POLITECNICO DI MILANO

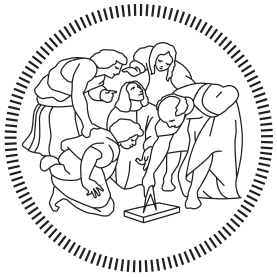
Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering



Segmentation and Tracking of Vehicles via Low Resolution LiDAR Sensors.

Advisor: PROF. MATTEO MATTEUCCI

Co-advisor: DR. SIMONE MENTASTI

Master Graduation Thesis by:

GIORGIO SOFISTI

Student Id n. 10500171

Academic Year 2019-2020

POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in
Computer Science and Engineering

Segmentazione e Tracking di Veicoli attraverso sensori LiDAR a bassa risoluzione.

Relatore: PROF. MATTEO MATTEUCCI

Correlatore: DR. SIMONE MENTASTI

Tesi di Laurea Magistrale di:

GIORGIO SOFISTI
Matricola n. 10500171

Anno Accademico 2019-2020

"I could either watch it happen or be a part of it."
Elon Musk

ACKNOWLEDGMENTS

Many are the people that supported me during my studies at Politecnico. I'd like to express my gratitude to my parents Fabrizio and Paola and to my sister Chiara, without them I could not have embraced this journey that formed me as a professional and as a person.

I'd like to thank Elena, always by my side from the first to the last day of this journey, who supported me in the difficult moments and celebrated with me the happiest ones.

I would like to thank my high-school friends Martina, Anna, Fabio and Andrea for always being there in these ten years and also all the one I have met during these five incredible university years, in particular Elisa, Federico, Davide, Francesco, Costantino and Daniele who shared this experience with me.

One special thanks goes to my grandparents Giancarlo and Bianca who always advised and supported me during these years.

I'd like also to thank Gianmario, a friend that provided me with great challenges and forced me always to explore new ideas.

Last but not least I'd like to thank my advisor, Prof. Matteo Matteucci and my co-advisor Simone Mentasti for giving me the passion and the support in this last step of my training path.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Automation of driving task has become in the recent years the main objective of automotive sector and legislators. Driver assistance technologies in today's vehicles help to save lives, preventing crashes, and improving the driving experience. These technologies allow the identification and reaction, in case of risks for the vehicle, for its occupants and for any other road user. The aim of this thesis is to develop a standalone system based on low resolution LiDAR sensor capable of identifying, classify and track obstacle in the road scene. The algorithm uses Convolutional Neural Network model for object detection and a custom Extended Kalman Filter for tracking. The results are encouraging and show that our solution can perform the task is designed for in real time, with a comparable accuracy to more complex systems but at a fraction of the cost. Moreover, the tracking results guarantee a good evaluation of the driving scenario, particularly useful for situation assessment and for decision making.

SOMMARIO

L'automazione della guida è diventata negli ultimi anni l'obiettivo principale del settore automobilistico e dei legislatori. Le tecnologie di assistenza alla guida nei veicoli di oggi aiutano a salvare vite umane, a prevenire incidenti e a migliorare l'esperienza di guida. Queste tecnologie consentono l'identificazione e la reazione, in caso di rischi per il veicolo, per i suoi occupanti e per qualsiasi altro utente della strada. Lo scopo di questa tesi è quello di sviluppare un sistema autonomo basato su sensore LiDAR a bassa risoluzione in grado di identificare, classificare e tracciare gli ostacoli nella scena stradale. L'algoritmo utilizza il modello di rete neurale convolutiva per il rilevamento di oggetti e un filtro Kalman esteso personalizzato per il tracciamento. I risultati sono incoraggianti e dimostrano che la nostra soluzione può svolgere l'attività per cui è stata progettata in tempo reale, con una precisione paragonabile a sistemi più complessi ma a una frazione del costo. Inoltre, i risultati del tracking garantiscono una buona stima dello scenario di guida, particolarmente utile per la valutazione della situazione e per il processo decisionale.

1 INTRODUCTION

A self-driving car, also known as an Autonomous Vehicle (AV), driverless car, or robo-car is a vehicle that is capable of sensing its environment and moving safely with little or no human intervention.

Self-driving cars combine a variety of sensors to perceive their surroundings, such as radar, LiDAR, sonar, Geo Positioning System (GPS), odometry and inertial measurement units. Advanced control systems interpret sensor information to identify appropriate navigation paths, as well as obstacles and relevant road marks.

Autonomous vehicle is considered the most disruptive and impactful technology of the XXI century and all the big automotive and tech companies are currently developing all the technologies and hardware required to tackle this non-trivial problem.

Self-driving technology is expected to revolutionize different sector and is seen as the natural evolution of the road vehicles. In the near future AV technologies will become more and more accessible and common in everyday life and will enable cheaper, safer and easier means of transport and shipping and will radically change the way we use and know them. However There are still many technologies that has to be invented and developed to allow its widespread adoption.

1.1 MOTIVATION

In the recent years thanks to regulation authority's enforcement and technological advancement the automotive industry has been forced to implement new system to increase vehicle safety and the overall sector is pushing towards full autonomous driving systems. Cars are one of the most dangerous way of transportation. According to European Commission report on road safety [1] 25100 people died in 2018 and many more remained severely injured for road traffic fatalities just in EU. For comparison, in the same period, only 885 people have lost their life in train disasters [2] and much less in airplane accidents according to the Annual Safety Review of the EASA [3].

Even if the figures have decreased by 30% in the last decade thanks to passive and active safety systems integrated in vehicles there is still a long road ahead of us and the ambitious goal of cutting by 50% fatalities by 2020 has not been reached despite the unilateral effort by car manufacturer and legislator.

From the data of EU [1] we know that at least 94% of the incident are caused by human errors in Europe and in US is not any better. It is clear that Advanced Driver Assistance Systems also known as ADAS and autonomous driving technology can help reduce those figures by removing the weak point in the loop, the human. Moreover, those system can reduce stress on the driver and let him free from the driving load. Additionally, it's unbelievable the amount of time people lose driving a vehicle from point a to point b.

Statistics says that we spend in average 18 days per year [5] inside a car but thanks to AV we can use this time in a better way. We can also re-imagine the way we move and live our cities by using the vehicle as a transportation medium that is able to bring us to the destination and then autonomously drive to a parking location so that cities can be freed up from cars and traffic. Furthermore, we can think at last mile deliveries, usually the most polluting and time-consuming ones carried out by specific autonomous small electric autonomous vehicles, a faster, cheaper, and more eco-friendly solution.

1.2 OBJECTIVES

While autonomous vehicle technology appears to be developing at a fast pace, no commercially available vehicles have yet reached the stability and reliability required for the highest level of driving autonomy. Nowadays we have systems on the market capable of different levels of automation, but no one is capable of full self-driving in every condition and situation.

System like Tesla autopilot 1.0 can drive autonomously on highways but struggle in city centers, problem that is expected to be solved with the next generation of vehicles.

To make the additional step and fill the gap between robot cars and human drivers a huge amount of technological improvement needs to be taken in serious consideration by manufacturers in order to ensure autonomous vehicle safety on public roads.

For the reasons mentioned above we decided with this work, to develop a modular and reliable system able to detect and prevent accidents leveraging on a low-cost sensor and a new technique. Our main goal is to solve the hard problem of detection and tracking for self-driving vehicles in a complex environment by using the robust output of a LiDAR sensor. The idea is to develop different modules that execute different computation steps that transform raw data into aggregated, highly informative and human comprehensive evidence with all the information required to carry out the driving task. Moreover, during the work, we benchmarked different LiDAR sensors with different specification to understand which is the best trade-off between accuracy and performance and which is the lower limit in terms of cost and resolution to achieve good results. We finally tested different tracking algorithm.

1.3 THESIS OUTLINE

This thesis is organized into 5 chapters. The first one is a brief introduction to the main macro theme. In the second Chapter we will describe more in depth how to classify and analyze autonomous vehicles by presenting a brief Historical tour on the development of this technology from the beginning to the present days. The chapter than presents the enabling hardware and software for autonomous vehicles and ends with a in depth analysis of the state-of-the-art approaches currently in use for LiDAR segmentation and object tracking. In chapter three then we will present our solution analyzing system architecture and project choices. In the fourth chapter we will introduce the experimental setup used for solution validation and testing and we will present the results obtained. The last chapter will be dedicated to the conclusion, the evaluation of system limits, possible improvements, and future addition. The chapter ends with the potential application of the system.

2

STATE OF THE ART

In this chapter we present the state of the art regarding autonomous driving focusing on perception pipeline and how sensor data are handled and used on the current best performing autonomous vehicles, we analyze different works concerning development of those systems comparing pros and cons of each approach.

First, we give a general overview description of what autonomous driving technology is, what it does and how. We present and explain the different autonomous driving levels. Then, we give a background about the history of autonomous vehicles following the biggest milestones achieved in the past years and analyzing the technological breakthrough that made it possible concluding with the present days and examining the current level of AVs. Moreover, we give a brief overview on what is missing for full automation and what the next developments can bring to the sector showing the possible scenarios that can change our lives.

We then dive deeper in the perception pipeline giving a better understanding on software and hardware architecture and integration analyzing which are the differences between a standard and an autonomous vehicle. We present the data flow from the acquisition to the generation of aggregated data and how that information can be used to take informed decisions.

We conclude this chapter presenting Machine Learning (ML) and Deep Learning (DL) techniques since are at the basis of our research work.

2.1 AUTONOMOUS DRIVING: LEVELS OF AUTOMATED DRIVING SYSTEMS

Before describing the existing projects concerning autonomous driving, we have to define what an autonomous vehicle is. Autonomous vehicles, also known as AV, are a specific class of robots capable of performing, with different levels of autonomy, the principal task of moving the vehicle itself and the cargo in it from a given starting point to a given destination. This is achieved through a specific set of hardware and software capable of substituting the human driver.

The Society of Automotive Engineers (SAE) defines 6 levels of driving automation ranging from 0 (fully manual) to 5 (fully autonomous) [4]. These levels have been adopted by the U.S. Department of Transportation and are today considered the standard classification for AV.

Depending on the human interaction in the driving task we have:

- **Level 0:** Manually controlled. The human controls the "dynamic driving task" although there may be systems in place to help the driver. An example would be the emergency braking system—since it technically does not "drive" the vehicle, it does not qualify as automation.
- **Level 1:** The vehicle features a single automated system for driver assistance, such as steering or accelerating. Adaptive cruise control, where the vehicle can be kept at a safe distance behind the next car, qualifies as Level 1 because the human driver still monitors and provide inputs for the other aspects of driving such as steering and braking.
- **Level 2:** This means advanced driver assistance systems or ADAS. The vehicle can control both steering and acceleration/deceleration input to the vehicle. Here the automation falls short of self-driving because a human is in the driver's seat and can take control of the car at any time. Tesla Autopilot and Cadillac Super Cruise systems both qualify as Level 2.
- **Level 3:** Vehicles have "environmental detection" capabilities and can make informed decisions for themselves, such as accelerating past a slow-moving vehicle. Those kinds of systems however still require human override if something goes wrong. The driver must remain alert and ready to take control if the system is unable to execute the task.

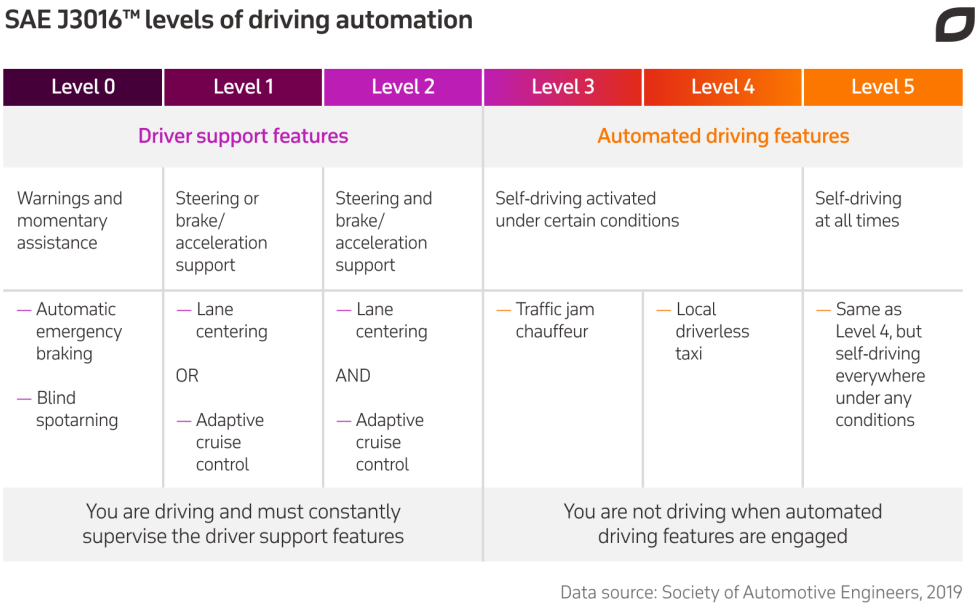


Figure 2.1: SAE Autonomous Driving Levels Classification

- **Level 4:** Vehicles can intervene if things go wrong or there is a system failure. In this sense, these cars do not require human interaction in most circumstances. However, a human still has the option to manually override system decisions. Level 4 vehicles can operate in full self-driving mode but until legislation and infrastructure evolves, they can only do so within a limited area (usually an urban environment where top speeds reach an average of 50Km/h).
- **Level 5:** Do not require human attention—the "dynamic driving task" is eliminated. Level 5 cars will not even have steering wheels or acceleration/braking pedals since are totally useless. They will be able to go anywhere and do anything that an experienced human driver can do without any external help.

2.2 AUTONOMOUS DRIVING: HISTORY

Here we want to present brief overview on the history of autonomous vehicles to give a background to the reader about the evolution in time from the first prototypes to the latest state of the art vehicles currently under testing.

2.2.1 *Precursor*

Since the beginning of the XX century have been conducted experiments on self-driving cars. In US in 1925, Houdina Radio Control demonstrated the radio-controlled "American Wonder" on New York City streets, a car specifically modified and capable of traveling up Broadway and down Fifth Avenue through the thick of a traffic jam. It was not an autonomous vehicle but instead was controlled wirelessly by an operator in a nearby car, it was the first vehicle with remote control and electronic actuation.

For the first truly autonomous vehicles we have to wait 30 years. In the 50s, in America has been shown the first autonomous capable car. After some scale models, in 1957 a full-size system was successfully demonstrated by RCA Labs and the State of Nebraska on a 120m strip of public highway. This vehicle, thanks to a series of sensors embedded in the road and radio receivers on the car, was capable of driving, following the preceding vehicle and reacting to changes in speed while following the road. Of course, this solution was quite rudimental, not particularly robust, and expensive since required modification not only to the vehicle itself but also to the road infrastructure.
In Europe similar experiments were carried out from 1960 to mid-1970.

2.2.2 *First attempts*

It is only at the beginning of the 80s that the ideas of a senosrized infrastructure left the space to a self-contained fully autonomous vehicle. This nontrivial step was unlocked thanks to new technologies like digital cameras and new, more powerful but efficient computing machine that could be integrated on the vehicle and had enough power to elaborate sensor data locally. The first vehicle of this kind was the Mercedes-Benz robotic van, designed by Ernst Dickmanns and his team at the Bundeswehr University Munich in Munich, Germany. During the test, the vehicle achieved a speed of 63 km/h on streets without traffic, performed different tests on overtakes and on safety features always in a controlled experimental environment. Thanks to those achievements many more similar projects started: in Italy the ARGO Project, the European Union founded EUREKA Prometheus Project and in the United

States DARPA (Defense Advanced Research Projects Agency) established the Autonomous Land driven Vehicle (ALV) project. The latter achieved the first road-following demonstration of LIDAR technology, computer vision and autonomous robotic control to direct a robotic vehicle at speeds of up to 31 km/h.
In 1987, HRL Laboratories demonstrated the first off-road map and sensor-based autonomous navigation on the ALV. The vehicle traveled over 610 m at 3.1 km/h on complex terrain with steep slopes, ravines, large rocks, and vegetation. By 1989, Carnegie Mellon University had pioneered the use of neural networks to steer and otherwise control autonomous vehicles, forming the basis of contemporary control strategies.

2.2.3 *Past Years*

Modern concept of autonomous vehicle and first robust and concrete results have been achieved after 2000. We clearly identify the DARPA Grand Challenge 2004 as the exact moment in which the method to solve the AV problem drastically changed turning towards the current approach. The first edition of the challenge was held on March 13, 2004 in the Mojave Desert region of the United States, along a 240 km route. The stakes were high but none of the robot vehicles finished the route. Carnegie Mellon University's Red Team and car Sandstorm traveled the farthest distance, completing 11.78 km of the course before getting hung up on a rock after making a switchback turn. This edition of the competition demonstrated that the exact control strategies hard-coded from the human experience are not enough to solve the general AV driving problem since the variability and the unpredictability of the driving scenario cannot be robustly handled and embedded a-priori in the code. No winner was declared, and the cash prize was not given. Therefore, a second DARPA Grand Challenge event was scheduled for 2005. This second round of the DARPA Grand Challenge began on October 8, 2005. All but one of the 23 finalists in the 2005 race surpassed the 11.78 km distance completed by the best vehicle in the 2004 race and five vehicles successfully completed the 212 km course.

The third competition of the DARPA Grand Challenge, [6] known as the "Urban Challenge", took place on November 3, 2007 at the site of the now-closed George Air Force Base, in Victorville, California. The course involved a 96 km urban area, to be completed in less than 6 hours. Rules included obeying all traffic regulations while negotiating with other agents and obstacles. Tartan Racing claimed the $2 million prize with their vehicle "Boss".
The Urban Challenge required designers to build vehicles able to obey all traffic laws while they detect and avoid other robots on the course. This is particularly challenging for vehicle software, as vehicles must make "intelli-

Figure 2.2: The autonomous vehicle of Stanford university named "Stanley", winner of the 2005 challenge during the competition

gent" decisions in real time based on the actions of other vehicles. Other than previous autonomous vehicle efforts that focused on structured situations such as highway driving with little interaction between the road actors, this competition operated in a more cluttered urban environment and required the cars to perform sophisticated interactions with each other, such as maintaining precedence at a 4-way stop intersection.

After those challenges many companies started investing on the development of autonomous driving technology and many startups were founded, the revolution of AV has begun.

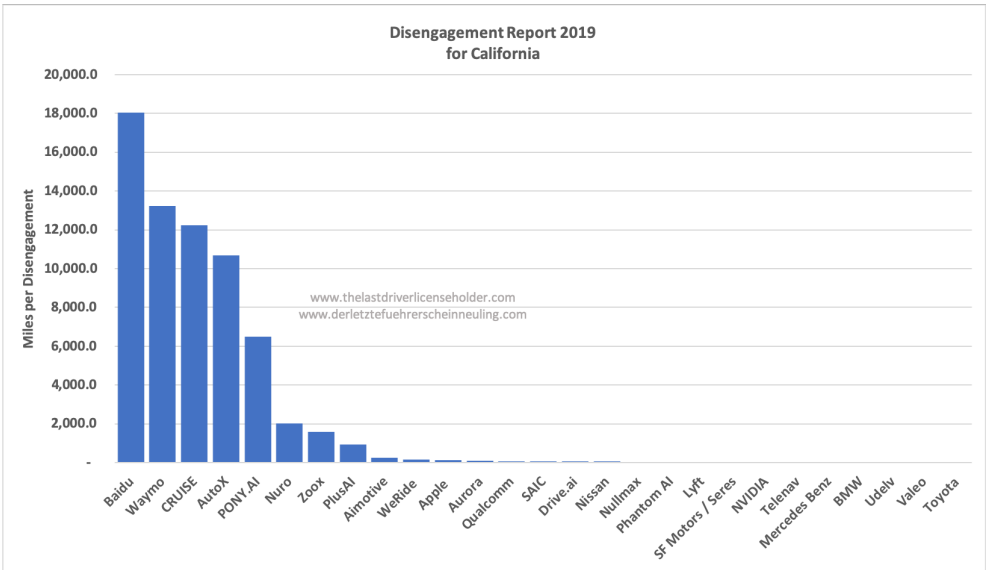


Figure 2.3: Disengagement report for major Av players in California during testing in 2019: Mile driven before disengagement

2.3 CURRENT STATE

Currently the market is developing fast, today we already have different vendors offering on premium products commercial solution capable of autonomously driving in specific settings up to Level 2 and in many cities are under testing prototype vehicles with higher autonomy ratings. Here we present the most advanced solution from the biggest players in this sector.

To correctly compare the performance of an autonomous vehicle has been developed a performance metric called disengagement rate which is the total amount of km driven by a vehicle in autonomous mode before requiring human intervention.

2.3.1 *Tesla*

When we think to autonomous driving the first company that comes to our mind is for sure Tesla. The company is developing its autonomous driving software and hardware since 2013. In the beginning the system, called Autopilot 1.0, was developed in collaboration with Mobileye, a company leader in this sector. It was intended as an advanced ADAS capable of lane keeping on highways combined with an adaptive cruise control plus some additional safety features. After the acquisition of the company by Intel, Tesla decided to move to the much promising and powerful NVIDIA hardware that powered the Autopilot 2.0 and allowed the car to perform Level 2 driving but with some limitation, it was introduced the lane change, collision avoidance and the navigate on autopilot feature capable of full self-driving on highways. In 2020 the system received the biggest upgrade ever moving to the Autopilot 3.0. In this latest iteration the system is installed on every Tesla and is based on a custom FPGA (Field Programmable Gate Array) developed in house by Tesla, a series of cameras all around the vehicle and different radars. The system is advertised as full self-driving capable and in the latest quarter of 2020 first US users received Full Self Driving Beta (FSD) software that is capable of driving autonomously in almost every situation. The human driver still must monitor the system constantly, but the impression is that the system is pretty capable and can be categorized as Level 3. Tesla plans to release the FSD functionality to all his vehicle worldwide in late 2021.
The main advantages of Tesla are the database of more than 3 billion miles driven recorded by a fleet of almost 1 million vehicles in rapid growth, the full control over hardware and software and the deep learning approach over the full pipeline. Tesla is the biggest AV payer in terms of vehicle fleet and hours driven but is also one of the few that do not use LiDAR sensor considered useless by company CEO Elon Musk that stated back in 2019:

> *"Anyone [AV developer company] relying on LiDAR is doomed"*

The car tries to mitigate the absence of the LiDAR leveraging more on radar data to assist cameras in all the situation where cameras are not enough.

