14).

[30] G Ajay Kumar, Jin Hee Lee, Jongrak Hwang, Jaehyeong Park, Sung Hoon Youn, and Soon Kwon. «LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles». In: *Symmetry* 12.2 (2020). ISSN: 2073-8994. DOI: 10.3390/sym12020324. URL: https://www.mdpi.com/2073-8994/12/2/324 (cit. on p. 16).

[31] *A Guide to Lidar Wavelengths for Autonomous Vehicles and Driver Assistance*. URL: https://velodyneLiDAR.com/blog/guide-to-LiDAR-wavelengths/. (accessed: 25.06.2023) (cit. on p. 16).

[32] *Hesai Technology*. URL: https://www.hesaitech.com/. (accessed: 25.06.2023) (cit. on p. 16).

[33] *LiDAR*. URL: https://en.wikipedia.org/wiki/LiDAR. (accessed: 25.06.2023) (cit. on p. 16).

[34] *What is LiDAR and how does it work?* URL: https://geoslam.com/what-is-LiDAR/. (accessed: 25.06.2023) (cit. on p. 16).

[35] *The Basics of LiDAR - Light Detection and Ranging - Remote Sensing*. URL: https://www.neonscience.org/resources/learning-hub/tutorials/LiDAR-basics. (accessed: 25.06.2023) (cit. on p. 16).

[36] *What is lidar?* URL: https://oceanservice.noaa.gov/facts/LiDAR.html. (accessed: 25.06.2023) (cit. on p. 16).

[37] *9 Types of Sensor Fusion Algorithms*. URL: https://www.thinkautonomous.ai/blog/9-types-of-sensor-fusion-algorithms/. (accessed: 25.06.2023) (cit. on p. 19).

[38] *Sensor Fusion - LiDARs and RADARs in Self-Driving Cars*. URL: https://www.thinkautonomous.ai/blog/sensor-fusion/. (accessed: 25.06.2023) (cit. on p. 19).

[39] *LiDAR and Camera Sensor Fusion in Self-Driving Cars*. URL: https://www.thinkautonomous.ai/blog/LiDAR-and-camera-sensor-fusion-in-self-driving-cars/. (accessed: 25.06.2023) (cit. on p. 19).

[40] *Satellite Architecture Basics*. URL: https://www.aptiv.com/en/insights/article/satellite-architecture-basics. (accessed: 25.06.2023) (cit. on p. 19).

[41] *Satellite Architecture*. URL: https://www.aptiv.com/en/solutions/advanced-safety/satellite-architecture. (accessed: 25.06.2023) (cit. on p. 19).

[42] *Sensor fusion levels and architectures*. URL: https://www.sensortips.com/featured/sensor-fusion-levels-and-architectures-faq/. (accessed: 25.06.2023) (cit. on p. 19).

[43] Xin Gao, Guoying Zhang, and Yijin Xiong. «Multi-scale multi-modal fusion for object detection in autonomous driving based on selective kernel». In: *Measurement* 194 (2022), p. 111001. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2022.111001. URL: https://www.sciencedirect.com/science/article/pii/S026322412200272X (cit. on p. 19).

[44] Jing Li, Rui Li, Jiehao Li, Junzheng Wang, Qingbin Wu, and Xu Liu. «Dual-view 3D object recognition and detection via Lidar point cloud and camera image». In: *Robotics and Autonomous Systems* 150 (2022), p. 103999. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2021.103999. URL: https://www.sciencedirect.com/science/article/pii/S0921889021002542 (cit. on p. 19).

[45] Esraa Khatab, Ahmed Onsy, and Ahmed Abouelfarag. «Evaluation of 3D Vulnerable Objects' Detection Using a Multi-Sensors System for Autonomous Vehicles». In: *Sensors* 22.4 (2022). ISSN: 1424-8220. DOI: 10.3390/s22041663. URL: https://www.mdpi.com/1424-8220/22/4/1663 (cit. on p. 19).

[46] Yaodong Cui, Ren Chen, Wenbo Chu, Long Chen, Daxin Tian, Ying Li, and Dongpu Cao. «Deep Learning for Image and Point Cloud Fusion in Autonomous Driving: A Review». In: 23.2 (Feb. 2022), pp. 722–739. ISSN: 1524-9050. DOI: 10.1109/TITS.2020.3023541. URL: https://doi.org/10.1109/TITS.2020.3023541 (cit. on p. 19).

[47] Wenbo Zhu, Quan Wang, Lufeng Luo, Yunzhi Zhang, Qinghua Lu, Wei-Chang Yeh, and Jiancheng Liang. «CPAM: Cross Patch Attention Module for Complex Texture Tile Block Defect Detection». In: *Applied Sciences* 12 (Nov. 2022), p. 11959. DOI: 10.3390/app122311959 (cit. on p. 27).

[48] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV] (cit. on p. 28).

[49] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*. 2020. arXiv: 1805.04687 [cs.CV] (cit. on p. 28).

[50] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. «Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition». In: *Neural Networks* 32 (2012). Selected Papers from IJCNN 2011, pp. 323–332. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.02.016. URL: https://www.sciencedirect.com/science/article/pii/S0893608012000457 (cit. on p. 28).

[51] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. «Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark». In: *International Joint Conference on Neural Networks*. 1288. 2013 (cit. on p. 28).

[52] *Road Signs Dataset*. URL: https://makeml.app/datasets/road-signs (cit. on p. 28).

[53] *Yolo_mark*. URL: https://github.com/AlexeyAB/Yolo_mark. (accessed: 25.06.2023) (cit. on p. 29).

[54] Darren Tsai, Stewart Worrall, Mao Shan, Anton Lohr, and Eduardo Nebot. *Optimising the selection of samples for robust lidar camera calibration*. 2021. arXiv: 2103.12287 [cs.CV] (cit. on p. 32).

[55] Surabhi Verma, Julie Stephany Berrio, Stewart Worrall, and Eduardo Nebot. *Automatic extrinsic calibration between a camera and a 3D Lidar using 3D point and plane correspondences*. 2019. arXiv: 1904.12433 [cs.CV] (cit. on p. 32).

[56] José María Martínez-Otzeta, Itsaso Rodríguez-Moreno, Iñigo Mendialdua, and Basilio Sierra. «RANSAC for Robotic Applications: A Survey». In: *Sensors* 23.1 (2023). ISSN: 1424-8220. DOI: 10.3390/s23010327. URL: https://www.mdpi.com/1424-8220/23/1/327 (cit. on p. 33).

[57] *What Is RANSAC?* URL: https://ww2.mathworks.cn/discovery/ransac.html. (accessed: 25.06.2023) (cit. on p. 33).

[58] *Maths - Conversion Euler to Axis-Angle*. URL: http://euclideanspace.com/maths/geometry/rotations/conversions/eulerToAngle/index.htm. (accessed: 25.06.2023) (cit. on p. 36).

[59] *message_filters Package Summary*. URL: http://wiki.ros.org/message_filters. (accessed: 25.06.2023) (cit. on p. 38).

[60] *message_filters ApproximateTime*. URL: http://wiki.ros.org/message_filters/ApproximateTime. (accessed: 25.06.2023) (cit. on p. 38).