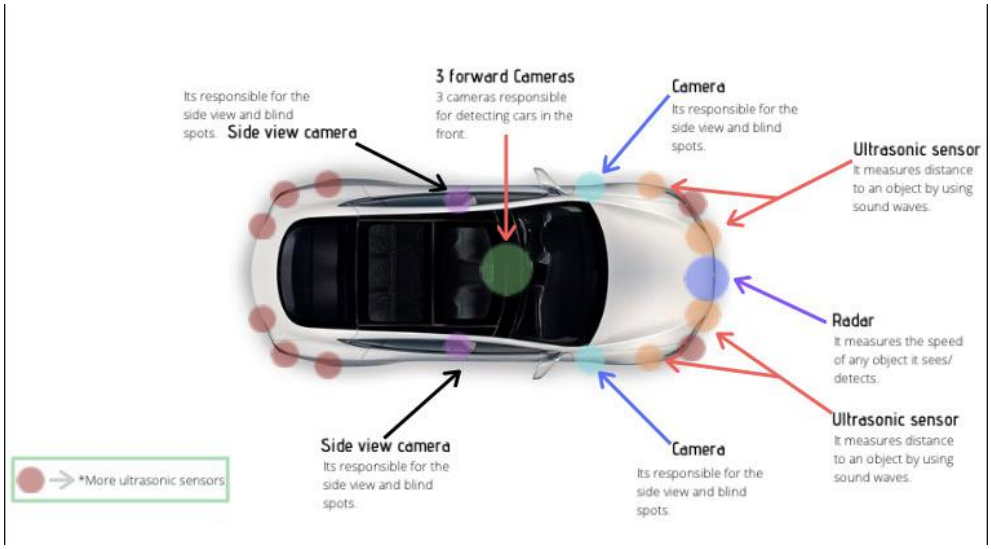


Figure 2.4: FSD Sensor suite on Tesla Model 3

2.3.2 *WAYMO*

In 2016 after the massive company restructuring Google, today Alphabet, decided to group all the AV development under the hat of WAYMO. Google was already active in the autonomous driving sector since 2009 inside the secrets X-lab run by Sergey Brin, one of the tree company co-founders.
The company has the aim of developing hardware and software to integrate in production vehicles to give the capabilities of full autonomous driving and has multiple partnership with big automotive companies like Daimler AG, FCA (Fiat Chrysler Automobile), Jaguar Lan Rover, Volvo and Nissan Renault.
Thanks to the access to Google Maps data and the power of Google Cloud the company has developed both vehicles prototypes and a simulation environment so that the iteration of the software can be tested with near reality accuracy in a virtual world before roll-out. Thanks to this approach Waymo achieved one of the lowest disengagement rate compared to competitors. The system designed by Google is composed of high level, and therefore high cost, but redundant sensor suite. The system in its latest iteration [8] has 3 LiDARs, 3 radars and multiple cameras all around the vehicle. Thanks to his high reliability in late 2018, the company launched a commercial self-driving car service called "Waymo One"; users in the Phoenix metropolitan area can use an app to request a pick-up. By November 2019, the company was operating autonomous vehicles without a safety backup driver and was the first service worldwide to have a remote tele-operator connected only if the system gets sucked at some point.

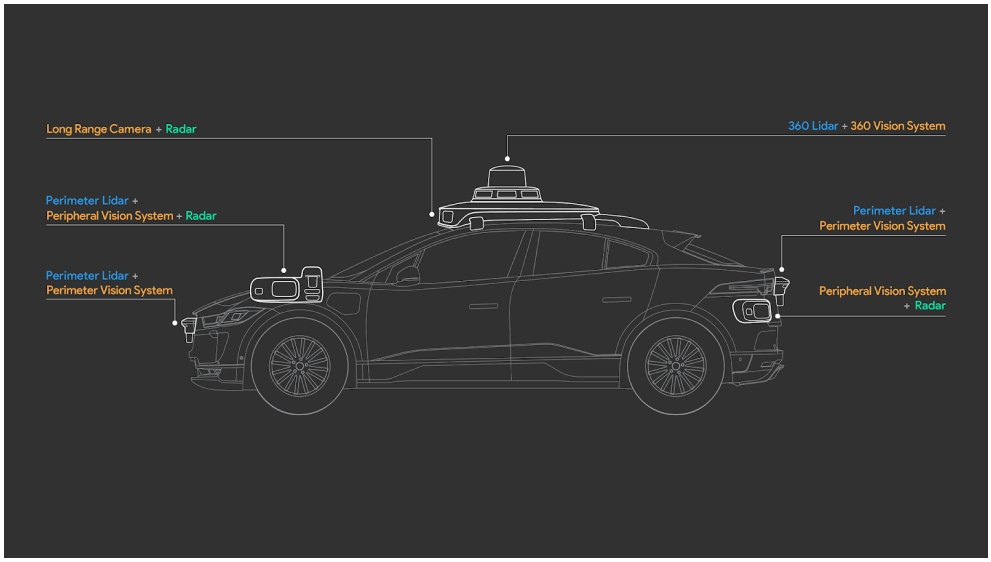


Figure 2.5: WAIMO experimental vehicle sensor setup

Figure 2.6: Comma.ai hardware installed on the windscreen of a compatible mass production vehicle

2.3.3 *Comma.ai*

The start-up, founded by the famous hacker George Hotz in 2015, is developing OpenPilot. The system is currently capable of autonomous driving of Level 2 in specific non complex situations and is composed by a commercially available android smartphone embedded in a custom case that has to be mounted on the windscreen and act both as a sensor and as main computing unit on which runs the code.
The system is compatible with multiple commercially available cars and is based on already integrated sensors in the vehicle like the radar and cameras mixed with the on board cameras of the OpenPilot hardware. The system has already driven over 30 million miles on over 2000 devices active on average every day. The figures are limited compared with the other player, but the results and the development phase is really high.
The main peculiarities of the system are the fact that is fully open-source and pretty cheap, is based on camera vision and do not require additional, expensive hardware or big modification to the vehicle except from the smartphone installation and connection with the car systems that can be done by the driver in a couple of minutes, it is already compatible with commercially available cars and has a feature that allows the monitoring of the behavior of the driver since Level 2 autonomy still requires a constant human scene monitoring.

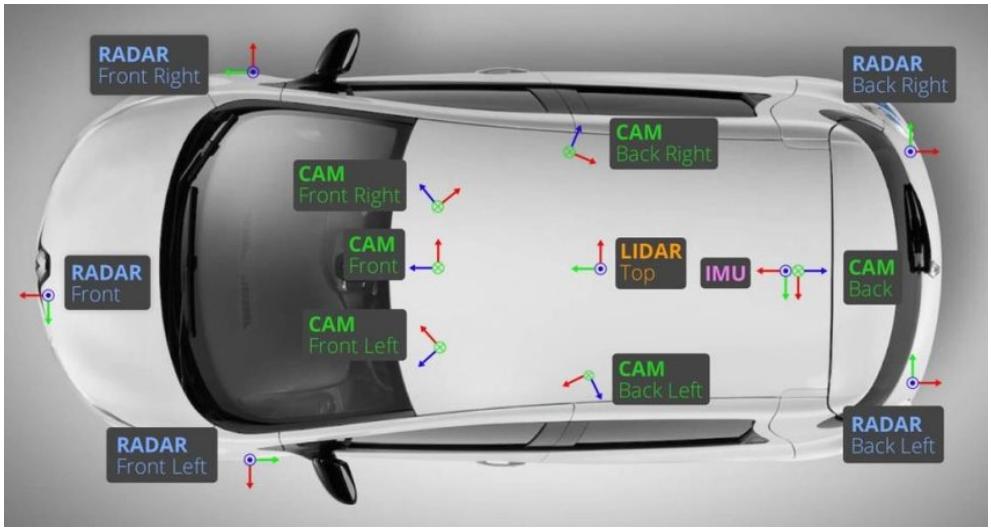


Figure 2.7: NuTonomy AV prototype sensor architecture

2.3.4 *NuTonomy*

Founded in 2013 by professors and students from MIT, NuTonomy is active in the sector of autonomous vehicles and robots. It has been the first company to publicly demonstrate a fleet of autonomous vehicles in 2016 in Singapore as an autonomous taxi service. The aim of the company, similarly to WAIMO, is to develop the technology not a full vehicle. The sensor suite is composed by a LiDAR on top of the vehicle, 5 radars and 6 cameras that combined give a full 360 degrees view on the surrounding. From 2017 NuTonomy is part of Delphi Automotive PLC, one of the biggest automotive suppliers in the world active in electrification and vehicle advance safety technologies. NuTonomy has publicly released part of their data record and created different challenges to further increase the hype in the field and speedup the development of this kind of system.
From 2018 NuTonomy has been publicly testing in Boston his autonomous prototypes based on the electric car Renault Zoe continuously improving the system reliability and safety.

2.3.5 *UBER/Aurora*

The company famous for having revolutionized the transport sector invested a big part of his income to develop the next step, an autonomous taxi system. In early 2015, the company hired approximately 50 people from the robotics department of Carnegie Mellon University and created the new branch focused on Autonomous vehicles. On September 14, 2016, Uber launched its first self-driving car services to select customers in Pittsburgh, using a fleet of Ford Fusion cars. Each vehicle was equipped with 20 cameras, 1 Global Positioning System, 1 LiDAR, and different radars, equipment that enabled the car to create a three-dimensional map of the surrounding space. On December 14, 2016, Uber began using self-driving Volvo XC90 SUVs in San Francisco. On December 21, 2016, the California Department of Motor Vehicles revoked the registration of the vehicles Uber was using for the test and forced the program to cease operations in California. Two months later, Uber moved the program to Arizona, where the cars were able to pick up passengers, although, as a safety precaution, two Uber engineers were always in the front seats of each vehicle. In March 2017, an Uber self-driving car was hit and flipped on its side by another vehicle that failed to yield. In November 2017, Uber announced a non-binding plan to buy up to 24,000 Volvo XC90 SUV vehicles designed to accept autonomous technology, including a different type of steering and braking mechanism and sensors. In December 2020, Uber announced the sale of ATG to Aurora. The company also stated it will invest $400 million into Aurora's research of self-driving technology. Uber itself is pulling out of the self-driving market.

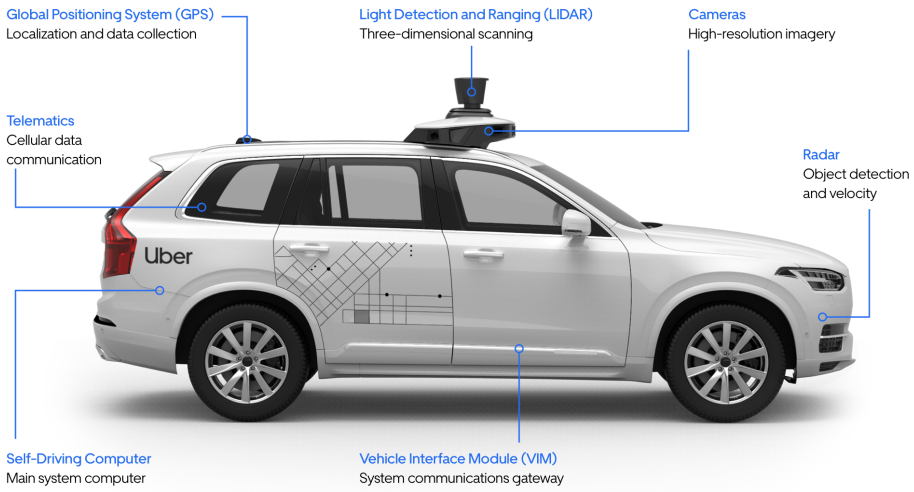


Figure 2.8: UBER AV prototype based on Volvo XC90 car

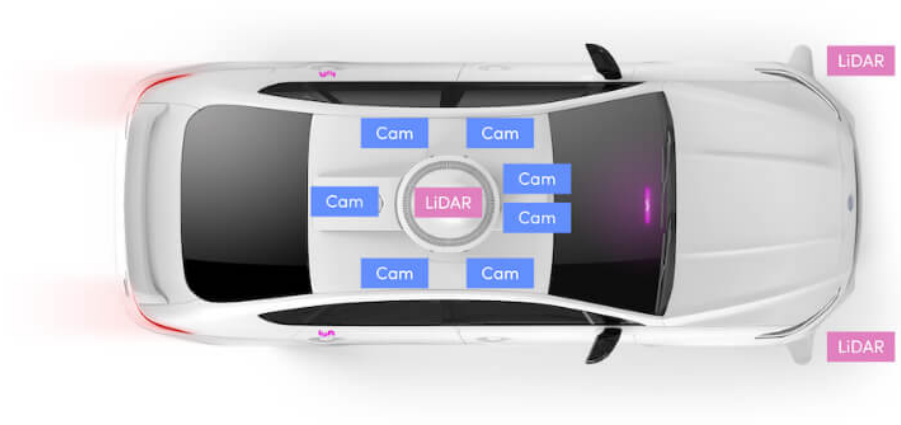


Figure 2.9: Lyft sensor architecture

2.3.6 *Lyft*

In 2012, the biggest UBER competitor in America, started to look at self-driving cars as part of Lyft's future offering. In January 2016, Lyft announced an autonomous car partnership with General Motors and planned to begin testing self-driving cars within one year. To speed up the development Lyft announced a new partnership with NuTonomy to eventually put autonomous, on-demand vehicles on the road. In September 2017, Lyft partnered with Ford Motor Company to develop and test autonomous vehicles.
The development of the autonomous driving system is continuing thanks to the partner with the state of California that approved road testing of Lyft level 4 prototypes. The company is planning to launch an autonomous fleet of taxi by 2023.
The system in the current iteration is composed by a big component integrated on the vehicle roof containing the main processing unit and a suit of sensors among which we can find a 64 plane LiDAR and 7 cameras that combined with the car built-in radar and two additional, lower resolution 16 layers LiDAR in the front one at each corner provide a full 360 degrees view of the vehicle surrounding [7].
The idea of Lyft is to build hardware and software in a single module that can be easily integrated with commercially available vehicles so the company can focus more on the technology rather than building the car itself.

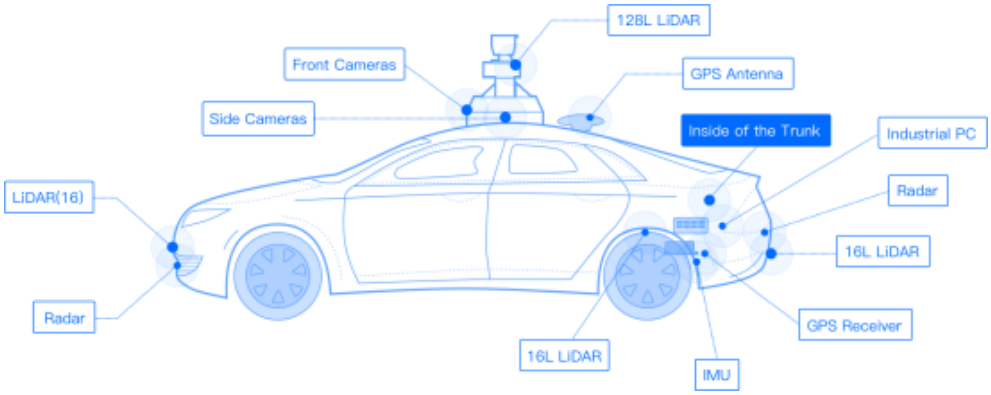


Figure 2.10: Baidu sensor architecture

2.3.7 *Baidu*

In 2013, Baidu, the biggest research engine in China, commenced the development of autonomous driverless vehicles, through the Baidu research institute. This project gradually expanded and now includes 10,000 developers working on an open platform, and more than 50 partners across the world, including Intel, BMW, Benz, Kinglong and XTE. In August 2017, an early version of the autonomous vehicle was unveiled at GoMentum, USA.
In July 2018, test drives had been carried out in Chongqing and Fujian. In July 2018, a series of buses had commenced mass production. The bus was first opened to an international audience in November 2018 at the Shanghai World Expo, where 1600 participants from 72 countries were ushered to the venue in those autonomous buses. The company rely on a standard sensor suite that include a high-resolution LiDAR on top and 4 lower resolution one at each side plus forward and backward radar and multiple cameras. Thanks to the strategic partners the company tested the system in different scenarios and on different vehicles with success. Recently Baidu has been granted the authorization for testing their system in public roads in California.

2.3.8 *ZOOX/Amazon*

Zoox was founded in 2014 by Australian artist-designer Tim Kentley-Klay, and Jesse Levinson, son of Apple Inc. and chairman Arthur D. Levinson, who was developing self-driving technology at Stanford University. Differently to all the other competitors the start-up, based in California, is creating an entirely new autonomous vehicle targeted at the robo-taxi market. The company's approach is centered around the fact that a retrofitted vehicle is not optimized for autonomy. ZOOX has applied the latest techniques in automotive, robotics and renewable energy to build a symmetrical, bi-directional battery-electric vehicle that solves for the unique challenges of autonomous mobility.
The company had previously retrofitted Toyota Highlanders with their self-driving system in final preparation for their commercial vehicle reveal in December 2020. The present-day test driving is taking place in both San Francisco's Financial District and North Beach districts, as well as Las Vegas. Since 2020 the company is part of Amazon Inc., the merger agreement under which Amazon acquired Zoox is evaluated over $1.2 billion. This agreement allowed the start-up to receive all the required resources both in terms of money and knowledge required for such an ambitious plan and pushed the accelerator on the development phase. Last September ZOOX became the fourth company in the State of California to receive permit to test driver-less automobiles on public roads and only couple of month later became the world's first company to showcase a fully autonomous, all-electric, purpose-built vehicle that capable of driving up to 120 km/h.

Figure 2.11: ZOOX autonomous bus during testing in California

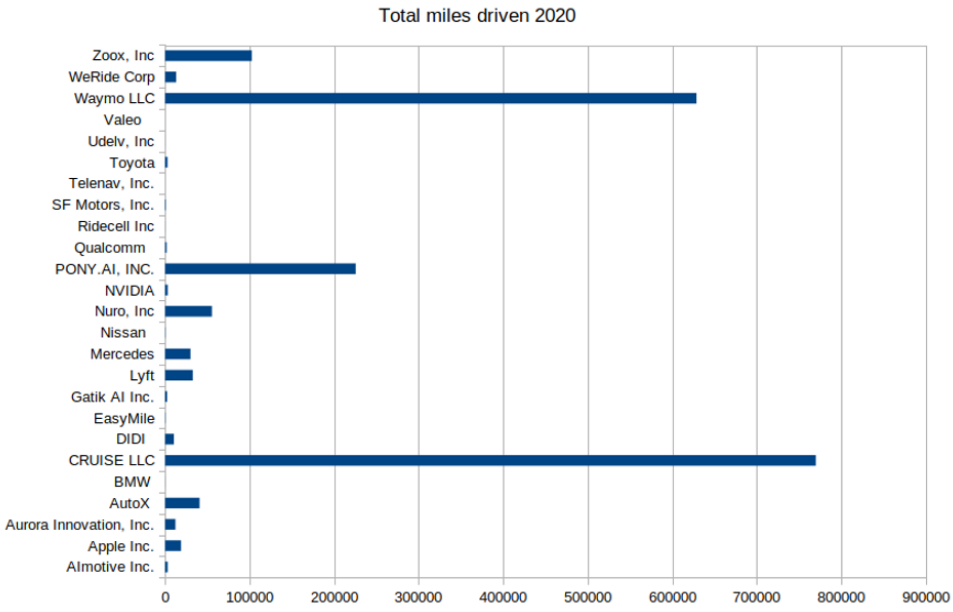


Figure 2.12: Miles driven by AV research vehicles in 2020

2.4 AUTONOMOUS VEHICLES COMPARISON

As we have seen, in the last years the market of autonomous vehicles has grown substantially thanks to different big companies' investments and startups innovative ideas. The projects mentioned above are only a short list of the most famous and interesting in the field for their approach to the problem and the adopted solutions.

It is early to choose a winner or to decide which system performs better since most of the data are not public and the little public information we have, is referred to beta version of the software still under development but what is clear is that there are two main lines of thought on how to approach the autonomous driving problem, the one that uses LiDAR as a main sensor pared with cameras and radars and the one based mainly on cameras and radars but with a smaller LiDAR or completely without it. What is clear from this comparison and from the data in Table 2.1, all the current prototypes are yet too expensive and complex for average people since the cheaper system like the Tesla FSD costs as much as a common utility car only for the autonomous optional. All the other systems, except for CommaAi solution that however, does not grant full autonomy, cost more than a luxury car and are extremely complex.

Autonomous Vehicles				
Developer	Km driven	Sensors Suite	Autonomy Level	Cost (€)
Tesla AP 2.0	4 Billion	7 Cameras, 12 Radars	Level 2	8k+vehicle
Tesla FSD Beta	–	7 Cameras, 12 Radars	Level 3	10k+vehicle
Comma Two	25 Million	1 Camera + vehicle sensors	Level 2	1k+vehicle
WAYMO	40 Million	6 Cameras,3 Radars, 4 LiDAR	Level 4	Estimate 150k
NuTonomy	–	6 Cameras,5 Radars, 1 LiDAR	Level 4	Estimate 150k
Lyft	0.05 Million	6 Cameras,5 Radars, 1 LiDAR	Level 4	Estimate 150k
Baidu	1 Million	10 Cameras,2 Radars, 5 LiDAR	Level 4	Estimate 150k
UBER/Aurora	–	8 Cameras,5 Radars, 5 LiDAR	Level 4	Estimate 200k
ZOOX/Amazon	0.4 Million	14 Cameras,10 Radars, 8 LiDAR	Level 5	Estimate 200k

Table 2.1: Comparison between AV systems

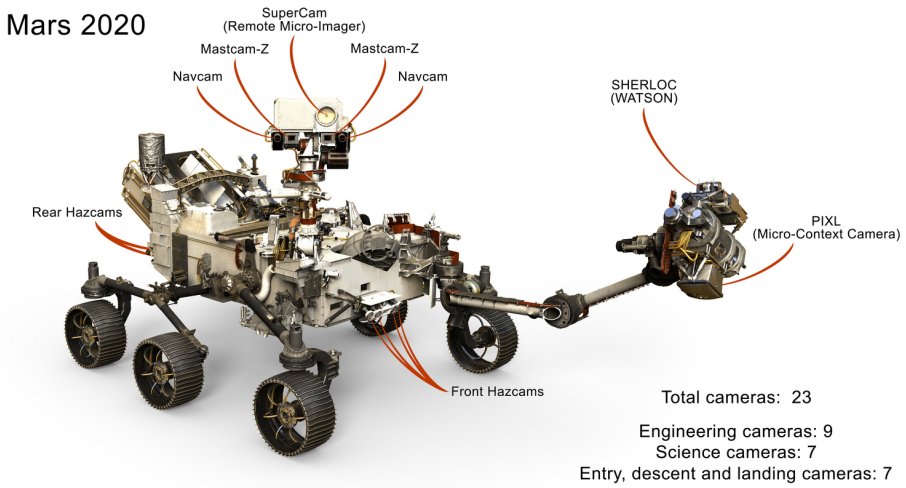


Figure 2.13: NASA Perseverance rover sensors

2.5 OTHER AUTONOMOUS VEHICLES

When we think to autonomous vehicles, we immediately think at cars but those are not the only autonomous system currently in use or in development. Many are the examples of autonomous systems around us. From autonomous lawnmowers to vacuum cleaning robots, from autonomous underground trains to aerial drones.

One interesting example is Perseverance rover built by NASA [10] landed on Mars the 18th of February 2021. It is the first fully autonomous human built vehicle to drive without human intervention on another planet. Thanks to its advanced suite of sensors and software the robot can avoid obstacles, drive trough rough terrains and reach the given destination safely.

In all those systems the perception and scene understanding pipeline is critical and the technology developed for cars can be easily transferred to all autonomous robotics systems and also to other road vehicles like buses trucks and motorcycles.

2.6 HARDWARE AND SOFTWARE

In this section we present all the components required to enable full self-driving capabilities to a vehicle. To make a vehicle autonomous in fact we need to install additional hardware and custom software that is meant to substitute completely a human driver.

We can proceed analyzing those components comparing them to the human driver counterpart since it exists a certain similarity between the two.
Concerning the hardware, we can identify 3 main subsystems:

- **Perception system:** it substitutes five human senses. It is composed by exteroceptive sensors that monitor the environment around the vehicle (other cars, pedestrian, obstacles etc.) and proprioceptive sensors that monitor the state of the vehicle by measuring all its internal parameters (speed, acceleration, position etc.).

- **Actuators:** they are the equivalent of human muscles. A series of electrical and mechanical actuators capable of controlling the throttle, the steering and if present the gearbox. We consider in this category also all the relays necessary to control all the vehicles lights.

- **Computing unit:** it can be considered as the driver brain of the vehicle. It is a powerful and efficient computing unit, usually custom made specifically for autonomous vehicle workloads and is composed by different CPU and an accelerator for Image processing and Machine learning computations.

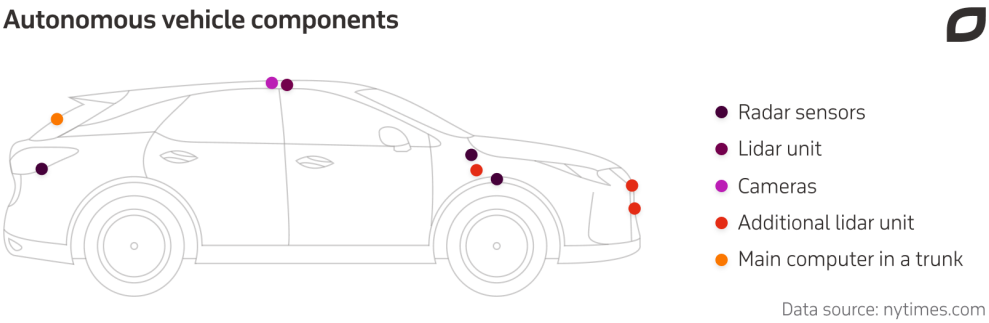


Figure 2.14: Autonomous vehicle common sensors location

2.7 HARDWARE: SENSORS

Given that this work is focused more on the perception part we present here all the sensors currently in use in the best performing autonomous vehicles. Even if not all the manufacturers agree on the necessity of using all those sensors combined, most of them integrate multiple redundant sensing elements to guarantee more robustness and accuracy to the system. As we have seen from previous analysis of the AV prototypes commonly used sensors are cameras, radars, LiDARs and GPS. Sensors are placed all around the vehicle and usually are placed in strategic spots to cover all the vehicle surrounding. In Figure 2.14 is shown the most common sensors setup for an autonomous car. Sensors constitute a big percentage of the cost of the autonomous system.

2.7.1 *Camera*

Camera is the most common type of sensor and has the function of seeing the objects in the road scene just like human drivers do with their eyes. This type of sensor is capable of converting light energy into an electrical quantity that can be elaborated and used by digital systems like computers.

By equipping cars with cameras at every angle, the vehicles can maintain a 360° view of the external environment, thereby providing a broader picture of the traffic conditions around it. The cost of the sensor is low thanks to the consumer electronic integration, it is widely available with different technical specification and with its small form factor it is easy to integrate and hide inside the vehicle body. Moreover, camera as a sensor is well understood and many, well performing algorithm are already available to work with image input.

On the market are available different families of cameras based on the same principles but with different outputs and with different strengths and weaknesses.

Unfortunately, camera sensors are still far from perfect and require the necessity of at least another type of sensor to guarantee the coverage of all the driving situation and the robustness of the final system required for this kind of applications. Poor weather conditions such as rain, fog, or snow can prevent cameras from clearly seeing the obstacles in the roadway, which can increase the likelihood of accidents. Similar problems affect human eyes but thanks to the greater capability in terms of high contrast of eyes and situation assessment the human can prevent and intervene in advance applying a filter like sunglasses or by stopping for couple of minutes if the visibility conditions are too bad. Additionally, there are often situations where the images from the cameras simply are not good enough for a computer to make a good decision about what the car should do. For example, in situations where the colors of objects are very similar to the background or the contrast between them is low, the driving algorithms can fail. Moreover, since cameras works with light if the light is absent like in a scenario of total darkness during the night in a road without illumination the presence of any non-illuminated and unpredictable obstacle in the road area is hard to detect, the same happens in the opposite situation when there is too much light due to direct illumination of the sun during sunrise or sunset. In those cases, the output of the camera tends to degrade and cause problems of blindness. Cameras can be also afflicted by dirt, rain, insects or any other impurity present on the optic that can deform or partially occlude the optic and make the data unusable. Despite

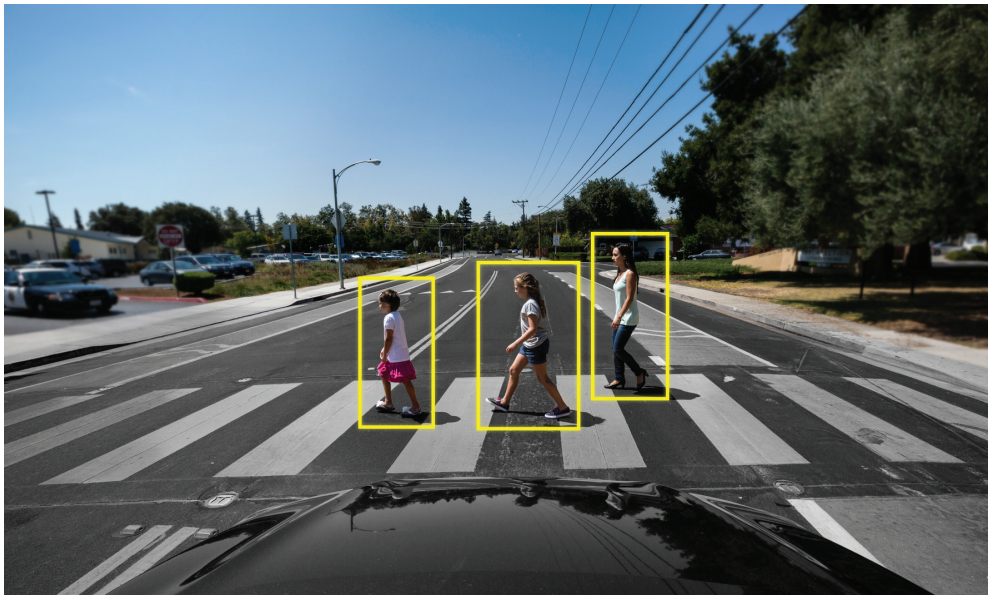


Figure 2.15: Image segmentation of pedestrians crossing the road from a monocular camera

those weaknesses cameras are still broadly used in autonomous vehicle since for every problem has been found solution capable of overtake them.

Let us analyze now different cameras and how they can be used in AV.

2.7.2 *Monocular camera*

Monocular camera is composed by one sensing element sensible to light, a filter that allows to make the camera sensible to visible light or to infrared spectrum and a single optic capable of guiding the light to the sensor. Monocular cameras used in automotive industry have often high resolution and high sensitivity in order to capture as much information as possible, with a high refresh rate (30-60Hz) and a wide field of view (FOV) to reduce blind spots. The sensor output is a single video stream or a sequence of images. This kind of sensors are used in all the systems where the special information is provided by another sensor like a LiDAR or radar or to cover blind spots like to the vehicle sides and the rear. Thanks to the fact that are very cheap many times many cameras are installed on the vehicle and used also for a posterior analysis and performance evaluation.

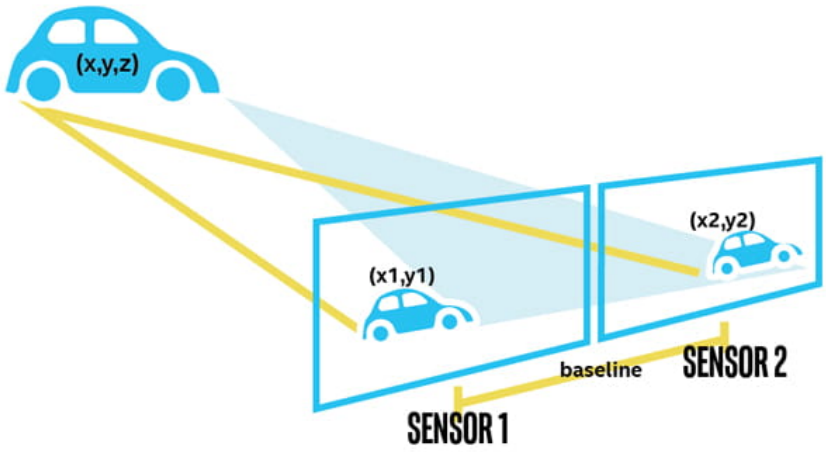


Figure 2.16: Stereo camera working principle

2.7.3 *Stereo camera*

The stereo camera consists of two high-resolution mono-cameras, housed approximately 20 centimeters apart and usually installed in the front part of the vehicle behind the windshield. This camera configuration is inspired by human anatomy in fact the distance between the cameras and their location in the sensor is similar to the distance that human has between eyes that is also the minimum distance that allows the camera to measure distances thanks to the difference in perspective between left and right optical paths. The fundamental strength of the stereo camera is its ability to compare the two optical paths and compute the depth at which each object lays. Moreover, the redundancy is a plus also in poor visibility conditions: it preserves its high-resolution capability even under cult circumstances (where other technologies for object recognition could be limited); for example, when several objects are next to each other, when objects are partially obscured or when there is poor contrast between the object and its background [11].

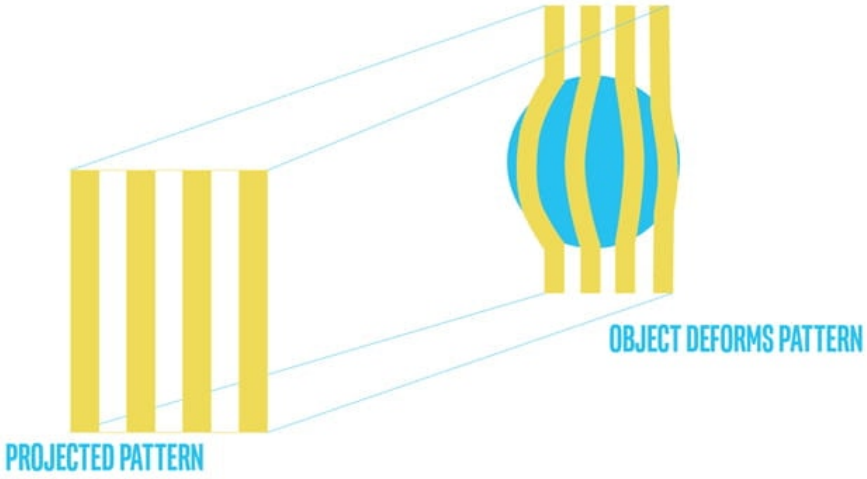


Figure 2.17: Depth camera working principle

2.7.4 *Depth camera*

Structured light and coded light depth cameras are not identical but based on similar technologies. They rely on projecting light, usually infrared light, onto the scene. The projected light is patterned, either visually or over time, or some combination of the two. Because the projected pattern is known, how the sensor in the camera sees the pattern in the scene provides the depth information. For example, if the pattern is a series of stripes projected onto a ball, the stripes would deform and bend around the surface of the ball in a specific way.

If the ball moves closer to the emitter, the pattern will change too. Using the disparity between an expected image and the actual image viewed by the camera, distance from the camera can be calculated for every pixel resulting in a 3D representation of the visible region.

Figure 2.18: Comparison between regular RGB camera and event base camera output

2.7.5 *Event based camera*

Those cameras are the latest addition to sensing suite for the AV since they are new in terms of technology. As the name implies event-based cameras are cameras triggered by events, by changes in the framed area. This technology allows to cut down on output bandwidth by sending images in from of difference between frames. Basically, the output of the camera is a series of updates over the previous frame that leads to the new one. This approach is interesting since the amount of data that moves in an AV is very high and this type of cameras can either reduce cost of the vehicle network or allow for higher resolution sensors to be implemented without additional cables and without increasing the load on the main computing unit. Since it is a fresher technology the sensor is still not fully understood and used in commercial vehicles, but it is promising. Thanks to their working principle, those cameras have an higher contrast ratio in situation of low light or high contrast but they still lack of RGB colors and are less intuitive to interpret at a first sight.

Camera technology comparison				
Technology	Mono cam	Stereo cam	Depth cam	Event based cam
Manufacturing	Single optic, single sensor	2 mono cam mounted 20 cm apart	1 mono cam and a light source	Monocular optic with custom readout and sensing element
Output data	2D video stream	3D image	3D image	2D frame difference
Resolution	720/1080/4K	720/1080/4K	720/1080	360/720
Refresh Rate (Hz)	30/60/120	60/90	10/20	Up to 1M
Data flow	Medium	High	High	Low
Cost (€)	50	100	3k	3k-6k
Pros	Well Understood	Depth info	Depth info	Compressed data
Cons	No depth info	High data flow	High data flow, high cost	No Depth, New tech

Table 2.2: Comparison between camera technologies

2.7.6 *LiDAR*

LiDAR stands for Laser imaging Detection and Ranging. As the acronym implies it is a method for measuring distances (ranging) by illuminating the target with laser beam and measuring the reflection with a readout sensor, a similar approach to radar systems, with the only difference being that they use lasers instead of radio waves. Differences in laser return times and wavelengths can then be used to make digital 3D representations of the target. It has terrestrial, airborne, and mobile applications. LiDAR uses ultraviolet, visible, or near infrared light to image objects. It can target a wide range of materials, including non-metallic objects, rocks, rain, chemical compounds, aerosols, clouds and even single molecules. A narrow laser beam can map physical features with very high resolutions; for example, an aircraft can map terrain at 30 cm resolution or better. LiDAR uses active sensors that supply their own illumination source. The energy source hits objects, and the reflected energy is detected and measured by sensors. Distance to the object is determined by recording the time between transmitted and back scattered pulses and by using the speed of light to calculate the distance traveled.
Autonomous vehicles use LiDAR for obstacle detection and avoidance to navigate safely through complex environments. Point cloud output from the LiDAR sensor provides the necessary data for robot software to determine where potential obstacles exist in the environment and where the robot is in relation to those potential dangers. Usually this type of sensors are mounted on top of the vehicles providing a 360° view of the surrounding. Examples of companies that produce LiDAR sensors commonly used in vehicle automation are Ouster and Velodyne.

Since rare earth metals are needed in order to produce LiDAR sensors, they are much more expensive than radar sensors used in autonomous vehicles. The systems required for autonomous driving can cost well beyond $10,000, while the top sensor being used by Google and Uber costs up to $80,000. Recently prices dropped significantly with the introduction of lower resolution sensors that nevertheless are still suitable for autonomous vehicle applications. In the Table 2.3 are presented the most adopted sensors with their specifications and costs. Yet another problem is that snow or fog can sometimes block LiDAR sensors and negatively affect their ability to detect objects in the road.

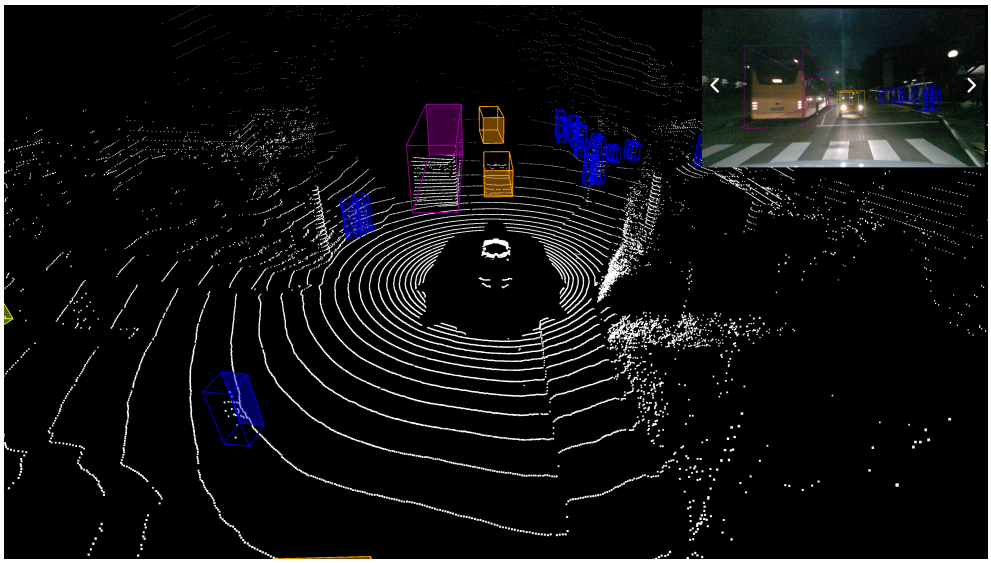


Figure 2.19: LiDAR data from NuScenes dataset with annotated objects

LiDAR sensors on market						
Model	Manufacturer	Layers	Range	Vertical Scope	Layer pitch	Price(€)
HDL64E	Velodyne	64	120m	2°/-24.9°	0.4°	75K
HDL32E	Velodyne	32	80m	10°/-30°	1.33°	30K
VLP16	Velodyne	16	100m	10°/-10°	1.33°	8K
VLS128	Velodyne	128	245m	15°/-25°	0.11	100K
OS0-32	Ouster	32	50m	45°/-45°	0.7°-5.5°	7K
OS0-64	Ouster	64	50m	45°/-45°	0.7°-5.5°	11K
OS0-128	Ouster	128	50m	45°/-45°	0.7°	14K
OS1-16	Ouster	16	120m	22.5°/-22.5°	0.35°-2.8°	3.5K
OS1-32	Ouster	32	120m	22.5°/-22.5°	0.35°-2.8°	8K
OS1-64	Ouster	64	120m	22.5°/-22.5°	0.35°-2.8°	10K
OS1-128	Ouster	128	120m	22.5°/-22.5°	0.35°	15K
OS2-32	Ouster	32	240m	11.25°/-11.25	0.18°-0.73°	20K
OS2-64	Ouster	64	240m	11.25°/-11.25	0.18°-0.73°	23k
OS2-128	Ouster	128	240m	11.25°/-11.25	0.18°	25K

Table 2.3: Comparison between LiDAR sensors on the market

2.7.7 *Radar*

RADAR stands for Radio Detection and Ranging System. It is an electromagnetic system used to detect the location and distance of a headway object from the point where the RADAR is placed, quickly providing its velocity, range and angle information. It works by radiating energy (radio waves) into space and monitoring the echo or rejected signal to detect and track various objects. Normally radars are directional and provide information of range (from pulse delay), velocity (from Doppler frequency shift) and angular direction/angle. From further elaboration a radar can also output the target size (from magnitude of return), shape and components (return as a function of direction); from the modulation of the return can detect moving parts and material composition.

The complexity (cost and size) of the radar increases with the extent of the functions that the radar performs. The simplest function of radar is the range estimation: the device emits a concentrated radio wave and listens for any echo, if there is an object in the path of the radio wave, it will reject some of the electromagnetic energy, and the wave will bounce back to the radar. Radio waves move through the air at a constant speed (the speed of light), so, based on how long it takes to the radio signal to return, the instrument can calculate how far away the object is.

Automotive radars have been the backbone for active safety and advanced driver assistance systems for decades. A typical automotive radar is mounted behind the front grille of a vehicle at a height of less than one meter, where it can monitor the road ahead and the adjacent lanes ahead of the vehicle.

This sensor can be employed in different types of driver assistance systems depending on its detection range: long range (LRR) for Adaptive Cruise Control, medium range (MRR) for cross traffic alert and lane change assistance, shortrange (SRR) for parking aid and obstacle/pedestrian detection. Range is one of the device's characteristics, together with the frequency, that distinguishes peculiar radar typologies and usages.

Despite the main disadvantage of having a low angle resolution, the numerous functionalities and advantages make the radar one of the most important sensors. It is computationally lighter than a camera and produces far less data than a LiDAR. Its main strengths are the greater robustness to weather or road conditions, the ability to measure the Doppler effect and its reliability. Radar is a proven technology increasingly becoming more efficient for the autonomous car. Modern self-driving prototypes rely on radar and LiDAR to "cross validate" what they're seeing and to predict motion or depend on the complementary action of radar and cameras.

Radar sensors are good but not perfect, the pedestrian recognition algorithm definitely needs a lot of improvement, seeing as the automotive radar sensors used in today's vehicles only correctly identify between 90% and 95% of

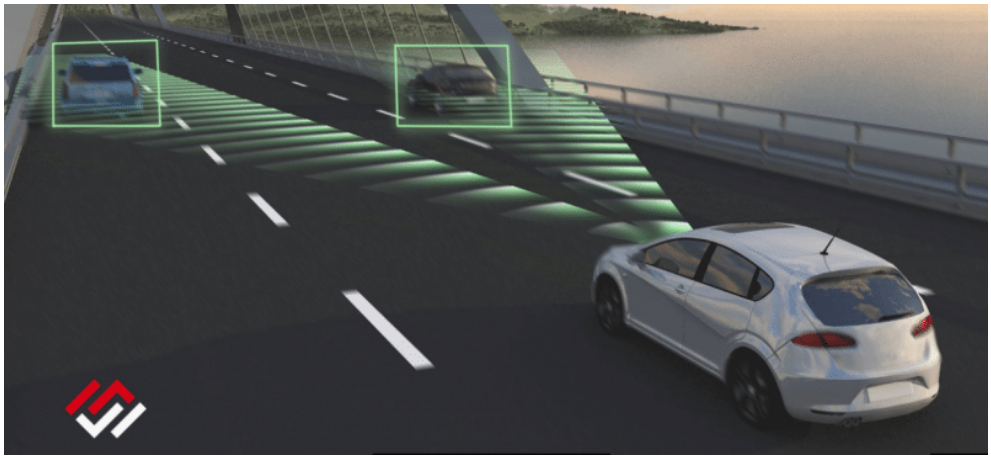


Figure 2.20: Radar working principle

pedestrians, which is hardly enough to ensure safety on the road. As well, the still widely-used 2D radars are not able to determine accurately an object's height, as the sensors only scan horizontally, which can cause a variety of problems when driving under bridges or road signs. A wider variety of 3D radar sensors are currently being developed to solve these issues.

2.7.8 *Global Positioning System*

Global Positioning System (GPS) is a sensor which uses real time geographical data received from several GPS satellites to calculate longitude, latitude, speed, and of course to help navigate a car. The GPS project was started by the U.S. Department of Defense in 1973, with the first prototype spacecraft launched in 1978 and the full constellation of 24 satellites operational in 1993. Originally limited to use by the United States military, civilian use was allowed from the 1980s following an executive order from President Ronald Reagan. In recent years similar systems has been developed by other countries like GLONASS by Russian government, Galileo by Europe, and BeiDou by China. All the systems combined with the newest GPS satellite generation increased a lot the precision, availability, and accuracy of the system. Today it can guarantee an accuracy of 0.05m making this sensor particularly useful for global localization purposes for autonomous vehicle [9]. In fact, if all the sensors presented above give to the vehicle a local scene understanding and localization, GPS gives a global absolute position. This information can be used both for autonomous navigation but also for incorporating the full map of the area the vehicle is driving through in the autonomous driving software pipeline allowing for predictive action like slowing down before a road intersection even if not in the full view of the sensors or being aware of tight road turn in advance.

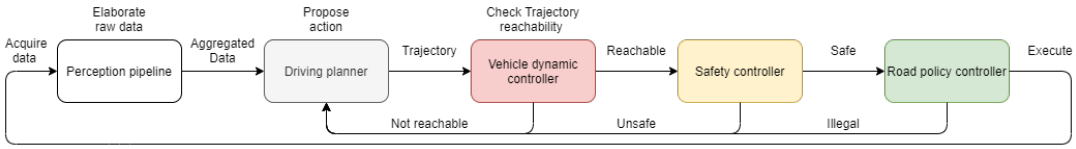


Figure 2.21: Autonomous vehicle software pipeline

2.8 SOFTWARE

The most important part of an autonomous driving system is for sure the software since it is the true intelligence of the vehicle. It is the equivalent of the intelligence and experience of the human driver and it has the main function of elaborating the sensor data and extract useful knowledge about the scene, based on this knowledge it plans actions that bring the vehicle from the starting point to the goal. The software is also the key difference between all the state-of-the-art system previously discussed since the sensor suite is quite similar with some small differences concerning primary and secondary sensors and their resolution and quality.

Every company organizes the software in its own way but at a high level depending on the function that are performed we can identify:

- **Perception pipeline:** has the task of acquiring and processing low level data from the sensors extracting aggregated and labelled high level data.
- **Driving planner:** based on sensor input should propose a possible solution to the current scene by planning a trajectory, choosing a speed of execution and a timing.
- **Vehicle dynamic controller:** checks if the proposed solution can be performed by the car in the times and speed given the physical quantities of the vehicle (power, dimensions, maximum steering angle, etc...). It can propose changes to the kinematic values to make the proposed solution feasible or reject it if the solution is unfeasible.
- **Safety controller:** verifies that the proposed solution by the Driving planner is safe for all the external road agents and for the car to be executed. It can reject or approve the maneuver depending on the actual risk involved.
- **Road policy controller:** the software that takes care of road rules and verifies that the motion proposed complies with all the local rules of the road and with all the road signs present.

All the last 3 software components are called controller since they have to monitor and decide if the planned trajectory can be achieved in a safe way and without breaking rules. If all those 3 components approve the maneuver the car will execute it otherwise the solution is discarded, and the driving planner should propose another solution.

2.8.1 *Software perception pipeline architecture*

Let us focus on the perception pipeline that is our main research focus and analyze the different possibilities available. Currently there are two main line of thoughts used by autonomous vehicles manufacturers to handle the complexity of the perception pipeline.
Those are:

- **Sensor independent pipeline**
- **Sensor fusion pipeline**

Furthermore, depending on where the sensor data are processed and transformed, we can have:

- **Centralized computing unit**
- **Partially Decentralized computing unit**
- **Distributed computing unit**

Any other mixing configuration called hybrid approaches obtained by applying partially one solution and partially another will be omitted here for brevity but exist and are used. What those mixed configurations do is trying to leverage the pros of each configuration mitigating the cons.

2.8.2 *Sensor independent pipeline*

In this approach each sensor is considered independent from all the other and its data are handled separately and considered as the only data available. Raw data recorded by the sensor are sent to the processing unit that can be either centralized in the central processing unit or decentralized and situated inside the sensor module. In the decentralized approach the sensor is called smart since it contains part of the intelligence for handling and processing data. The inverse sensor model (ISM), the function that gives a meaning to the sensor data recordings is applied and the processed data are then sent to the actual perception algorithm that will infer meaning and extract useful information to be used by Driving planner. The advantages of an independent pipeline are the fact that the system is more robust to sensor fails since a sensor is independent from the other and gives its view of the surrounding world. Moreover, with independent pipelines each inverse sensor model is custom and can easily remove specific sensor noises and improve data outputs. With a partially decentralized solution the central unit is freed from the weight of pre-processing computation leaving more power to the main control algorithms or leading to a cheaper and more efficient central computing unit. In a fully decentralized solution instead all sensors process their own data and share the final aggregated information with the central processing unit. With a consensus algorithm the final best world representation is chosen by voting the most probable one. In this case the central unit is completely freed up by the data processing task, each sensor contains a pre-processing unit and a processing unit dedicated to the perception specific for each sensor type.

2.8.3 *Sensor Fusion*

In this approach the sensor raw data are mixed with little to no pre-processing in order to preserve as much information as possible in a centralized way or after the inverse sensor model computation in a centralized or partially decentralized manner to increase the certainty of the output and maximize it. Since the idea of fusion requires a unique centralized node to perform data merge while the distributed solution reckons on separated computing units the two architecture are not compatible. The additional complexity introduced in the system is not justified by tangible benefits [61]. The main advantages of sensor fusion solution is the ability of capturing as much information as possible and combining them increasing the certainty of the outcome. The main drawbacks are the fact that the system is less resilient to single or multiple sensor failure and the fact that in raw data fusion only similar data type can be merged like radar, LiDAR and depth camera outputs or RGB and IR images and in this case after the fusion data are treated independent from

Sensors pipeline		
Computation	Independent	Fusion
Centralized	V	V
Partially Decentralized	V	V
Distributed	V	X

Table 2.4: Feasibility of different architectural combinations

the sensor that has generated them and so in a more general way. In the case of fusion of pre-processed data, the data aggregation is done on high level information.

2.9 OBJECT DETECTION METHODS

Given that the main purpose of this research work is the development of a robust ad accurate object detection pipeline based on low resolution LiDAR sensor we present here state-of-the-art approaches currently in use for different road vehicle prototypes. This task is sometimes also referred to as "general obstacle detection", "free space estimation" or "occupancy grid mapping" and ensures that the vehicle detects obstacles such as pedestrians, cyclists, and other vehicles [14].
This problem consists in predicting oriented 3d bounding boxes, represented in the LiDAR coordinate frame, corresponding to target actors in the scenes given a point cloud obtained by a LiDAR scan. The algorithm should provide in output the box dimensions (width, length and height), the coordinates of the center of the box (x, y, z) with respect to the LiDAR coordinate frame and the heading angle ϕ of the box measured in the body coordinate frame which is a translated version of the LiDAR coordinate frame to the center of the box.
This problem has been approached in the past years following two main lines of thought, using white box approach based on occupancy grid modelling or using a black box approach based on Machine learning techniques.

2.9.1 *White box approaches*

White box approach is based on Occupancy Grid (OG) which is well-known Robotics method of representing the environment.
In this methodology the space around the vehicle, scanned by the vehicle sensors, is represented as a grid where each grid cell contains the sensor output in that location. Once the sensor output is converted in occupancy grid format a clustering algorithm can be applied to classify, based on relevant features like length, width and shape, each cell of the grid. In literature we find many variations of the binary 2D occupancy grid [15] approach in which the grid is a plane, and each cell contains a 1 if the spot is occupied and a 0 if it is free, that are used for object detection and tracking in the field of AV.
The first variation is the probabilistic occupancy grid in which each cell contains a number between 0 and 1 depending on the confidence that the cell is occupied given one or multiple sensor measures and sensors accuracy [16]. This approach is particularly used when there are multiple sensors output available like a LiDAR and a Radar or a depth camera that can be fused together since each sensor view increases the confidence of the presence or absence of obstacles.
There are variations of the 2D occupancy grid that stores in each cell the number of points present in the vertical column above the cell itself or the maximum height of the sensor output in each spot, in this case we call them

2.5D occupancy grid [18].
Finally, we have 3D Occupancy Grid also called Voxel Grid [17] that represents the vehicle surrounding space using Voxels (Volumetric pixels), 3d volumetric entities equivalent to pixels in 2D space, that can contain either a binary or a probabilistic value depending on the approach chosen and represents in 3d obstacles around the vehicle. This last approach is less used since it is heavy both in computational and memory terms and is considered mainly for off road driving where a 3D view of the vehicle surrounding can come in handy.

2.9.2 *Black box approaches*

In the recent years, with the advent of many new techniques in machine learning different interesting solutions based on a black box approach for object detection and classification of 3D LiDAR point cloud data have been developed. Those methods are basically classified under two categories, the first one involves conventional heuristic approaches such as model fitting by employing iterative approaches [19] or histogram computation after projecting LiDAR point clouds to 2D space [20]. In contrast, the second category investigates advanced deep learning approaches [21][22][23] which achieved significant improvements in performance in the last years. These approaches in the latter class differ from each other not only in terms of network architecture but also in the way the LiDAR data are represented before being fed to the network. Regarding the network architecture, high performance segmentation methods use fully convolutional networks [24], encoder-decoder structures [25], or multi-branch models [29].
In the context of 3D LiDAR point cloud representation, there exist three popular methods: voxel creation [27][25][34][36], point-wise operation [28] and image projection [23][21][33]. Voxel representation transforms a point cloud into a high-dimensional volumetric form, i.e. 3D voxel grid [27][28][25]. In [37], point cloud data are converted into voxels containing feature vectors, and then a convolution-like voting-based algorithm is used for detection. Due to sparsity in point clouds, the voxel grid, however, may have empty voxels which leads to redundant computations; to solve this issue in [38] sparsity is exploited by using a feature-centric voting scheme to implement novel convolutions, thus increasing the computation speed. These methods use hand-crafted features, and even if they lead to acceptable results on specific datasets, they are not suitable for complex scenes like the one of autonomous driving. With a different approach, in [28][39] it is described a system based on CNN architecture that could learn point wise features directly from point clouds. These methods directly process point cloud data to perform 1D convolution on k-neighborhood points, but they cannot be applied to a large number of points; thus, image detection results are needed to filter the original data points and propose regions of interest. VoxelNet

presented in [27] groups point cloud data into voxels, extracts voxel wise features, and then converts these features into a dense tensor to be processed using 3D and 2D convolutional networks. At the core of this solution, we find a region proposal networks [40] that have become an important building block of top-performing object detection frameworks [41][42][43]. VoxelNet propose a variation to the RPN architecture proposed in [40] and combine it with the feature learning network and convolutional middle layers to form an end-to-end trainable pipeline.

The input to the RPN is the feature map provided by the convolutional middle layers. The network has three blocks of fully convolutional layers. The first layer of each block down samples the feature map by half via a convolution with a stride size of 2, followed by a sequence of convolutions of stride 1. After each convolution layer, BN and ReLU operations are applied. We then up sample the output of every block to a fixed size and concatenate to construct the high-resolution feature map. Finally, this feature map is mapped to the desired learning targets: a probability score map and a regression map. The major problem with this method is the high computational cost of 3D CNNs that grows cubically with the voxel resolution. PointCNN [44] and spatially sparse convolution [45] use two similar approaches that increase the 3D convolution speed. A great advantage of those approaches is that no information is loss due to dimensionality reduction or compression in pre-processing and in computation phase.
Point-wise methods [28] instead process points directly without converting them into any other form. The main drawback here is the processing capacity which cannot efficiently handle large LiDAR point sets unless fusing them with additional cues from other sensory data, such as camera images as shown in [35]. To handle the sparsity in LiDAR point clouds, various image space projections, such as Bird-Eye-View (BEV) (i.e. top view) [30][31][32] and Spherical-Front-View (SFV) (i.e. panoramic view) [23][21][33] have been introduced. MV3D [43] is the first method to convert point cloud data into a BEV representation. In this method, point cloud data are converted into several slices to obtain height maps, and these height maps are then concatenated with the intensity map and density map to obtain multi-channel features. ComplexYOLO [31] uses a YOLO (You Only Look Once) [46] network and a complex angle encoding approach to increase speed and orientation performance, but it uses fixed heights and z-locations in the predicted 3D bounding boxes. A key problem with all of these approaches, however, is that many data points are dropped when generating a BEV map, resulting in a considerable loss of information on the vertical axis. This information loss severely impacts the performance of these methods in 3D bounding box regression.

2.10 MULTI OBJECT TRACKING

The second part of our research work is focused on Multi Object Tracking (MOT). This problem consists in finding a temporal association between entities in different consecutive scenes. In our case we want to keep track of each vehicle and pedestrian present in the view of the system and compute its speed, trajectory and, when necessary, fill missing bounding boxes not segmented by the network and correct misclassified objects.
This problem can be split into two sub-problems, the first one is called data association and tackle how bounding boxes are matched between different frames. The second one is a prediction problem, centered on finding missing bounding boxes and correct misclassified ones.
Bayesian filters are widespread in literature: these filters exploit the Chapman-Kolmogorov theorem through the system transition density to achieve predicted probability density functions (PDF) [49] for the objects under consideration. Measurements are then used to update the predicted PDF to find the posterior PDF, from which the estimates can be obtained. Prediction and update steps in Bayesian filtering involve complicated integrals that lead to a high computational cost. Every time the system to be modelled is linear and the noise follows a Gaussian distribution the integrals can be computed analytically and provides the optimal solution; we call it Kalman filter [47][48]. However, if the system behavior is nonlinear, Extended Kalman Filter (EKF) [50][51] and Unscented Kalman Filter (UKF) [54][55] are favored solutions. When non-linearities become huge, EKF provides less accurate solutions due to the first-order linearization of the system's equations through Taylor-series expansion. Conversely, UKF is based on the so-called unscented transformation, which approximately provides Gaussian distributed outputs even when dealing with nonlinear transformations. A last category of filters is particle filters or Sequential Monte Carlo (SMC) [52] that are other variants of Bayesian Filters that can be used for nonlinear systems and non-Gaussian Noise Distributions. As the name implies, they use weighted particles, each represented by possible state estimation and posterior distribution.
The usage of Random Finite Set (RFS) [53] statistics is common in Multi-Object-Tracking (MOT). In particular, RFS enables MOT without a priori measurement association through the implementation of recursive Bayes filtering. When dealing with scenarios in which the birth and death of objects are regular, with a significant amount of clutter and false positives, the association process provided by traditional Bayes filters leads to erroneous results. Conversely, RFS allows accounting for objects birth (regular or spawning), occlusions, misdetections, and disappearances by taking the number of objects under consideration as a stochastic variable. Gaussian Mean-Probability Hypotheses Density (GM-PHD) Filter [56], Multi-Bernoulli Mixture (MBM)

Filter, Poisson Multi-Bernoulli Mixture (PMBM) Filter, etc. are other filters adopted in the literature.

2.11 DATASETS

During the years have been collected different databases of drive rounds both in urban and extra urban areas from different companies and research groups around the world. Many of them are nowadays freely available online for research purposes. Here we propose a selection of the currently most used dataset available online free of charge that we have evaluated for the training of our model. To train the best segmentation model, the dataset is fundamental since it contains basic information that the network implicitly learn. A good dataset makes the difference between good and bad performance of the network output. All the dataset considered must have all the sensor data available (Cameras, Radar and LiDAR) and should be still actively maintained and used. In the evaluation we gave particular importance to LiDAR data and to the quality of class annotation for vehicles, pedestrian and other road obstacles since a good machine learning model starts with a good training dataset.

2.11.1 *KITTI*

KITTI is the oldest, most famous and used dataset in autonomous driving field. It has been developed by the Karlsruhe Institute of Technology in 2012 [13]. It contains different cameras and LiDAR data recordings, and it is in constant grow since the first release. All the data are organized by scenarios (urban driving, highways etc.) and are manually annotated. Accurate ground truth is provided by a 64 layers Velodyne laser scanner and a GPS localization system. The dataset is not extremely heavy and big but covers most of the scenarios of driving from urban areas to highways. The major drawbacks are the absence of the radar data and for the thesis purposes the inconsistent quality of LiDAR annotation that are present but not extremely accurate and sometimes are missing. Moreover, the LiDAR data are from a high end 64 plane with 4 time the resolution of the final sensor we want to use.

2.11.2 *A2D2*

This dataset developed by a research group from Audi is very heavy not only because contains a lot of scenes but also due to the uncommon sensor setup. The testing vehicle in fact has been equipped with 5 16 layers LiDAR and multiple cameras and this caused the explosion of the dimension of the dataset [12]. Moreover, the uncommon sensor positioning, and the cost of so many LiDAR makes this dataset not so appealing for developing purposes. The unique point in its favour is the fact that is the only publicly available

dataset, beyond KITTI, recorded entirely in Europe, in Germany to be exact. It contains mainly scenes of urban, densely populated areas.

2.11.3 *Waymo*

This dataset from Google is the most complete and big in terms of scenes and overall dimension, it contains recordings of all the sensors except the Radar. All sensors data are high quality and well annotated. The only drawback is that the main LiDAR sensors is a high end 64 layers LiDAR that is extremely expensive and far away from our target of low-cost LiDAR sensor. The data format is custom but well documented and easy to use.

2.11.4 *Lyft/NuScenes*

The NuScenes dataset is inspired by the pioneering KITTI dataset. NuScenes is the first large-scale dataset to provide data from the entire sensor suite of an autonomous vehicle (6 cameras, 1 LIDAR, 5 RADAR, GPS, IMU). Compared to KITTI, NuScenes includes 7x more object annotations. This dataset has been recorded during 2019 in the cities of Boston and Singapore, two cities that are known for their dense traffic and highly challenging driving situations. The scenes of 20 second length are manually selected to show a diverse and interesting set of driving maneuvers, traffic situations and unexpected behaviors. Even though Lyft and NuScenes datasets are different in terms of data and test vehicles they share the same data structure and sensor suite thanks to a partnership between the two companies. This makes the dataset particularly appealing since is not only big and complete with radar and good LiDAR annotations, even if they are marked automatically by an external annotation tool made by SCALE that maps the image annotation and builds a 3D bounding box for the point cloud but is available with different vehicle setup. The data format is easy to understand, well organized and easy to replicate. Moreover since 2020 NuScenes developed a specific data collection with 3D annotation especially built for LiDAR segmentation development. Last but not least this dataset is recorded using ROS and then converted in the final data format that makes even easier the data conversion from our custom dataset to the final target. The full dataset includes approximately 1.4M camera images, 390k LiDAR sweeps, 1.4M RADAR sweeps and 1.4M object bounding boxes in 40k key frames.

Dataset Comparison					
Dataset	Camera	Radar	LiDAR	Size	Notes
KITTI	2 Grayscale, 2 Color	X	1 Velodyne HDL 64E	30Gb	Quite old. Lidar label from images
Lyft Perception Dataset	6 Color	V	1 Velodyne HDL32E	120Gb	NuScenes format
NuScenes Dataset	6 Color	V	1 Velodyne HDL32E	550Gb	
A2D2 Dataset	6 Color	X	5 Velodyne VLP 16	2.3Tb	Uncommon setup
Waymo Open Dataset	5 Color	X	1 Mid range, 4 short range	2Tb	Multiple High-end LiDARs

Table 2.5: Comparison between different dataset

3

IMPLEMENTATION

As we have seen in the state-of-the-art analysis in recent years new approaches in machine learning unlocked the possibility of segmentation in real time on 3D LiDAR point cloud without any need for data compression. However, we have found no solution that combine those new techniques with a tracking layer for further elaboration. The output of the segmentation layer, even if is good is susceptible to noise, but with the additional filtering layer the aim is to obtain a stable and safe estimation of the overall environment. Moreover, with the integration with ROS we present a ready to deploy solution capable of interacting with a real autonomous vehicle and to fully support self-driving capabilities. Given these observations, we propose the design and development of a novel perception system capable of detecting, classifying and tracking active and passive road objects combining different state of the art techniques and a custom filtering model. To this end, the system should be able not only to detect the vehicles, other road users, and obstacles but also to classify them into different categories and keep track of their motion, estimate speed and heading.
Given also the lack of testing in real-world scenarios for most of tracking systems, and the availability of a testing vehicle in our Lab, we also require our work to be evaluated on real-world data, avoiding potentially unrealistic simulations.

In the following pages we are going to discuss how we designed system architecture, we present the algorithms and the approaches used and give a clear explanation of the project choices.

3.1 SYSTEM ARCHITECTURE

The main objective of our system is to track active agents and detect static obstacles in the road scene around the ego-vehicle starting from LiDAR point cloud and without any other exteroceptive sensor input.
Given the system constraints we decided to adopt a centralized architecture where all the data are sent to a central node for elaboration. The system also falls into the category of systems with independent sensor pipeline as we have only the LiDAR as perception sensor.

In order to achieve the system objective, we need the list of classified obstacles and the heading, the speed and the steering angle of the ego-vehicle.

For this reason, the system's first block is a segmentation node. This component is meant to identify the position of objects in the point cloud, associate a class based on object features and determine its dimensions and heading. Then the system sends the list of objects to a pre-processing algorithm contained in the tracking node that maps objects between frames and filter out noisy detection. Once the objects are matched are then used as input of the Extended Kalman Filter. The filter outputs present its final view of the world by combining the custom model prediction to the actual network detections. The final estimation provides the controller with accurate highly informative data that can be later use in the decision-making algorithm. At the beginning of the project, we fixed some constraint. In pursue of keeping the overall cost of the system as low as possible, but reliable, we based the detection pipeline only on LiDAR, being the most adopted sensor in autonomous vehicles and given that the cost per unit is dropping very fast. Moreover, we decided to be agnostic on the sensor specification accepting, with small changes to setting parameters, basically all the LiDAR sensors available on the market. Our system can accept every point cloud input both from sensors with a limited FOV and from 360 ° sensors, regardless of the number of Layers and planar resolution.

Moreover a modular design has been chosen in order to guarantee code separation for future maintenance, upgrades and to allow for a faster and easier customization of the system itself.
Another constraint of the project was the use of Robot Operating System (ROS) as a framework for the integration of the modules since it provides already a structure that handles communication and integrates multiple data structure particularly handy for AV and robotic applications.
The ROS system architecture is shown in Figure 3.1. The code-base in his final form is fully integrated with ROS and is composed by two sub modules. The

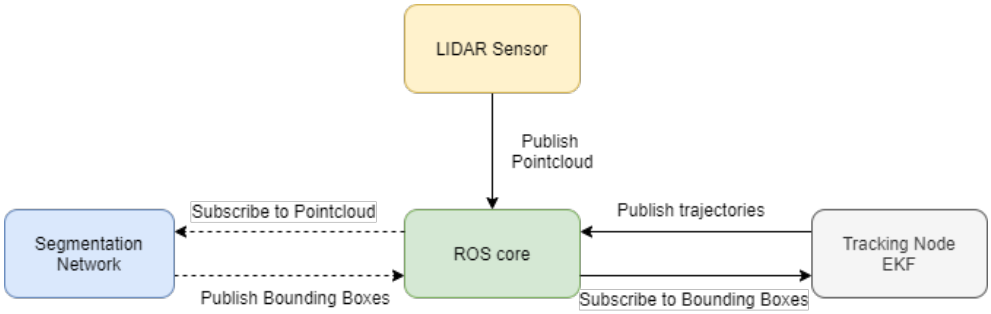


Figure 3.1: System architecture

first one is the inference node that takes as input LiDAR data and outputs annotated bounding boxes, the second one is the tracking node that takes as input the list of bounding boxes, kinematics data from the ego-vehicle and outputs a trajectory of each vehicle, also acting as filtering node to further improve the inference output.

3.2 DATA ACQUISITION AND CONVENTIONS

The first and obvious step of the system is data acquisition. To this end, we expect the acquisition process to be performed with a vehicle, equipped with the necessary sensors. For the segmentation layer the only required data are the ones coming from the LiDAR sensor that should be mounted on top of the vehicle to provide a 360° view around the vehicle itself. For the filtering node we also require GPS position, steering angle and speed of the ego-vehicle.

For the entire scope of our work, we represent the vehicle following the coordinate systems indicated by the ISO 8855 [57], with the origin on the ground below the center of mass of the vehicle, and axis x, y and z pointing respectively forward, left, and up. The LiDAR reference frame has its origin set in the sensor center of rotation and with the axis oriented in the same direction of the one of the ego-vehicle. Every other bounding box will follow the same standard.

Data coming from the sensor are represented using ROS PointCloud2 message, a message type designed specifically for LiDAR sensors. It is composed by a header that contains metadata regarding the information in the message body. In the header, we find the number of layers, the name of the fields available for each point and their bit sizes. The majority of LiDAR sensor, depending mainly by the vendor and the sensor cost retrieve more information than strictly required for our application, like reflectivity, the normal, the intensity, and many others. To speed up network elaboration and lower the

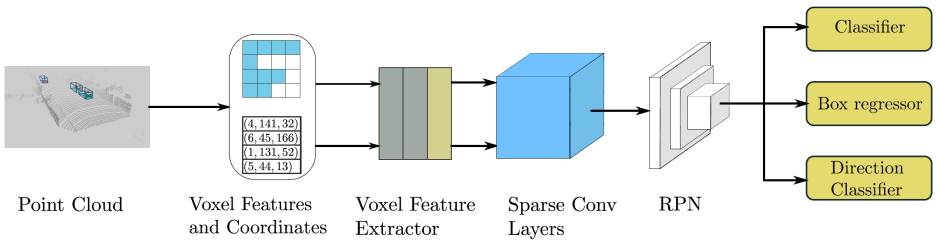


Figure 3.2: SECOND network structure

footprint in memory we keep only necessary information representing each point by means of only x, y and z coordinates and removing the header.

3.3 INFERENCE NODE

To reach optimal performance we started the development of the inference node following the architecture of the segmentation network proposed in [34]. The network architecture is represented in Figure 3.2. We implemented the network adapting the structure of SECOND detector to work with NuScenes dataset as training input. This part of the system has been developed both as standalone inference system and as ROS node. Any piece of code in ROS that takes an input and/or publish an output is called node. In both cases we integrated a visualization tool to help with debug and to visually validate our results.

In this section, we describe the architecture of the SECOND detector and present the relevant details regarding training and inference phases peculiar to our implementation.

3.4 NETWORK ARCHITECTURE

The network structure can be split in four macro blocks:

1. Point Cloud preprocessing.
2. Voxel wise feature extractor.
3. Sparse convolutional middle layer.
4. Region Proposal Network (RPN).

3.4.1 *Point Cloud Clustering*

Here we describe the followed approach to convert raw input data obtained from the LiDAR to voxels in the so called voxelization procedure where we give a physical volume to a-dimensional points. We first pre-allocate buffers based on the specified limit on the number of voxels set in the configuration file then, based on the maximum memory available in the system, we iterate over the points in the cloud and assign each one to a corresponding voxel; we save the voxel coordinates and the number of points per voxel. We check the existence of the voxels based on a hash table during the iterative process. If the voxel related to a point does not yet exist, we set the corresponding value in the hash table; otherwise, we increment the number of voxels by one.

The iterative process will stop once the number of voxels reaches the specified limit. Finally, we retrieve all voxels, their coordinates, and the number of points per voxel. This process is carried out from the closest point outward so that if the maximum memory limit is reached this happened to point further away from the ego-vehicle with a high probability that are not relevant for the analysis.

3.4.2 *Feature Extractor*

We implemented a voxel feature encoding (VFE) layer, as proposed in VoxelNet [24], to extract relevant features from a voxel cloud.
This layer takes all points in the same voxel as input and uses a fully connected network (FCN) consisting of a linear layer, a batch normalization (BatchNorm) layer and a rectified linear unit (ReLU) layer to extract pointwise features. Then, it applies max pooling to obtain the locally aggregated features for each voxel. At the end all the single voxel features are combined together. The voxelwise feature extractor consists of several VFE layers and an FCN layer.

3.4.3 *Sparse Convolutional Middle Extractor*

In spatially sparse convolution, firstly introduced in [54] the output points are not computed if there is no related input point. This approach offers computational benefits in LiDAR-based detection because the grouping step for the point clouds end up with a low sparsity coefficient since LiDAR data are quite dense. This middle layer extracts region of interest from a simplified 2d bird eye view of the point cloud. In this way the network performs the segmentation 3D only in areas where interesting features are found saving resources and greatly speedup the inference process.

3.4.3.1 *Region proposal network*

The RPN architecture is composed of three stages. Each stage starts with a down sampled convolutional layer, which is followed by several convolutional layers and finally by BatchNorm and ReLU layers. We then up sample the output of each stage to a feature map of the same size and concatenate these feature maps into one single map. Finally, three 1 x 1 convolutions are applied for the prediction of class, regression offsets and direction.

3.4.4 *Network Models*

The network results are mainly influenced by training data and by the network model. To obtain best performance we created and tested different configurations of the network and carried out training on the most promising 4 models that performed the best on average on the 9 classes we want to identify. The developed model can be easily configured thanks to the configuration file that contains all the important parameters for general network structure and for the identification of each class. In there we find general parameters like the number of iterations, the dimensions of inputs and outputs, number of workers, data paths for training and testing sets and class specific parameters that identify the features of each class by means of maximum and minimum dimensions. We find also strengthen parameters as noise and data augmentation approaches such as rotations and data transformation that make the final model better in generalization and reduce over-fitting.

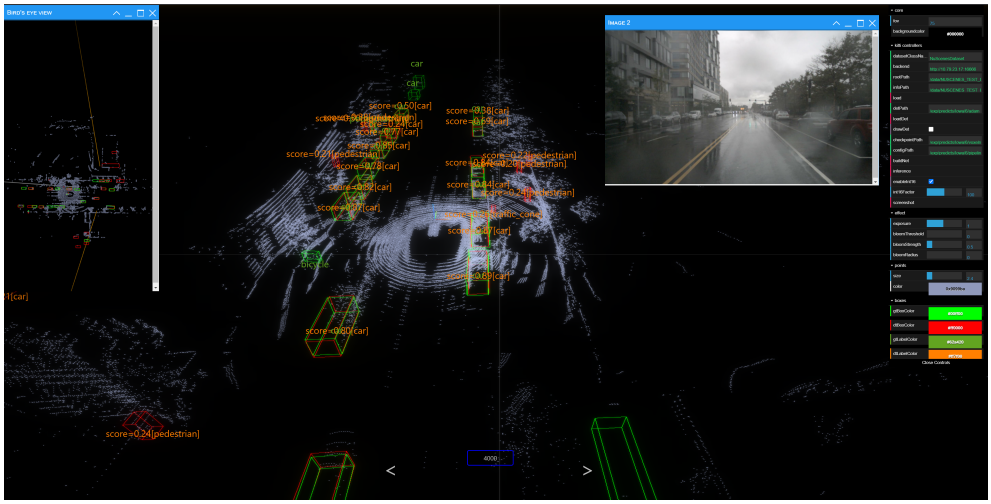


Figure 3.3: Network segmentation results from NuScenes test set data displayed in KittiWebViewer

3.4.5 *Result Visualization*

We integrated in both solution tools for showing the outputs of the network in an easy-to-understand way by means of visual representation. Concerning the solution without integration with ROS we modified KittiWebViewer, a web app compatible with KITTI data format that can show the Point Cloud in input, ground truth bounding boxes and the network outputs boxes, classes, and confidence in real time. Moreover, it shows the corresponding image for visual result comparison and a BEV of the point cloud with predicted and ground truth bounding boxes. This viewer has been modified to show NuScenes Data and every other custom LiDAR data and the visualization results have been enriched by adding the class and confidence label for the predicted boxes and the label for the ground truth. The tool is convenient for a fast and easy comparison between different models thanks to the fact that allows for real time model exchange keeping the same data in input. It has been a fundamental piece of software during the creation of the different models.

For ROS implementation instead we relied on Rviz, the visualization tool included in ROS that provides a ready to use, stable and light viewer for the system.

3.5 TRACKING NODE

This feature has been implemented only in ROS version of the system since is based on data structure and subsystems provided by the Robotic Operating System. The aim of this module is to filter the output of the network and

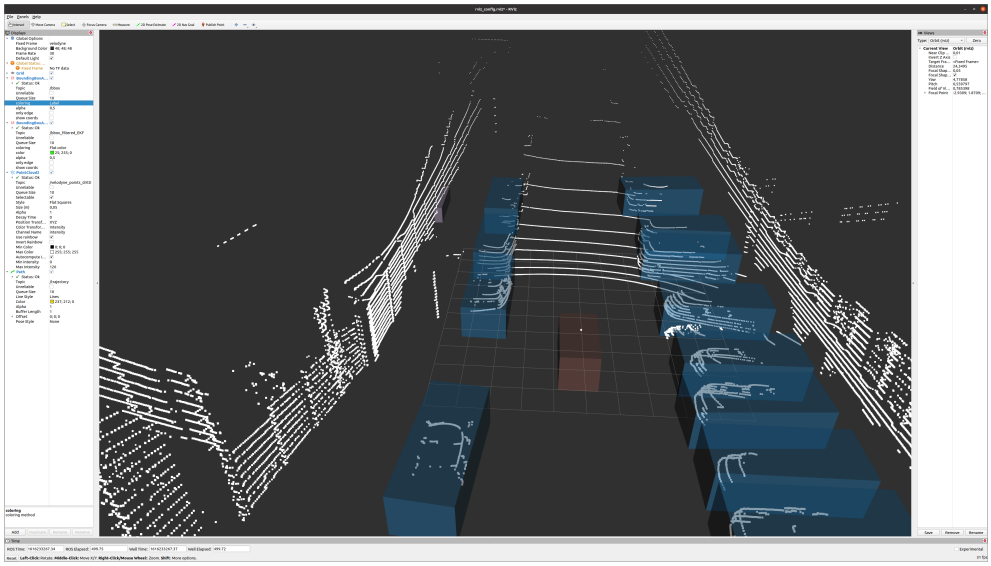


Figure 3.4: Network segmentation results from custom VLP16 data displayed in Rviz

provide multiple objects tracking ability over time of the agents around the ego-vehicle. At the system core we developed an Extended Kalman Filter, a well-known technique used for state estimation. The node implemented in python subscribes to the bounding boxes topic and to the topics of the steering angle, speed and position of the ego vehicle, all required data necessary for the computation of the output.

3.5.1 *Data pre-processing*

The main code, once it receives all the data pre-process them to obtain the variables required by the filter.
The first important parameter is the ego-heading. We explored different approaches during the development of the filter for the computation of the heading of the vehicle. The only absolute data we have are the front and back GPS positions. From GPS we decided to try computing the heading as the angle between two vectors passing through 2 couple of points. At startup, the systems uses the first 10 measurements, that we will call p_f and p_b, from the front and from the back sensor and average them out. The vector passing between the position of the front sensor and the back sensor becomes the starting heading or heading 0. From there every new couple of measurements, called respectively p'_f and p'_b, are used, with some trigonometry, to compute

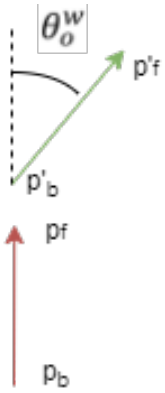


Figure 3.5: Drawing for ego-vehicle heading computation

the new heading as shown in Figure 3.5 and using the Equation 3.1. The computation of the heading θ_o^w is done using the formula:

$$\tan\theta_o^w = \frac{-\frac{y_{p_f}-y_{p_b}}{x_{p_f}-x_{p_b}} + \frac{y_{p'_f}-y_{p'_b}}{x_{p'_f}-x_{p'_b}}}{1+(-\frac{y_{p_f}-y_{p_b}}{x_{p_f}-x_{p_b}})(-\frac{y_{p'_f}-y_{p'_b}}{x_{p'_f}-x_{p'_b}})} \tag{3.1}$$

This method demonstrated to be quite noisy for our purposes. We decided to try a simple variation of the previous solution where, instead of using the front and back position, we compute the second vector using two consecutive ego-poses obtained as an average of the front and back position. The equation is identical to the previous one but instead of using p'_b we will use p_{old} and instead of p'_f we will use p_{new}:

$$\tan\theta_o^w = \frac{-\frac{y_{p_f}-y_{p_b}}{x_{p_f}-x_{p_b}} + \frac{y_{p_{new}}-y_{p_{old}}}{x_{p_{old}}-x_{p_{old}}}}{1+\left(-\frac{y_{p_f}-y_{p_b}}{x_{p_f}-x_{p_b}}\right)\left(-\frac{y_{p_{new}}-y_{p_{old}}}{x_{p_{old}}-x_{p_{old}}}\right)} \tag{3.2}$$

For both methods we find the heading computing the arc-tangent of the previously found value:

$$\phi = \arctan(\tan\theta_o^w) \tag{3.3}$$

Even if the results obtained with this approach are slightly better are still far from ideal. For this reason another solution has been tested with success. The method relies on two consecutive poses that we will call p_{old} and p_{new} and employs the sine and cosine to estimate the heading. We have:

$$\sin\theta_o^w = \frac{y_{p_{new}} - y_{p_{old}}}{\sqrt{(x_{p_{new}} - x_{p_{old}})^2 + (y_{p_{new}} - y_{p_{old}})^2}} \tag{3.4}$$

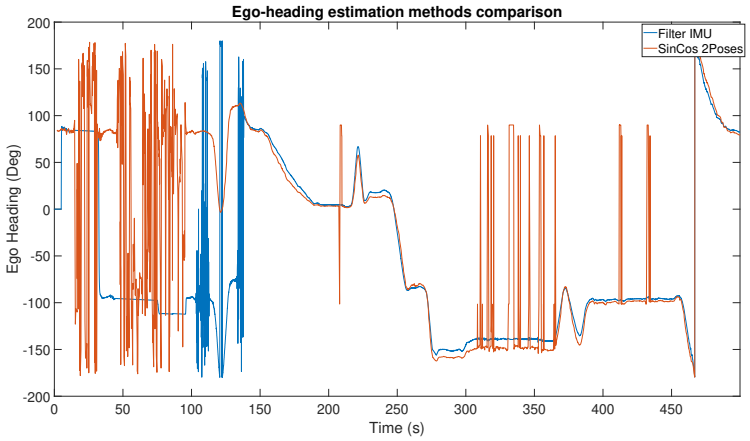


Figure 3.6: Ego-vehicle heading estimation results from "Filter IMU" and "SinCos 2Poses" methods in degrees on test data

$$\cos\theta_o^w = \frac{x_{p_{new}} - x_{p_{old}}}{\sqrt{(x_{p_{new}} - x_{p_{old}})^2 + (y_{p_{new}} - y_{p_{old}})^2}} \tag{3.5}$$

the angle is obtained computing the arc sine of the value found:

$$\theta_o^w = \arcsin\sin\theta_o^w \tag{3.6}$$

Then we evaluate the sine and cosine sign to find in which quadrant we are and to compensate adding or subtracting $\pi/2$ rad to obtain the correct heading.

This approach demonstrated to be the most accurate and stable when data from GPS are present. We had also the opportunity to test an external method for computation of the ego-heading that uses a filter. This filter combines data from GPS and IMU of the vehicle to obtain a reliable and precise estimation of the heading as explained in [62]. The filter fills the missing data by estimating the heading based on IMU input. We used this ideal estimation for all our tests and as a comparison with our estimation methods. The results from test data are compared in Figure 3.6. It is important to note that we need a relative heading and for an easier comparison the graphs has been translated to match each other. All the estimations are normalized between plus and minus π rad. In Figure 3.6 it is clear that the heading estimation based on sine and cosine is quite accurate even if it's spiky. The Filter IMU as you can see is more stable and do not have problem in estimating the heading even when some GPS data are missing. At the beginning of the test run however GPS signal was low and inaccurate and this lead to unstable and bad heading estimation. Once we have the heading angle, we still require the wheels angle which is easily obtained from the steering angle divided by 18, we obtain the

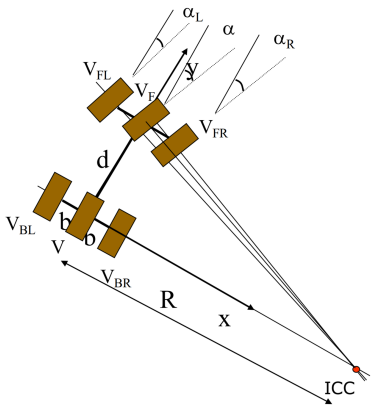


Figure 3.7: Ackerman steering (bicycle approximation)

ego-vehicle speed directly from the vehicle telemetry and we compute the angular speed using the approximation of the Ackerman kinematics to the bicycle model as shown in Figure 3.7.

Bicycle approximation simplify significantly the analysis without introducing a significant error. Thanks to this simplified model we compute the turning radius of the vehicle:

$$R = \frac{d}{\tan \alpha} \tag{3.7}$$

We can also compute the ego-vehicle angular speed from the vehicle speed v_o^w, vehicle inter axis d and wheels turning angle α, all parameters that we have. From the bicycle model we know that:

$$v_o^w = \frac{\omega d}{\sin \alpha} \tag{3.8}$$

$$v_o^w = \omega R \tag{3.9}$$

Combining those two equations we get:

$$\omega_o^w = v_o^w \frac{\tan \alpha}{d} \tag{3.10}$$

In parallel to those computations all the bounding boxes are filter out and divided into two categories: active objects containing all moving agents (cars, trucks, pedestrians, cyclists, motorcyclist, construction vehicles, busses) and static objects (traffic cones and barriers). The filter will be active only on moving objects that are interesting for situation assessment where else the static ones will be left unchanged. Now that we have all the data required for the filter and the active bounding boxes, we can introduce our filter model.

Let us define the notation used for a clearer understanding. In the following pages we will use the symbol x_t^w to denote for the position over the abscissa

of the tracked object with respect to the world and similarly x_o^w to denote the abscissa of the observer with respect to the world. All the other notation follows this rule.

3.5.2 *Prediction*

From the analysis of the problem, it is clear that we have to model the motion of a third-party entity, the tracked object from a moving observer, our ego-vehicle. To do that a simple Kalman Filter is not sufficient for modelling the complex dynamics so we opted for the Extended version of the Kalman Filter that uses the trick of linearization by employing the Taylor first order approximation to deal with nonlinear systems.

We firstly define the position of the tracked object with respect to the world as:

$$X_t^w = X_o^w + X_t^o \rightarrow \dot{X_t^w} = \dot{X_o^w} + \dot{X_t^o} \tag{3.11}$$

We will call θ_t^w simply θ since θ_o^w is equivalent to the heading of the ego-vehicle that we call ϕ.

Moving to the cinematic analysis of the motion we can write the following equations. For the Tracked object we have:

$$\begin{cases} x_t^w = x_t^w + v_t^w c_t^w \Delta t \\ y_t^w = y_t^w + v_t^w s_t^w \Delta t \end{cases} \tag{3.12}$$

For the observer ego-vehicle view point instead:

$$\begin{cases} x_o^w = x_o^w + v_o^w c_o^w \Delta t \\ y_o^w = y_o^w + v_o^w s_o^w \Delta t \end{cases} \tag{3.13}$$

Focusing the analysis on the speeds and combining the two systems:

$$\begin{bmatrix} v_t^w c_t^w \\ v_t^w s_t^w \\ \omega_t^w \end{bmatrix} = \begin{bmatrix} v_o^w c_o^w \\ v_o^w s_o^w \\ \omega_o^w \end{bmatrix} + \begin{bmatrix} v_t^o c_t^o \\ v_t^o s_t^o \\ \omega_t^o \end{bmatrix} \tag{3.14}$$

From where we derive the following formula:

$$\begin{bmatrix} \dot{x_t^o} \\ \dot{y_t^o} \\ \dot{\theta_t^o} \end{bmatrix} = \begin{bmatrix} v_t^o c_t^o \\ v_t^o s_t^o \\ \omega_t^o \end{bmatrix} = \begin{bmatrix} v_t^w c_t^w \\ v_t^w s_t^w \\ \omega_t^w \end{bmatrix} - \begin{bmatrix} v_o^w c_o^w \\ v_o^w s_o^w \\ \omega_o^w \end{bmatrix} = \begin{bmatrix} v_t^w c_t^w - v_o^w c_o^w \\ v_t^w s_t^w - v_o^w s_o^w \\ \omega_t^w - \omega_o^w \end{bmatrix} \tag{3.15}$$

And extending the solution to a general angle we can write the model like so:

$$\begin{bmatrix} \dot{x_t^o} \\ \dot{y_t^o} \\ \dot{\theta_t^o} \end{bmatrix} = \begin{bmatrix} v_t^w c_o^w c_t^o - v_t^w s_o^w s_t^o - v_o^w c_o^w \\ v_t^w s_o^w c_t^o + v_t^w c_o^w s_t^o - v_o^w s_o^w \\ \omega_t^w - \omega_o^w \end{bmatrix} \tag{3.16}$$

To double check the solution proposed we can impose $\theta_o^w = 0 \rightarrow$ which is the case of relative motion in the same direction.

$$\begin{bmatrix} \dot{x}_t^o \\ \dot{y}_t^o \\ \dot{\theta}_t^o \end{bmatrix} = \begin{bmatrix} v_t^w c_o^w c_t^o - v_t^w s_o^w s_t^o - v_o^w c_o^w \\ v_t^w s_o^w c_t^o + v_t^w c_o^w s_t^o - v_o^w s_o^w \\ \omega_t^w - \omega_o^w \end{bmatrix} = \begin{bmatrix} v_t^w c_t^o - v_o^w \\ v_t^w s_t^o \\ \omega_t^w - \omega_o^w \end{bmatrix} \tag{3.17}$$

A similar test can be done imposing the observer stationary respect to the world. In this case we impose $v_o^w = 0$ and $\omega_o^w = 0$ for an easier analysis we can also impose the coincidence of the reference system of the world and of the observer by fixing $\theta_o^w = 0$.

$$\begin{bmatrix} \dot{x}_t^o \\ \dot{y}_t^o \\ \dot{\theta}_t^o \end{bmatrix} = \begin{bmatrix} v_t^w c_o^w c_t^o - v_t^w s_o^w s_t^o - v_o^w c_o^w \\ v_t^w s_o^w c_t^o + v_t^w c_o^w s_t^o - v_o^w s_o^w \\ \omega_t^w - \omega_o^w \end{bmatrix} = \begin{bmatrix} v_t^w c_t^o \\ v_t^w s_t^o \\ \omega_t^w \end{bmatrix} \tag{3.18}$$

We obtain the final formulation of the cinematic of the motion:

$$\begin{bmatrix} x_t^{o\prime} \\ y_t^{o\prime} \\ \theta_t^{o\prime} \\ v_t^{o\prime} \\ \omega_t^{o\prime} \end{bmatrix} = X_t^o + \Delta t \begin{bmatrix} v_t^w c_t^o - v_o^w \\ v_t^w s_t^o \\ \omega_t^w - \omega_o^w \\ 0 \\ 0 \end{bmatrix} = F_t^o \begin{bmatrix} x_t^o \\ y_t^o \\ \theta_t^o \\ v_t^o \\ \omega_t^o \end{bmatrix} + \begin{bmatrix} -v_o^w \Delta t \\ 0 \\ -\omega_o^w \Delta t \\ 0 \\ 0 \end{bmatrix} \tag{3.19}$$

We can finally compute the matrix F_t^o:

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial \boldsymbol{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \tag{3.20}$$

And we obtain:

$$F_t^o = \begin{bmatrix} 1 & 0 & -v_t^w \sin\theta_t^o \Delta t & \cos\theta_t^o \Delta t & 0 \\ 0 & 1 & v_t^w \cos\theta_t^o \Delta t & \sin\theta_t^o \Delta t & 0 \\ 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

And the matrix H_k:

$$\mathbf{H}_k = \left. \frac{\partial h}{\partial \boldsymbol{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}} \tag{3.22}$$

And we obtain:

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.23}$$

We can now write the model of the system:

$$\begin{bmatrix} x_t^o{}' \\ y_t^o{}' \\ \theta_t^o{}' \\ v_t^o{}' \\ \omega_t^o{}' \end{bmatrix} = \begin{bmatrix} 1 & 0 & -v_t^w \sin\theta_t^o \Delta t & \cos\theta_t^o \Delta t & 0 \\ 0 & 1 & v_t^w \cos\theta_t^o \Delta t & \sin\theta_t^o \Delta t & 0 \\ 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t^o \\ y_t^o \\ \theta_t^o \\ v_t^o \\ \omega_t^o \end{bmatrix} + \begin{bmatrix} -v_o^w \Delta t \\ 0 \\ -\omega_o^w \Delta t \\ 0 \\ 0 \end{bmatrix} \tag{3.24}$$

The filter presented is incorporated in the code in the prediction phase. We define additional matrices used in the implementation of the filter:

$$C = \begin{bmatrix} -v_o^w \Delta t \\ 0 \\ -\omega_o^w \Delta t \\ 0 \\ 0 \end{bmatrix} \tag{3.25}$$

The only missing parameter is the speed of the tracked object with respect to the world that is computed as the sum of ego-speed vector and the estimated tracked object speed:

$$v_t^w = \pm\sqrt{\left(v_o^w \cos\theta_o^w + v_t^{o\prime} \cos\left(\theta_t^{o\prime} + \theta_o^w\right)\right)^2 + \left(v_o^w \sin\theta_o^w + v_t^{w\prime} \sin\left(\theta_t^{o\prime} + \theta_o^w\right)\right)^2} \tag{3.26}$$

Finally we write the formulation for the prediction:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{3.27}$$

Whit:

$$f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) = F\hat{\mathbf{x}}_{k-1|k-1} + C \tag{3.28}$$

The predicted error covariance is:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k \tag{3.29}$$

3.5.3 *Correction*

In the correction step the prediction output is adjusted by incorporating the new knowledge provided by the sensor and all the internal parameters of the filter are updated accordingly.
Firstly, the near-optimal Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top \mathbf{S}_k^{-1} \tag{3.30}$$

Where Innovation (or residual) covariance K is computed as the pseudo inverse of:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k \tag{3.31}$$

Then we obtain the new corrected state as:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{3.32}$$

Where $\tilde{\mathbf{y}}_k$ is:

$$\tilde{\mathbf{y}}_k = z_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{3.33}$$

and z_k is the new measure. Finally, the updated covariance estimate is computed:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{3.34}$$

3.5.4 *Temporal data association*

For associating different bounding boxes between frames, we tested different approaches. The first and the simplest one is the Euclidean distance where for every element tracked, we compute the geometrical distance to every new detected bounding box and associate the two if the new one is not yet associated and its distance is smaller than a given threshold. Even if this method is quite accurate, it is not optimal for our solution since it does not consider the uncertainty associated with the filter. In the final solution we decided to introduce Mahalanobis distance. This approach compares the position of the vehicle with respect to a given distribution. The distance is computed using the formula:

$$D_M(\vec{x}) = \sqrt{(u_k - \hat{x}_{k|k-1})^T \mathbf{S}^{-1} (u_k - \hat{x}_{k|k-1})}. \tag{3.35}$$

Where S is the covariance matrix computed by the filter. Later we also added a constraint on the physical dimensions of the bounding box that should be close to the new one. Last but not least in the final iteration we also use a class similarity which associates the two bounding boxes only if all the preceding constraints have been satisfied and the label given by the network to the new bounding box is, as we have defined it, exchangeable, basically if the tracker label is a class that for average shape and features can be misclassified by the network. All those constraints allow for a unique association between two frames and a couple of bounding boxes resulting in a pretty accurate matching.

Once we have associated all the new bounding boxes with the old one, we update all the remaining trackers and boxes. In particular trackers without any matching will increase the skipped frames counter and the unmatched boxes will be associated to new trackers.

Then all the trackers with a skipped frame counter greater then a certain threshold will be killed meaning that the vehicle they are tracking exited the visible area. On the other side all the trackers with an active history will be kept alive. Before publishing the results a final check is performed on overlapped boxes where for each box we check if its shape lays partially inside another box volume. If so the box with a lower score is removed leaving only the best one.

4

EXPERIMENTAL RESULTS

In this chapter we proceed with the evaluation of the proposed solution and compare the performance with the state-of-the-art currently available. The validation of a system can in general be done in several ways, according to the type of problem it solves, the tools available and the nature of the system to be validated. Given that the problem we are tackling is quite new, when compared to other engineering fields, and in our solution we extensively use machine learning processes, in literature has not been defined any standard procedure for testing. For this reason, we have combined numerical results and comparison with extensive testing on real world data building statistics to easily compare our system output with the results presented in papers of other solution.

At the beginning of the chapter, we will present the two vehicles used for the development of the solution, then we will go on showing experimental results obtained and comparisons with other similar systems.

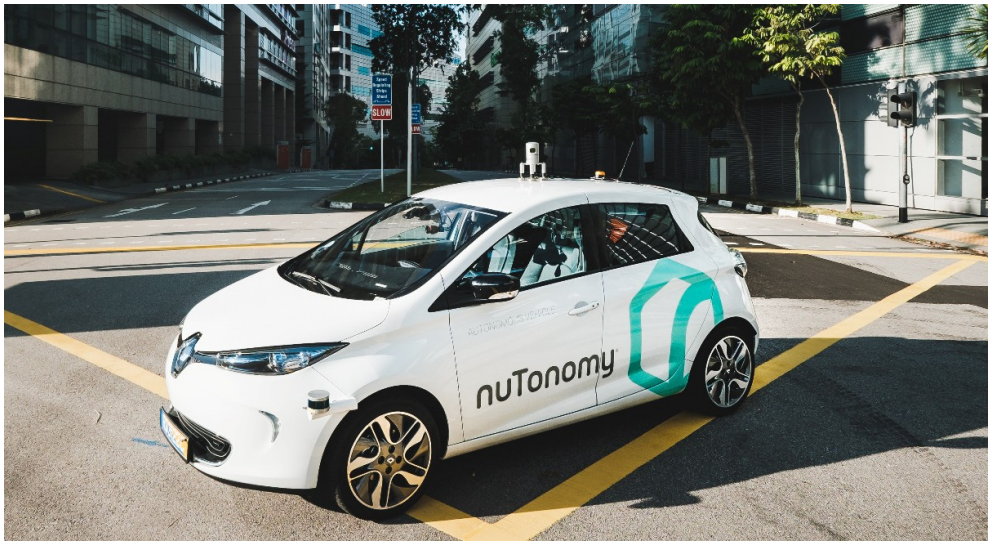


Figure 4.1: NuTonomy autonomous vehicle prototype based on Renault ZOE

4.1 NUTONOMY AV PROTOTYPE

The base for the NuTonomy experimental vehicle is a Renault Zoe with a complete and modern sensor suite. As already mentioned above, the vehicle is equipped with a HDL32 Velodyne LiDAR, 6 high resolution cameras sited in strategic position to cover the surrounding of the vehicles at 360°, a radar that points forward, an IMU and a GPS.
This setup is interesting because it is close to the one we have on the experimental vehicle at Politecnico di Milano that is important for a reliable and safe reuse of the model trained on the data coming from this car.

4.2 DATA DESCRIPTION

The dataset comes in a custom format and contains over 1000 scenes of 20 second each manually selected to show a diverse and interesting set of driving maneuvers, traffic situations and unexpected behaviors. The data are in raw format and contains images from the 6 cameras, lidar point cloud and radar data, GPS and IMU sorted in different folders. All the data are provided with additional metadata and sensor information. For Neural Network training purpose data come already annotated and split in a train and test set randomly selected to avoid biases. Focusing on LiDAR data, the point clouds are recorded at a frequency of 2Hz and are provided with the annotation of objects in from of bounding boxes with a center, the 3 dimension and the tree rotation in space with respect to the object center and a class label.

The classes available are ten identified by a number from 0 to 9 in this order:

0. Car
1. Bicycle
2. Bus
3. Construction vehicle
4. Motorcycle
5. Pedestrian
6. Traffic cone
7. Trailer
8. Truck
9. Barrier

The data are automatically labelled using a custom software provided by SCALE, a company specialized in solution for autonomous driving application. This software works by extracting the labels from the images using a neural network image classifier and then applies the labels to the 3D point cloud reconstructing the 3D shape of vehicles, the dimensions, and the heading combining different consequent images. Before the dataset publication, all the data annotations have been manually checked and fixed if necessary leading to an accurate and reliable annotated dataset. The annotation covers all the objects in the sensor visible area and for every annotation in the point cloud a corresponding annotation is present on the corresponding images for visual examination. The only note about this software is that this annotation algorithm works with the complete scene having a-priori all the images and point clouds of the 20 second scene therefore tends to produce some bounding boxes with as low as a dozen LiDAR points in some scenes, border line cases that are discarded in a pre-filtering step before the data are used in the training procedure.

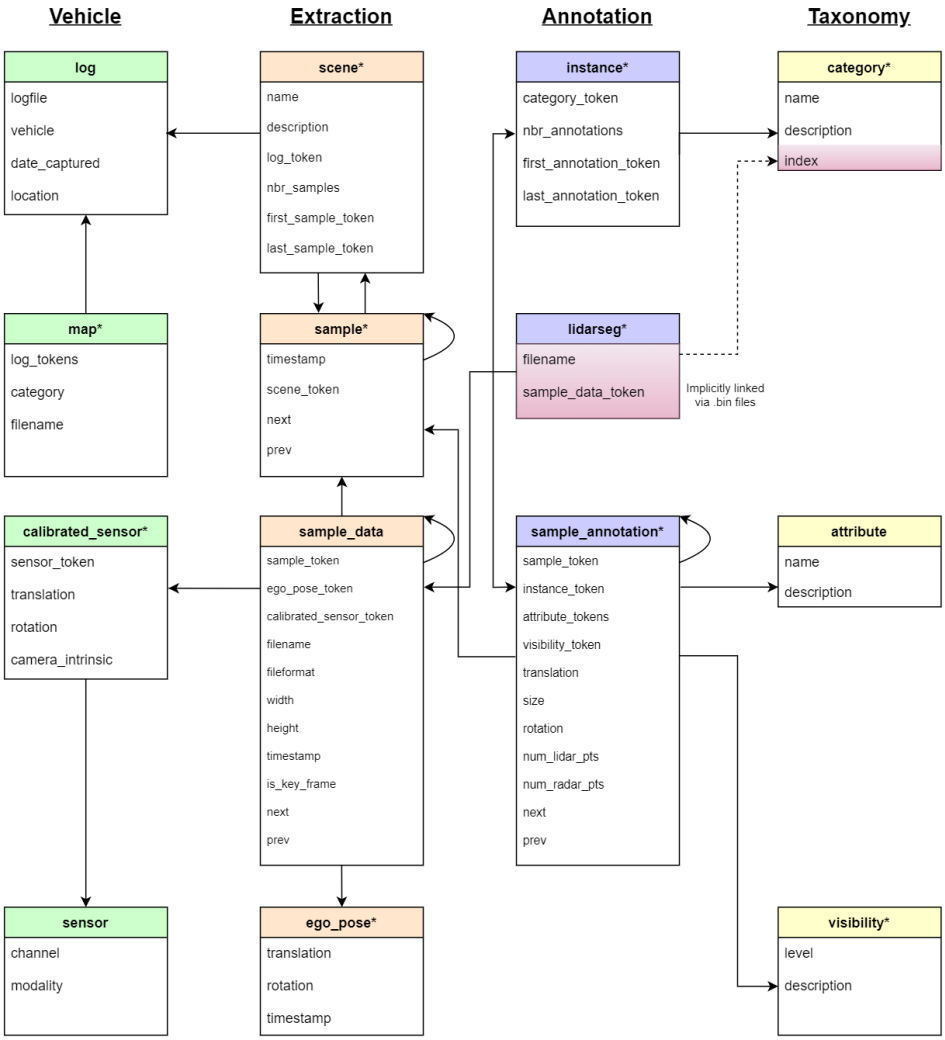


Figure 4.2: NuScenes dataset data format

4.3 POLIMI AV PROTOTYPE

The experimental vehicle was developed by the Department of Mechanics and the Department of Computer Science of Politecnico and is specifically meant as a test platform for experimenting new solutions for autonomous driving and advance driving assistance solutions. The experimental setup is built around a ZED One electric car. The vehicle can be either equipped with a Velodyne VLP16 or with HDL32E lidars mounted on the roof for 360 degrees field of view. The LiDAR is mounted on a slightly tilted forward support to direct and cover all the area in front of the vehicle. The sensor suite also includes a Continental Radar, a Leddartech M16 solid state laser and a Texas Instruments AWR 1642 Radar. On top of the vehicle, facing forward, is also mounted a Stereolabs ZED stereo camera. The absolute position of the vehicle is provided by a set of Swiftnav RTK GPS, one mounted on the hood of the car and one mounted on the roof at the back of the car. Odometry data like the steering angle and the vehicle speed are provided by the car telemetry while the IMU provides acceleration and inertial measurements of the motion.
On the vehicle is installed a Jetson Xavier embedded board built by NVIDIA corporation that runs ROS and handles data recording and can also support processing for our system.

Figure 4.3: Polimi autonomous vehicle prototype

4.4 DATA FORMAT

All listed sensors provide information on the surrounding of the car, but they differ in precision, field of view, and quality of data. In particular, the LiDAR is the only sensor retrieving information at 360 degrees, ranging from 3 meters up to 100 meters with centimeter precision on 16 layers for VLP16 model and on 32 layers for HDL32E model. The retrieved data from the LiDAR is a dense point cloud, a 3D representation of the environment. The speed data and steering angle are contained in a specific topic called swiftnav and the GPS information are stored into two custom messages.
The dataset came in the format of a ROS bag. Bags are the primary mechanism in ROS for file data logging: the bag file tool chain is used to record datasets, visualize, label and store them for future use. The bag contains many topics, reflecting all the information stored in it; of these topics, we have been working mainly on the ones coming from LiDAR, GPS and odometry.

4.5 ADDITIONAL EXPERIMENTAL HARDWARE

During the development of the project, to support specific heavy tasks also external computational resources have been employed. In particular, multiple slots on the Polimi Westworld server have been used. The server provides environment with 1 dedicated 12GB GeForce GTX1080Ti out of 8 shared,

40 shared 2.20 GHz Intel Xeon E5 CPU cores, and shared 256 GB of RAM. Moreover, for testing purposes the entire pipeline was deployed also locally on a Ubuntu 20.04 machine fitted with an Intel i7 4770K 3.5 GHz 4 core CPU, a GeForce GTX780 with 3 GB dedicated memory and 16GB RAM.

4.6 RESULTS

We will present here the results obtained during this research work. To make the evaluation easier and more understandable we will split the analysis into two parts.

In the first part we will focus on the segmentation results, while in the second part we will present tracking and filtering results. Even if the two nodes work also independently and are analyzed here in separate sections the system has been tested also as a whole and is meant to be deployed as a unique block. The entire work was carried out in a Dockerized environment leveraging the power of containers. The virtual environment has all the libraries, configurations and software necessary for the training and the deployment of the system. The great advantage of this technique is the fact that by paying a little overhead in terms of performance we gain the possibility of porting the system from a machine to another, in our case from the Polimi server to a local machine or to the embedded system installed on the vehicle in matter of minutes without having to reconfigure all the environment.

4.6.1 *Network results analysis*

The development of the network was carried out in Python 3 using PyTorch and CUDA libraries to fully take advantage of NVIDIA hardware available in the training Polimi Westworld server.
The network was developed on the proposed structure by SECOND paper and was adapted to use NuTonomy NuScenes dataset as training and test data. This dataset has been selected for his diverse and complete set of driving scenarios and interesting sensor setup. After the code implementation has been completed, we mainly focused on testing different network configuration. Configurations of the networks are the key for good performance of the classifier. During the testing we notice significant differences in terms of performance and convergence speed between the tested models. Performance varied up to 10% in average precision on class bases and the convergence speed changed quite a lot and was up to double for some solutions.
We can analyze the configuration files by splitting them into three sections: the network configuration, the class features descriptors, and the training setup. The main changes between configurations are in the first two sections, the training setup, as previously mentioned has been kept unchanged for better comparison on convergence and network performance.
As the name implies the first document section contains the configuration of the network starting from pre-processing parameters for voxelization and clustering, continuing on network internal structure and finishing with the

definitions for initial losses and weights parameters. Here the main changes are on sample importance that varies between 0 and 1 and influences how much a new sample modifies the model and influences its output during the training process. Another relevant parameter is the non-maximum suppression (NMS) class agnostic that if set to true suppress nested or intersecting bounding boxes of any class. Even if, for the network output we obtained better results with this parameter set to false we have seen that having multiple bounding boxes proposal, even if overlapped, leads to a better result for the filter stage. In fact, in this way the filter can chose between different bounding boxes, choose the best matching box and discard all the others.
Regarding the class features descriptors, the main changes are on two parameters called "matched threshold" and "unmatched threshold". The two parameters define the acceptance threshold and affect how restrictive we are on similarity between features. The goal is to find the right value that enable to match two features if similar and instead discard them if they present differences. A lower value demonstrated to be more effective, basically we accept weaker features in exchange for more anchors to evaluate.

4.6.2 *Best models*

The four more promising models was compared to reach the one used in the final solution. Here we present and compare them; we have call them multi head area (mhead), mid area (mida), low area (lowa) and low area improved (lowa').
All those configurations were tested at the same training step with the same reduced dataset input. To speed up this pre analysis phase a subset composed by 1/8 of complete dataset was created randomly selecting 1 scene every 8 and used for this step. On average the training of each of those models took around 20h to reach the point of analysis.
For the comparison we have used a metric called Average Precision (AP) defined as:

$$AP@k = \frac{1}{N(k)} \sum_{i=1}^{k} \frac{TP(i)}{i} \tag{4.1}$$

$$N(k) = \min(k,\ TP_{tot}) \tag{4.2}$$

$$TP(i) = \begin{cases} 0 & i^{th} \text{ is } False \\ TP_{seen\ to\ i} & i^{th} \text{ is } True \end{cases} \tag{4.3}$$

Where k is the number of detections that we consider for the analysis. Once we have defined this parameter, we can easily compare the partial results obtained.

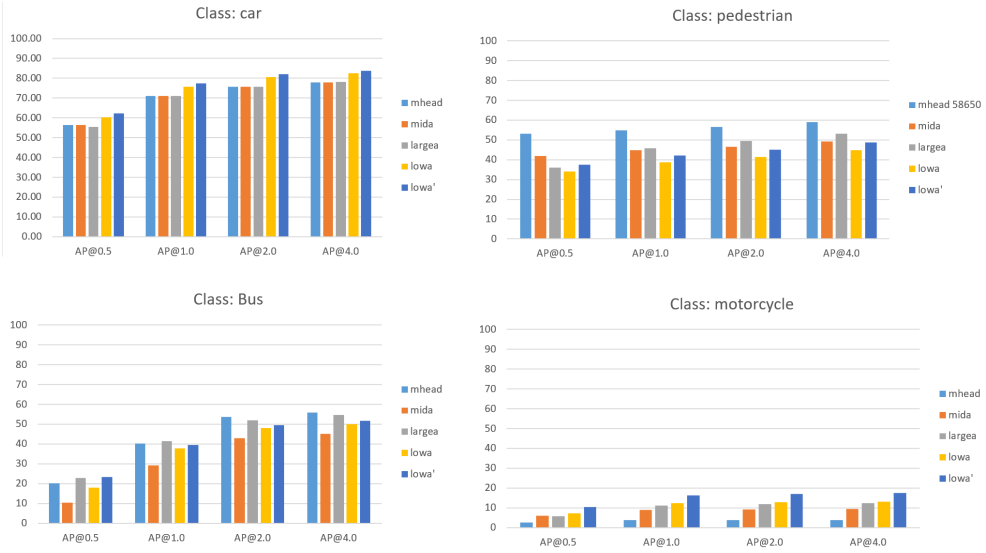


Figure 4.4: Performance comparison of different network models for specific classes

Looking at the graphs for the 4 more relevant classes in urban environment we can see that lowa configuration excels in the detection of cars and in general on mid sizes vehicles. Compared to the other three performs slightly worst on pedestrians but it has the best performance on barrier and traffic cones. Here are also reported the results for each class obtained by each of the four models developed. For a faster and easier comprehension each cell is filled with a different nuance from red to green. Values that are highlighted in green perform better than all the models in that specific class while the values highlighted in red perform the worst. It is important to note that reducing the dataset lead to the poor performance on rare classes like Construction Vehicles, Bicycles and Motorcycles that do not appear in train and test data reduced.

Configuration mhead										
Class	**Car**	**Motorcycle**	**Pedestrian**	**Bicycle**	**Bus**	**C. Vehicle**	**Trailer**	**Truck**	**Barrier**	**T. Cone**
AP@0.5	59.27	2.58	53.11	0.00	20.09	0.00	1.55	12.37	7.31	11.56
AP@1.0	73.23	3.73	54.84	0.00	40.11	0.00	13.01	23.85	20.7	12.25
AP@2.0	77.15	3.83	56.52	0.00	53.69	0.00	24.93	29.33	27.9	13.79
AP@4.0	79.31	3.9	58.96	0.00	55.76	0.00	33.71	32.01	34.16	17.09

Table 4.1: Network model mhead reduced dataset input performance for each class AP@k percentage

Configuration lowa										
Class	Car	Motorcycle	Pedestrian	Bicycle	Bus	C. Vehicle	Trailer	Truck	Barrier	T. Cone
AP@0.5	60.31	7.19	34.11	0.00	17.85	0.00	1.09	7.71	6.74	10.78
AP@1.0	75.76	12.27	38.8	0.00	37.66	0.00	13.54	20.41	28.79	14.97
AP@2.0	80.67	12.89	41.46	0.00	48.04	0.25	21.74	25.8	38.22	18.86
AP@4.0	82.44	13.19	44.91	0.00	49.84	0.66	28.65	28.74	43.06	24.17

Table 4.2: Network model lowa reduced dataset input performance for each class AP@k percentage

Configuration mida										
Class	Car	Motorcycle	Pedestrian	Bicycle	Bus	C. Vehicle	Trailer	Truck	Barrier	T. Cone
AP@0.5	56.30	5.89	42.01	0.00	10.38	0.00	0.03	6.48	4.53	7.49
AP@1.0	71.12	8.98	44.8	0.00	29.24	0.00	5.07	14.94	17.94	8.75
AP@2.0	75.68	9.30	46.62	0.00	42.89	0.00	13.14	19.84	24.65	10.52
AP@4.0	77.9	9.52	49.22	0.00	45.00	0.00	17.08	22.36	29.71	14.23

Table 4.3: Network model mida reduced dataset input performance for each class AP@k percentage

Configuration larga										
Class	Car	Motorcycle	Pedestrian	Bicycle	Bus	C. Vehicle	Trailer	Truck	Barrier	T. Cone
AP@0.5	55.36	5.81	36.00	0.00	22.77	0.00	3.15	12.13	7.74	6.48
AP@1.0	71.01	11.08	45.76	0.00	41.42	0.02	17.51	23.31	33.49	11.37
AP@2.0	75.65	11.77	49.57	0.00	51.91	1.41	29.65	30.28	42.59	15.46
AP@4.0	78.06	12.36	53.17	0.00	54.72	2.68	37.09	32.97	47.14	21.56

Table 4.4: Network model larga reduced dataset input performance for each class AP@k percentage

Method	Time (s)	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [8]	0.36	71.09	62.35	55.12	N/A	N/A	N/A	N/A	N/A	N/A
MV3D (LiDAR) [8]	0.24	66.77	52.73	51.31	N/A	N/A	N/A	N/A	N/A	N/A
F-PointNet [11]	0.17	81.20	70.39	62.19	**51.21**	**44.89**	**40.23**	**71.96**	**56.77**	**50.39**
AVOD [9]	0.08	73.59	65.78	58.38	38.28	31.51	26.98	60.11	44.90	38.80
AVOD-FPN [9]	0.1	81.94	71.88	**66.38**	46.35	39.00	36.58	59.97	46.12	42.36
VoxelNet (LiDAR) [14]	0.23	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND	**0.05**	**83.13**	**73.66**	66.20	51.07	42.56	37.29	70.51	53.85	46.90

Figure 4.5: SECOND inference performance from the paper

After extensive testing we decided to carry on the training of lowa that has been considered the more promising model and the one that on average performs the best over the all the classics. The model lowa' was trained in its final form with the complete NuScenes dataset for approximately 3 days equivalent to 234450 iterations on a virtual machine running on Westworld server with 1 GPU and 10 CPU cores. We trained the model up to the point where we reached the maximum performance, and the results tend only to degrade going forward in training.

In Table 4.5 are reported the final lowa' model AP@k for each class.

Configuration lowa'										
Class	**Car**	**Motorcycle**	**Pedestrian**	**Bicycle**	**Bus**	**C. Vehicle**	**Trailer**	**Truck**	**Barrier**	**T. Cone**
AP@0.5	62.36	10.42	37.48	0.00	23.33	0.00	3.79	10.15	9.35	12.42
AP@1.0	77.4	16.26	42.2	0.00	39.47	0.00	15.89	22.27	33.34	17.24
AP@2.0	82.1	16.96	45.09	0.00	49.43	0.78	25.13	28.97	43.03	21.87
AP@4.0	83.74	17.39	48.67	0.00	51.68	1.5	33.78	31.97	47.58	27.42

Table 4.5: Network final model lowa' performance for each class AP@k percentage

For comparison, even if the SECOND paper present data using a slightly different metric, on average SECOND paper results obtained by training the network with KITTI dataset resulted in an accuracy of 75.63% on average on cars whereas our solution reached 76,4% on the same class while keeping a similar inference time. On pedestrian instead our solution performed almost identically to SECOND correctly identifying the 43.36% of the cases, while in the original paper has been achieved an average precision of 43.64%.

LiDAR sensors used						
Model	Manufacturer	Layers	Range	Vertical Scope	Layer pitch	Price(€)
HDL32E	Velodyne	32	80m	10°/-30°	1.33°	30K
VLP16	Velodyne	16	100m	10°/-10°	1.33°	8K

Table 4.6: Comparison between LiDAR sensors used

4.7 LIDAR COMPARISON

Given that the network has been implemented to be agnostic to the LiDAR model we took the opportunity to test out different sensors to compare and understand which are the limits of the trained model in terms of resolution and number of layers of the sensor. The unit tested had been chosen because of one or more characteristics like being light, small, cheap, with a good quality or/and accuracy. All the sensors considered for the analysis are 360° LiDARs meaning we obtain a point cloud scan of the entire space around them but again the network can work without retrain also with LiDARs with a limited field of view, LiDAR from other brand or with different specification as long as the sensor output is a point cloud.
We had the opportunity to directly compare performance for the two Velodyne LiDARs by using a recorded bag with the more expensive model and removing the 16th lower layers. Given the similar construction characteristic we were able to estimate on the same scene the network output. Unfortunately, we do not have any ground truth of the vehicles and object present in the scene, but we decided to compare the identified objects at each instant with the two setups. The results demonstrate that the difference in number between detected objects by VLP16 and HDL32E is extremely small. As shown in the Figures 4.6 the pattern during the entire session is identical. Considering the two most common classes in urban environment, cars in Figure 4.7 and Pedestrian in Figure 4.8 the variation is minimal. The only big loss of detection happens for small objects closer to the vehicle.

The 32 layers sensor leads to more bounding boxes detections as predictable however all those boxes are located at the side rear of the vehicle and in the back. This region is not so critical for the driving task automation. From those results we can say that the difference in price is not justified by an increase in performance and this support our thesis objective that 16 layers sensor provide enough data to detect obstacles in the region of interest. Moreover, we have to remember that our LiDAR is slightly tilted forward, setup that somewhat increases the front detection but affects negatively the detection in the back area of the vehicle.

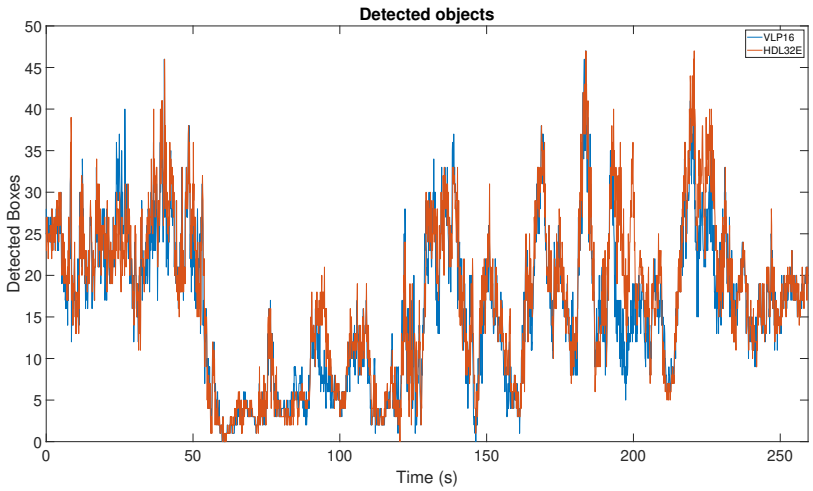


Figure 4.6: Performance comparison between detection of all classes: VLP16 in blue vs HDL32E in orange

4.8 TRACKING RESULTS

The second part of the thesis work has been devoted to the development of tracking algorithm. We wanted to enrich our network output with temporal data and link each frame with the past ones to increase accuracy of the prediction and to develop a better understanding of the drive scene.

During testing on data recorded at Monza we highlighted that our filter robustly handles noisy detection even in an uncommon driving scenario like the one found at a racetrack. During the entire run the filter separated correctly the active and static classes and tracked the technical support van. The filter has been able to select the correct bounding box between the one present in the segmentation node output, estimate its position, dimensions and heading with an acceptable accuracy. Moreover, thanks to the class history saved in the filter we were able to correctly fix the wrong label assigned to the van in some frames.

We have demonstrated also the multi object tracking capabilities thanks to different test run recordings obtained in Milan with the Polimi Car. The results in this case have been validated visually since we do not have any ground truth available for our custom data.

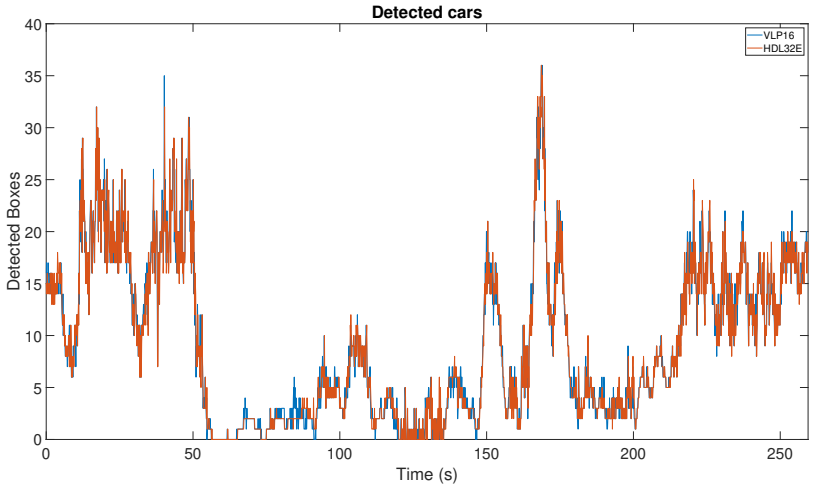


Figure 4.7: Performance comparison between detection of cars: VLP16 in blue vs HDL32E in orange

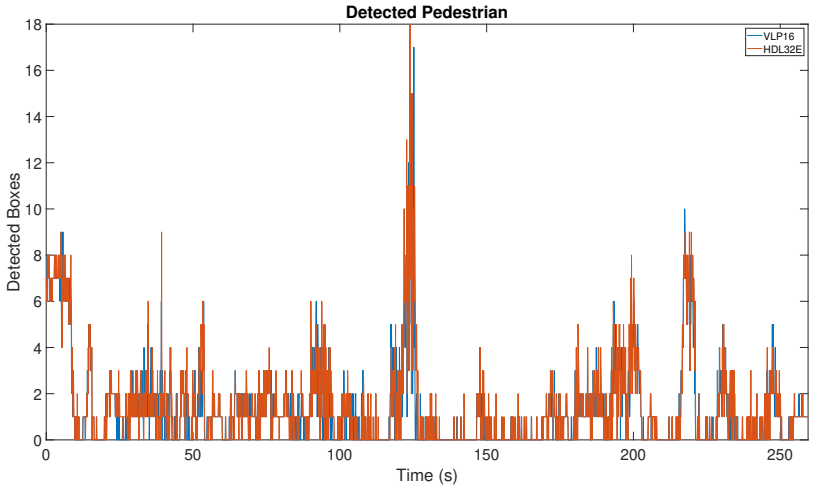
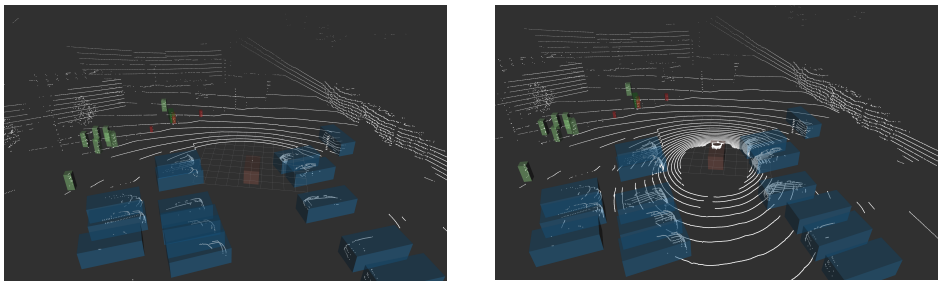


Figure 4.8: Performance comparison between detection of pedestrians: VLP16 in blue vs HDL32E in orange



(a) VLP16 sensor input **(b)** HDL32E sensor input

Figure 4.9: Segmentation result comparison: VLP16 vs HDL32E

4.9 ALGORITHM PERFORMANCE

A strict requirement of every autonomous driving system is time. A good performing non real time system is totally useless for automating the driving task. For this reason, our solution has been developed since the beginning as a real time system with strict time constraint. During the tests on the local machine system, we were able to detect and track objects in real time at an average frequency of 5Hz. Considering the sensor data flow that has a rate of 20Hz for the LiDAR we use on average four scans to obtain one view of the scenes. Of course, the number of layers and resolution of the LiDAR and the maximum number of detected objects in the scene affects those figures. Those results are encouraging since showed the potential of those type of CNN for fast segmentation and a consequent filtering and tracking stage. From a rapid analysis we identified that the main bottleneck of the system that limits the performance and does not let the system go faster is the pre-processing over the row point cloud. This step cycles through the entire point cloud different times, first to convert the data to a standard format, then to create voxels, finally to analyze the features inside each voxel.

The filter node on the other side demonstrated to be pretty efficient in terms of processing and the time it takes to be computed at each step is negligible when compared to the inference node. We imposed a limit of active trackable objects of 50 units. The execution time varies proportionally to the number of active trackers but is never the bottleneck of the system.

4.10 QUALITY OF THE ESTIMATION

In terms of accuracy in the run of Monza we have been able to check multiple parameters given the availability of the ground truth provided by the GPS position of the preceding vehicle and its dimensions. Filter performance are strictly bounded to the goodness of all the input parameters. The images below show the filter input data of the vehicle kinematics. At the beginning of the test we encountered noisy measurements that cause an instability of prediction. For this reason the estimation accuracy analysis has been done on data from Monza run from time 150 seconds till the end.
The run has been recorded using Polimi car in Autodromo Nazionale Monza. During all the run a support van moves in front of the vehicle to simulate the interaction with traffic. The GPS position of the van is recorded and used as ground truth.
The van, a Fiat Talento, has been detected for all the interval in a position that on average has an accuracy of plus and minus 0.5m in x and y directions in the worst case scenario. When all the input data are correct the position error drops significantly and the position is estimated with an accuracy of plus

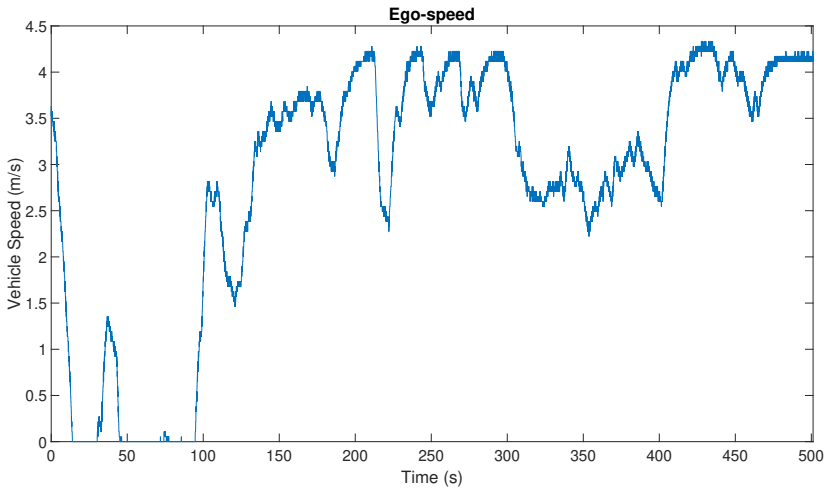


Figure 4.10: Ego-vehicle speed in meter per second of ego-vehicle during Monza test run

and minus 0.15m. The heading of the tracked object is estimated by the filter correctly. Given the fact that we do not have a ground truth we had only the opportunity to compare the heading with the van back side detection from LiDAR. We can say that the predicted heading is correct with an average error below 1 degree. The dimensions of the van has been detected by the network almost perfectly considered that all the input bounding boxes of the training set has more or less 0.15 m of slack at each side to give a safer estimation of the vehicle occupancy. On average the network predicted a vehicle width of 2.258 m, a length of 5.293 m and a height of 2.438 m where the van data-sheet reports a width of 1.956 m, a length of 5.100 m and height of 2.493 m. Overall estimations are quite accurate in terms of dimensions and in any case are over the vehicle size giving a safe view of the scene. We expect to have similar results on all the other classes segmented by the network. The filter performed as expected removing noisy detection and robustly identifying the correct position the preceding vehicle.

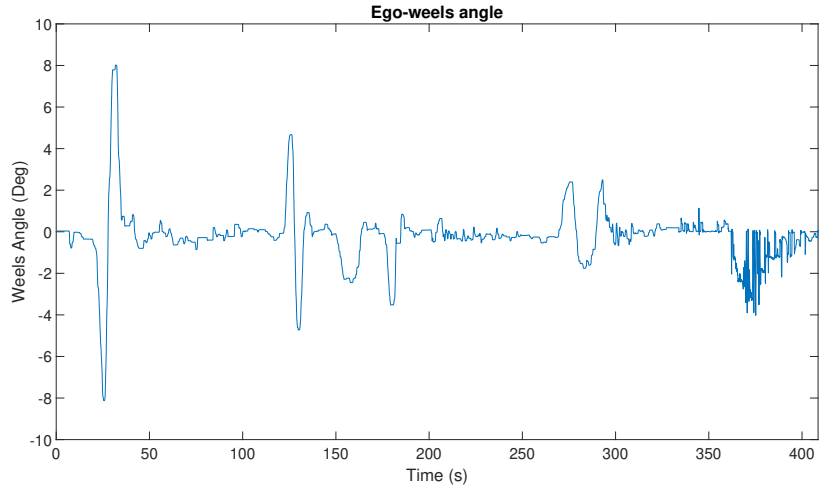


Figure 4.11: Ego-vehicle wheels angle in degrees of ego-vehicle during Monza test run

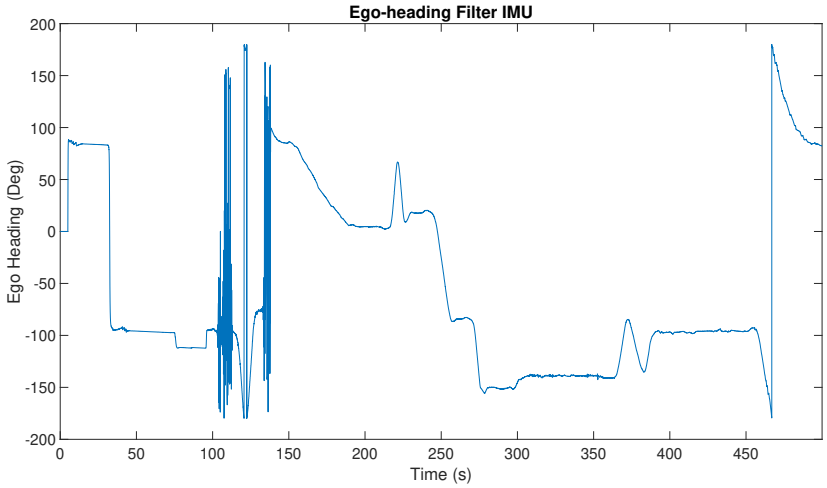


Figure 4.12: Ego-vehicle heading in degrees estimated by Filter IMU during Monza test run

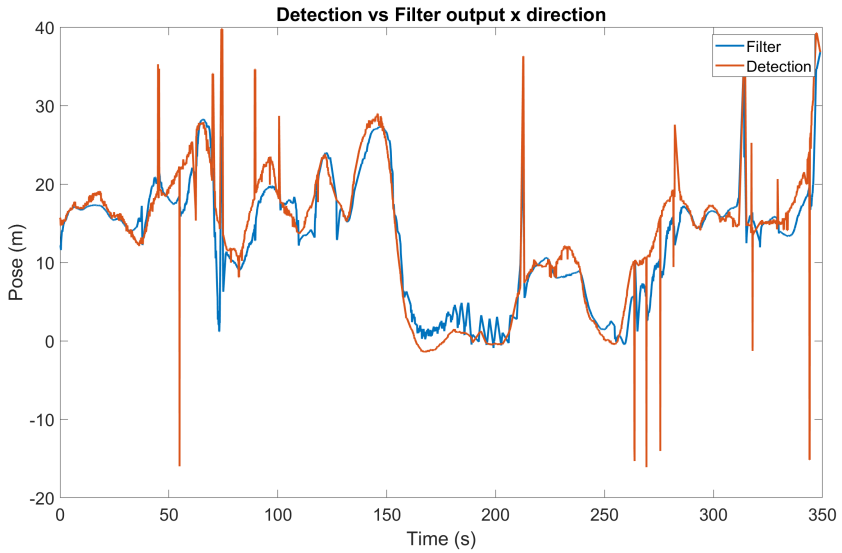


Figure 4.13: Filter output compared to sensor detection for x position. Spikes in GT graph are caused by missing or wrong GPS data.

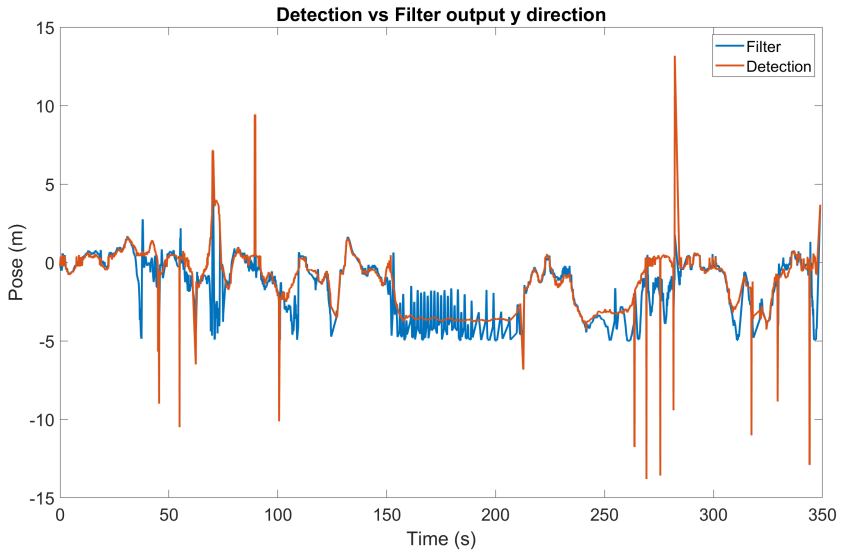


Figure 4.14: Filter output compared to sensor detection for y position. Spikes in GT graph are caused by missing or wrong GPS data.

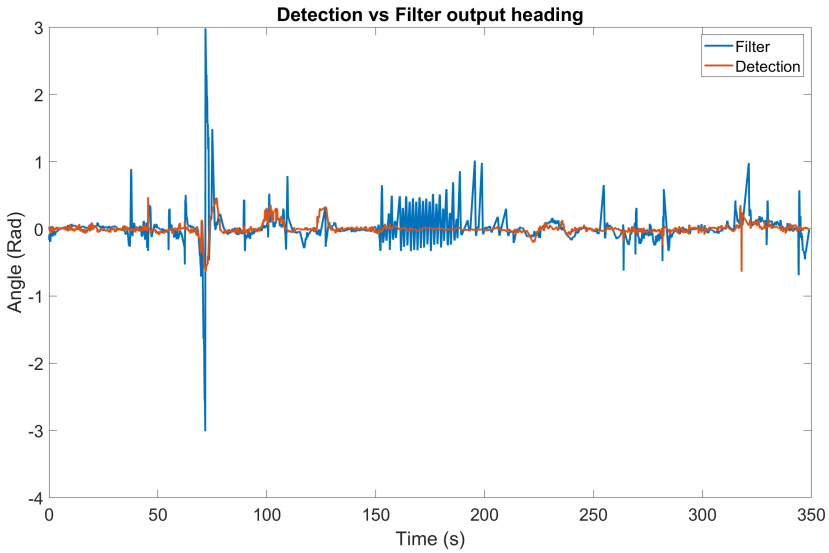


Figure 4.15: Filter output compared to sensor detection for heading. Spikes in GT graph are caused by missing or wrong GPS data.

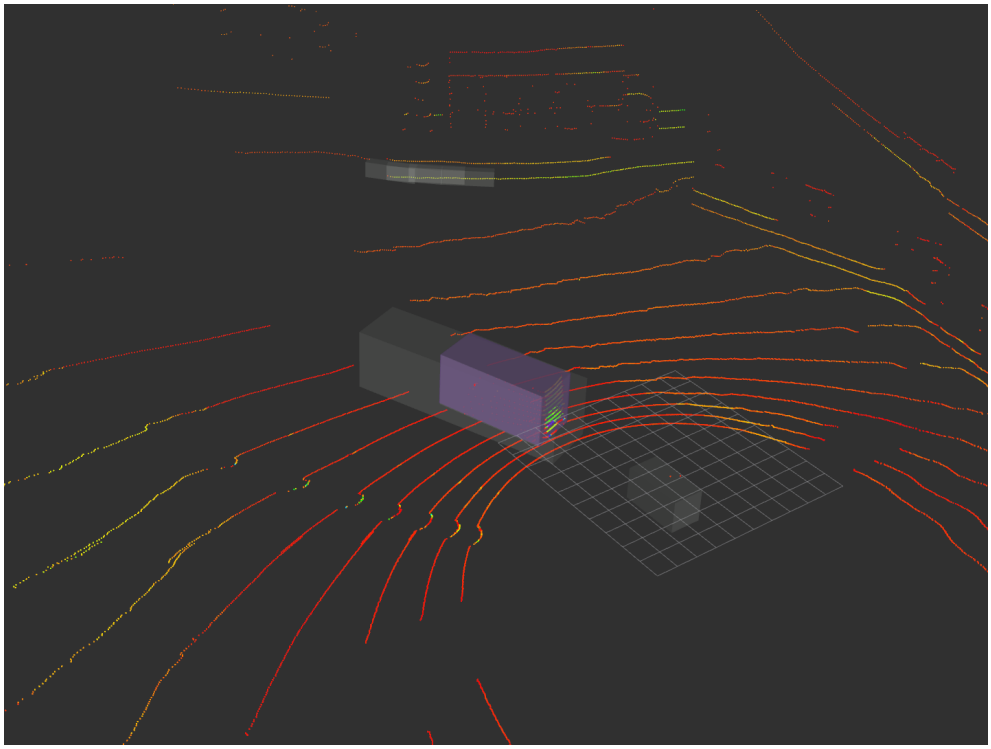


Figure 4.16: System in action during Monza test run. In purple the filter output corresponding with the preceding van and in light gray the detections coming form the segmentation network and the ego-vehicle

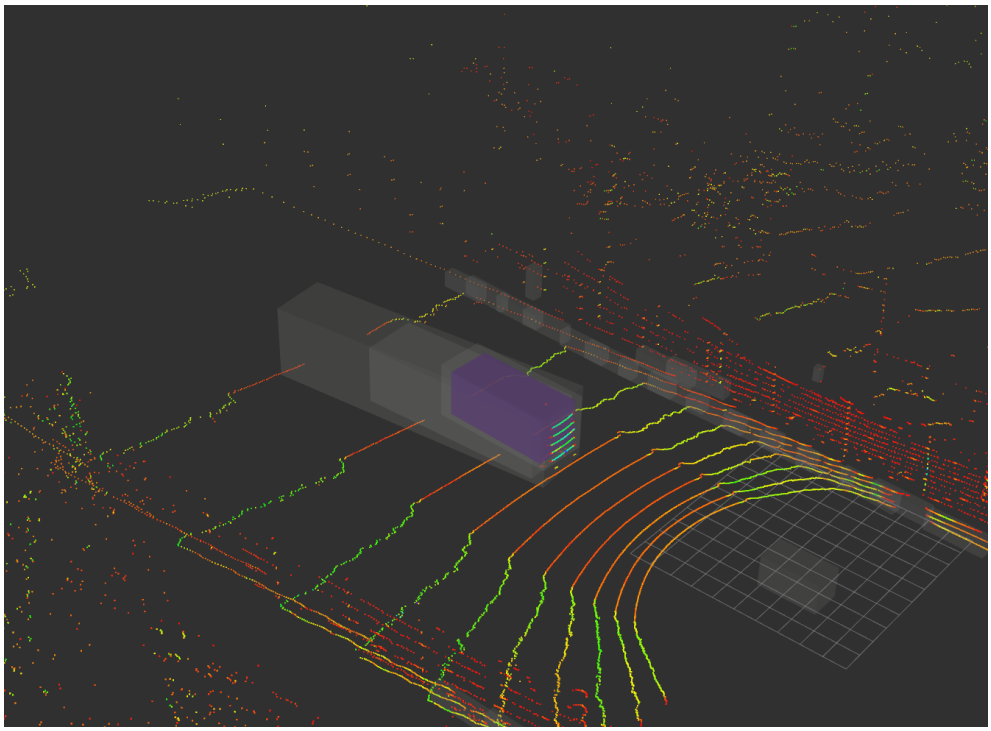


Figure 4.17: System in action during Monza test run. In purple the filter output corresponding with the preceding van and in light gray the detections coming form the segmentation network and the ego-vehicle

5 CONCLUSIONS

5.1 RESULTS DISCUSSION

The results obtained during this research work highlighted the possibility of leveraging LiDAR data not only from high level expensive sensors and not only for the automation of driving task but also for advanced safety systems implementation. We demonstrated a real time inference on different LiDAR sensor maintaining the same trained model for the network without loss of performance even when the resolution of the sensor is as low as 16 layers. Parallel to this we developed and tested the use of the information obtained by the network as position, heading, dimensions and class of the vehicles as input of an Extended Kalman Filter to further improve the final output quality. Finally, thanks to the filter we also obtained the trajectory and speed of the vehicles, useful parameters for scene assessment and understanding. We also tested different configuration of the system thanks to virtualization techniques guaranteeing real time performance with one GPU and as low as 4 cores on the Polimi server. Given the particular period we are traversing, working from home at the thesis gave me the opportunity also of testing the possibility of outsourcing computation of sensor data outside the vehicle on the Polimi Westworld server that can be easily handled by ROS and with enough data throughput can almost match the performance and latency obtained during local computation. In fact, we have seen that the network latency is hidden by a faster processing step thanks to more powerful hardware, of course a uncommon setup for autonomous vehicle but still an interesting test.

5.2 SYSTEM LIMITATIONS

As for all the systems, also this one presents some open questions and some limitations strictly related to the initial research query and to how the project evolved.

The overall results of the system are encouraging and show the feasibility of this approach for autonomous driving applications. As explained before the system has been developed as a path finding solution to demonstrate multi object real time tracking capabilities stating from raw LiDAR data. For this reason, the solution was developed using Python, a programming language particularly suitable for fast prototyping and rapid system iteration which, however, comes with computational overhead that is reflected in performance limitations. The system is capable of performing inference and tracking in real time on a high-performance PC with some limitation on the maximum input data rate of the LiDAR. To unlock the full performance it is required a rewriting of the main algorithm in C++, a programming language performance oriented and more suitable for embedded real time applications.

Another improvement that can greatly help in terms of performance is the use of upgraded hardware for inference. Latest graphics card models and dedicated autonomous vehicle computational hardware from NVIDIA support the Floating Point on 16 bit (FP16) data format that can speed up the inference process like demonstrated by those articles [58][59]. The system already supports this feature thanks to the CUDA libraries but has never been tested since we do not have the compatible hardware on hand.

As mentioned in the previous chapter we decided to work, during the development phase, into a Docker container to be able to fast deploy and test the system with different configurations and on different machines. This approach is particularly convenient for testing purposes, but it introduces a small overhead in terms of performance [60]. To remove the overhead in the system a native deployment is suggested for maximum performance.

5.3 SEGMENTATION NETWORK LIMITATIONS

For the Inference part of the system, the biggest limitation is the fact that the network has been trained with NuScenes dataset that, as explained, has been recorded mainly in Boston, US. The network is able to generalize pretty well on our data recorded in Milan but sometimes it struggles to identify the correct class of vehicles and road signs for couple of frames, problem that has been mitigated with the introduction of the Extended Kalman Filter but can be addressed also at a step prior retraining the network with a local, maybe European dataset. This can lead to a performance improvement since the American vehicles and signs are quite different in terms of sizes and shapes. For instance even if the bicycle class is present, it is rare to encounter given the different habits of Americans. The class is so rare that in testing the metric used resulted in an AP of 0 since no boxes with that class were encountered. Moreover, during a test run with the Polimi Car we have encountered a tricycle vehicle in use by Poste Italiane for mail delivery that in US does not even exist and the network sometimes identifies it as a pedestrian sometimes as a motorcycle and sometimes totally misses it and considers it as noise. Another example is the fact that small and mid-size vans, particularly common in Italy are sometimes identified with the class truck and sometimes as cars, problem that arises in the Monza test run where the Fiat Talento's label is wrong for a few instants. The filter covers the problem by fixing the misclassification but again the problem can be eliminated using a local dataset. Finally, Italian and in general the European version of traffic cones are usually missed by the network due to their smaller shape compared to their American counterpart, since they are half the height and present a different taper angle as you can see in the Figure 5.1.

Another problem that we encountered with our data is that the configuration of the Polimi Car, with the LiDAR slightly tilted forward of 6 degrees helps scan the front area of the vehicle but lowers the overall reachable height and returns data that are not usable for segmentation purposes in the back part of the vehicle. This explains why trucks and van slightly further than 40m ahead of the ego vehicle are sometimes identified and sometimes not since the network does not find the top edge of the vehicle shape even if, from the sensor data-sheet, the sensor is rated for up to 100m detection distance. Moreover this tilting angle confuses the network in some situation and leads to misclassification of vehicles due to different pattern of the data like with road barriers that are identified not planar with the ground but with a tilt angle. This problem again is related to our setup and can be solved mounting the sensor in level with the ground. The network is trained with a similar configuration but with a 32 Layers LiDAR that only increases the maximum detection distance up to 120m. Experiments were carried out

(a) Italian Traffic Cones **(b)** US Traffic Cones

Figure 5.1: Comparison between Italian and US Traffic Cones

by extracting 16 Layer information from the training dataset, reproducing the Polimi Car LiDAR sensor output but in no tilted configuration and the problem disappeared resulting in more stable and robust results.

5.4 FILTER LIMITATIONS

Concerning the developed Extended Kalman Filter,it works in tracking multiple objects and predicting correctly the next position of the agent in the scene within a good accuracy. It shows some limitations when sudden changes to the system happen, like when the tracked vehicle makes a sharp turn and the predicted bounding box tends to deviate a bit in respect to the vehicle real position. Moreover due to the noise data sometimes the filter shows unstable behaviours. Thanks to the fact that we have classes of each agent as network output a further development can implement ad hoc kinematic model for each class that keeps in consideration the different kinematic constraint of the motion for vehicles, bicycles and motorcycles, trucks and pedestrians substituting the assumption and kinematic model simplifications we have done here.

Moreover the filter is strictly bounded to the presence of accurate GPS data since a fast and simple approach for computing the heading angle has been implemented. We encountered some problems of GPS accuracy and overall stability in urban scenarios where high buildings cause a loss of signal and a consequent worsening of filter accuracy. This problem can be addressed by employing more refined solutions for the heading estimation that keep in consideration also the IMU data like we have done during the testing with Monza test data or by using dedicated sensor for heading measurement.

Another improvement to further ameliorate the final results can be to incorporate in the filter the road map or a road line detection algorithm to remove false positive outside the road scene; sometime in fact happen that the network recognize some features of building like edges and walls as big trucks and publish boxes outside the road and outside the area of interest of the system.

5.5 FUTURE WORKS

This work is only the beginning and multiple expansion can be implemented to enhance the results. First of all a retraining on an European dataset is required, a dataset that at the moment of writing does not exist in a form that we need for training; the one available are far behind NuScenes both in dimensions, quality of data and completeness. This alone can greatly benefit the accuracy of the classifier in the prediction of bounding boxes and the corresponding classes. On top of a regional model, the development of more refined Extended Kalman Filter for each class can be helpful for a more stable and precise outcome particularly for the tracking of pedestrian, the only class that does not have any kinematic constraint. For what concerns possible improvements based on the path of each vehicle can be predicted a possible future trajectory that is fundamental for driving task automation and in particular to the motion planner to choose the best path to be followed while avoiding other agents and reach the final goal.

To improve system accuracy during the segmentation step an interesting idea is to combine multiple consequent scans from the LiDAR into one higher density point cloud and correct for the misalignments between frames by using odometry data.

(a) LiDAR installed in Henderson, NV **(b)** System Rendering

Figure 5.2: Road Crossing Monitoring

5.6 POSSIBLE APPLICATIONS

This work can be applied to multiple fields and applications, and has been developed from the beginning to be independent and easy to be transferred from a system to another, from a sensor to a newer one, from an application to a different one. It can be integrated in an autonomous vehicle pared with cameras and radars to provide full self driving capabilities to a vehicle as the Polimi Car or can work stand alone on a public transportation vehicle to assist and prevent accidents from happening especially in complex and chaotic environments in a dense populated area like Milan. Moreover it can be deployed on a fixed installation, alone or in pair with a security camera, to monitor and track vehicles and pedestrian in a road cross for security and monitoring purposes preserving the privacy of the citizen since LiDAR data are not enough to provide identification or plates numbers. Velodyne has started just recently in US, in partner with University of Nevada and Qualcomm, a pilot project that involves the deployment of LiDARs to monitor traffic and pedestrian and test out new concepts for Smart Cities of the future.

BIBLIOGRAPHY

[1] European Union. "Road Safety 2018." In: (2018), p. 45. eprint: https://doi.org/10.2832/39063 (cit. on p. 2).

[2] European Union. "Report on Railway Safety and Interoperability in the EU." In: (2020), pp. 28–30. eprint: doi:10.2821/30980 (cit. on p. 2).

[3] EASA. "Annual Safety Review." In: (2020), pp. 26–27. eprint: https://doi.org/10.2822/147804 (cit. on p. 2).

[4] SAE. "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles." In: (2016), pp. 21–23. eprint: https://doi.org/10.4271/J3016_201806 (cit. on p. 6).

[5] OnePoll. "Americans spend 18 days in their car per year, forge close bonds with a vehicle." In: (2019), p. 1. eprint: https://www.thecarconnection.com/news/1122782_study-americans-spend-18-days-in-their-car-per-year-forge-close-bonds-with-a-vehicle (cit. on p. 2).

[6] DARPA. "DARPA autonomous vehicles challenges archive." In: (2007), p. 1. eprint: https://archive.darpa.mil/grandchallenge/overview.html (cit. on p. 9).

[7] Lyft. "Self-Driving Safety Report." In: (2020), pp. 1–43. eprint: https://self-driving.lyft.com/wp-content/uploads/2020/06/Safety_Report_2020.pdf (cit. on p. 20).

[8] Google. "ntroducing the 5th-generation Waymo Driver." In: (2020), pp. 1–43. eprint: https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html (cit. on p. 14).

[9] W. Rahiman and Z. Zainal. "An overview of development GPS navigation for autonomous car." In: (2013), pp. 1112–1118 (cit. on p. 39).

[10] NASA. "Mars 2020 Mission Overview." In: (2020), p. 142. eprint: https://doi.org/10.1007/s11214-020-00762-y (cit. on p. 26).

[11] Nasser Kehtarnavaz, Norman C. Griswold, and J. K. Eem. "Comparison of mono- and stereo-camera systems for autonomous vehicle tracking." In: 1468 (1991). Ed. by Mohan M. Trivedi, pp. 467–478 (cit. on p. 31).

[12] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schu-

berth. "A2D2: Audi Autonomous Driving Dataset." In: (2020). arXiv: 2004.06320 [cs.CV] (cit. on p. 49).

[13] A Geiger, P Lenz, C Stiller, and R Urtasun. "Vision meets robotics: The KITTI dataset." In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. eprint: https://doi.org/10.1177/0278364913491297 (cit. on p. 49).

[14] Liat Sless, Bat El Shlomo, Gilad Cohen, and Shaul Oron. "Road Scene Understanding by Occupancy Grid Learning from Sparse Radar Clusters using Semantic Segmentation." In: (2019) (cit. on p. 44).

[15] R. Yagfarov, M. Ivanou, and I. Afanasyev. "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth." In: (2018), pp. 1979–1983 (cit. on p. 44).

[16] Y. Li, D. Duan, C. Chen, X. Cheng, and L. Yang. "Occupancy Grid Map Formation and Fusion in Cooperative Autonomous Vehicle Sensing (Invited Paper)." In: (2018), pp. 204–209 (cit. on p. 44).

[17] H. Xue, H. Fu, R. Ren, T. Wu, and B. Dai. "Real-time 3D Grid Map Building for Autonomous Driving in Dynamic Environment." In: (2019), pp. 40–45 (cit. on p. 45).

[18] A. A. S. Souza and L. M. G. Gonçalves. "2.5-Dimensional Grid Mapping from Stereo Vision for Robotic Navigation." In: (2012), pp. 39–44 (cit. on p. 45).

[19] M. Himmelsbach, F. v. Hundelshausen, and H. . Wuensche. "Fast segmentation of 3D point clouds for ground vehicles." In: (2010), pp. 560–565 (cit. on p. 45).

[20] L. Chen, J. Yang, and H. Kong. "Lidar-histogram for fast road and obstacle detection." In: (2017), pp. 1343–1348 (cit. on p. 45).

[21] Yuan Wang, Tianyue Shi, Peng Yun, Lei Tai, and Ming Liu. "PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud." In: (2018). arXiv: 1807.06288 [cs.CV] (cit. on pp. 45, 46).

[22] B. Yang, W. Luo, and R. Urtasun. "PIXOR: Real-time 3D Object Detection from Point Clouds." In: (2018), pp. 7652–7660 (cit. on p. 45).

[23] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. "SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud." In: (2017). arXiv: 1710.07368 [cs.CV] (cit. on pp. 45, 46).

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: (2015). arXiv: 1411.4038 [cs.CV] (cit. on p. 45).

[25] C. Zhang, W. Luo, and R. Urtasun. "Efficient Convolutions for Real-Time Semantic Segmentation of 3D Point Clouds." In: (2018), pp. 399–408 (cit. on p. 45).

[26] S. Alireza Fatemi Jahromi, A. Asghar Khani, H. Otroshi Shahreza, M. Soleymani Baghshah, and H. Behroozi. "A Deep Learning Framework for Viable Tumor Burden Estimation." In: (2020), pp. 1–7.

[27] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection." In: (2017). arXiv: 1711.06396 [cs.CV] (cit. on pp. 45, 46).

[28] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In: (2017). arXiv: 1612.00593 [cs.CV] (cit. on pp. 45, 46).

[29] Rudra P K Poudel, Stephan Liwicki, and Roberto Cipolla. "Fast-SCNN: Fast Semantic Segmentation Network." In: (2019). arXiv: 1902.04502 [cs.CV] (cit. on p. 45).

[30] Y. Zeng, Y. Hu, S. Liu, J. Ye, Y. Han, X. Li, and N. Sun. "RT3D: Real-Time 3-D Vehicle Detection in LiDAR Point Cloud for Autonomous Driving." In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3434–3440 (cit. on p. 46).

[31] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. "Complex-YOLO: Real-time 3D Object Detection on Point Clouds." In: (2018). arXiv: 1803.06199 [cs.CV] (cit. on p. 46).

[32] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. "Fast LIDAR-based Road Detection Using Fully Convolutional Neural Networks." In: (2017). arXiv: 1703.03613 [cs.CV] (cit. on p. 46).

[33] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. "SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud." In: (2018). arXiv: 1809.08495 [cs.CV] (cit. on pp. 45, 46).

[34] Yan Yan, Yuxing Mao, and Bo Li. "SECOND: Sparsely Embedded Convolutional Detection." In: *Sensors* 18.10 (2018). ISSN: 1424-8220 (cit. on p. 45).

[35] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. "Frustum PointNets for 3D Object Detection from RGB-D Data." In: (2018). arXiv: 1711.08488 [cs.CV] (cit. on p. 46).

[36] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. "PointPillars: Fast Encoders for Object Detection from Point Clouds." In: (2019). arXiv: 1812.05784 [cs.LG] (cit. on p. 45).

[37] Dominic Wang and Ingmar Posner. "Voting for Voting in Online Point Cloud Object Detection." In: (July 2015) (cit. on p. 45).

[38] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. "Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks." In: (2017). arXiv: 1609.06666 [cs.RO] (cit. on p. 45).

[39] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: (2017). arXiv: 1706.02413 [cs.CV] (cit. on p. 45).

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: 28 (2015). Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (cit. on p. 46).

[41] S. Song and J. Xiao. "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images." In: (2016), pp. 808–816 (cit. on p. 46).

[42] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection." In: (2018). arXiv: 1708.02002 [cs.CV] (cit. on p. 46).

[43] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. "Multi-View 3D Object Detection Network for Autonomous Driving." In: (2017). arXiv: 1611.07759 [cs.CV] (cit. on p. 46).

[44] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. "PointCNN: Convolution On $\mathcal{X}$-Transformed Points." In: (2018). arXiv: 1801.07791 [cs.CV] (cit. on p. 46).

[45] Benjamin Graham. "Spatially-sparse convolutional neural networks." In: (2014). arXiv: 1409.6070 [cs.CV] (cit. on p. 46).

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: (2016). arXiv: 1506.02640 [cs.CV] (cit. on p. 46).

[47] Chunlei Yi, Kunfan Zhang, and Nengling Peng. "A multi-sensor fusion and object tracking algorithm for self-driving vehicles." In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 233 (Aug. 2019), pp. 2293–2300 (cit. on p. 47).

[48] K. Jo, M. Lee, J. Kim, and M. Sunwoo. "Tracking and Behavior Reasoning of Moving Vehicles Based on Roadway Geometry Constraints." In: *IEEE Transactions on Intelligent Transportation Systems* 18.2 (2017), pp. 460–476 (cit. on p. 47).

[49] Tobias Klinger, F. Rottensteiner, and C. Heipke. "PROBABILISTIC MULTI-PERSON TRACKING USING DYNAMIC BAYES NETWORKS." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2015), pp. 435–442 (cit. on p. 47).

[50] Babak Shahian Jahromi, Theja Tulabandhula, and Sabri Cetin. "Real-Time Hybrid Multi-Sensor Fusion Framework for Perception in Autonomous Vehicles." In: *Sensors* 19.20 (2019). ISSN: 1424-8220 (cit. on p. 47).

[51] H. Cho, Y. Seo, B. V. K. V. Kumar, and R. R. Rajkumar. "A multi-sensor fusion system for moving object detection and tracking in urban driving environments." In: (2014), pp. 1836–1843 (cit. on p. 47).

[52] Lyudmila Mihaylova, Avishy Y. Carmi, François Septier, Amadou Gning, Sze Kim Pang, and Simon Godsill. "Overview of Bayesian sequential Monte Carlo methods for group and extended object tracking." In: *Digital Signal Processing* 25 (2014), pp. 1–16. ISSN: 1051-2004 (cit. on p. 47).

[53] S. Reuter, B. Wilking, J. Wiest, M. Munz, and K. Dietmayer. "Real-Time Multi-Object Tracking using Random Finite Sets." In: *IEEE Transactions on Aerospace and Electronic Systems* 49.4 (2013), pp. 2666–2678 (cit. on p. 47).

[54] Daniel Simon. "Optimal State Estimation: Kalman, H, and Nonlinear Approaches." In: (Jan. 2006) (cit. on p. 47).

[55] S. J. Julier and J. K. Uhlmann. "Unscented filtering and nonlinear estimation." In: *Proceedings of the IEEE* 92.3 (2004), pp. 401–422 (cit. on p. 47).

[56] R. Juang and P. Burlina. "Comparative performance evaluation of GM-PHD filter in clutter." In: (2009), pp. 1195–1202 (cit. on p. 47).

[57] CH: International Organization for Standardization Standard. Geneva. "Road vehicles—Vehicle dynamics and road-holding ability—Vocabulary." In: (Jan. 2018), p. 42 (cit. on p. 55).

[58] "Titan RTX: Quality time with the top Turing GPU." In: (Jan. 2019). eprint: https://blog.slavv.com/titan-rtx-quality-time-with-the-top-turing-gpu-fe110232a28e (cit. on p. 92).

[59] "FastAI Mixed Precision training comparisons." In: (Jan. 2019). eprint: https://hackernoon.com/rtx-2080ti-vs-gtx-1080ti-fastai-mixed-precision-training-comparisons-on-cifar-100-761d8f615d7f (cit. on p. 92).

[60] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. "An updated performance comparison of virtual machines and Linux containers." In: (2015), pp. 171–172 (cit. on p. 92).

[61] S. Mentasti and M. Matteucci. "Multi-layer occupancy grid mapping for autonomous vehicles navigation." In: (2019), pp. 1–6 (cit. on p. 42).

[62] Mattia Bersani, Simone Mentasti, Pragyan Dahal, Stefano Arrigoni, Michele Vignati, Federico Cheli, and Matteo Matteucci. "An integrated algorithm for ego-vehicle and obstacles state estimation for autonomous driving." In: *Robotics and Autonomous Systems* 139 (2021), p. 103662. ISSN: 0921-8890 (cit. on p. 62).