Universidad de Alcalá
Escuela Politécnica Superior

**Máster Universitario en Ingeniería Industrial**

**Trabajo Fin de Máster**

# Deep Learning based 3D Object Detection for Automotive Radar and Camera Fusion



**Author:** Santiago Montiel Marín

**Advisor:** Ángel Llamazares Llamazares

**Co-advisor:** Luis Miguel Bergasa Pascual

2023

# UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

**Máster Universitario en Ingeniería Industrial**

**Trabajo Fin de Máster**

**Deep Learning based 3D Object Detection for Automotive Radar and Camera Fusion**

Author: Santiago Montiel Marín

Advisor: Ángel Llamazares Llamazares

Co-advisor: Luis Miguel Bergasa Pascual

**Tribunal:**

**President:** Noelia Hernández Parra

**1st Vocal:** Cristina Losada Gutiérrez

**2nd Vocal:** Ángel Llamazares Llamazares

Deposit date: September 20th, 2023

**To my beloved parents, those who believe in me when I don't even believe in myself. . .**

*"And those who were seen dancing were thought to be insane*
*by those who could not hear the music."*
Friedrich Nietzsche

# Acknowledgements

*"Veni, vidi, vici."* Es lo que dijo el general y cónsul romano Julio César tras su contundente victoria sobre los Ejércitos del Ponto en la batalla de Zela. Esta gran figura, con quien tengo el placer de compartir día de nacimiento, dio una frase para la posteridad que se utiliza para destacar la rapidez con la que alguien consigue sus éxitos. Una frase de la Roma clásica que encaja perfectamente con la fugacidad, instantaneidad y ansia de éxitos que vivimos en nuestros contemporáneos días. Sin embargo, querido Julio César, quiero decirte que no estoy de acuerdo contigo.

Quiero decirte que el éxito o la consecución de las metas de uno mismo viene tras recorrer un camino lento y pedregoso, de esfuerzo, lleno de dudas y altibajos. Porque como dijo Séneca: *"Per aspera ad astra"*, no hay camino fácil de la Tierra a las estrellas. Y como este camino no es fácil, es recomendable hacerlo acompañados. Acompañados y rodeados de personas que aportan en diferentes aspectos de nuestra vida y que hacen que seas la persona que eres en el día a día. Así que ahora es mi turno de mostrar el agradecimiento que merecidamente les debo a las personas de mi vida.

A mis tutores, Ángel Llamazares y Luis Miguel Bergasa. Gracias por la oportunidad que se me brindó hace tres años de formar parte del grupo de investigación RobeSafe. Día a día hacéis una inversión a fondo perdido en mí (así como la investigación misma), tanto en el plano profesional como en el personal, que me permite ser mejor que el día anterior en ambas facetas. Gracias por confiar en mí y espero poder devolvéroslo de vuelta.

A mi profesor del instituto, Manuel Gálvez, y a mi mentor en el equipo de robótica de la Universidad, Ángel Álvarez. El primero tuvo la culpa de que me iniciase en la robótica y el segundo de que la descubriese con profundidad. Gracias a vuestras influencias he encontrado el campo donde deseo desarrollarme profesionalmente.

A mis amigos de la Universidad, en orden alfabético, Navil Abdeselam, Ana Belén Bartolomé, Andrés Chavarrias, Guillermo Madariaga, César Murciego, Javier Ortiz, Javier Quintanar y Fabio Sánchez. Sois auténticos apoyos incondicionales e indispensables, motores de ideas y proyectos. Os admiro mucho y de mayor quiero ser un poco más como vosotros. También a mis amigos de la infancia, Javier Arenas, Diego Hidalgo, Alejandro Martínez, María Perea y Laura Suárez. Gracias porque aunque nos veamos tan poco, la amistad se mantiene como siempre.

A mis compañeros de laboratorio, a los más veteranos: Felipe Arango, Carlos Gómez, Javier Araluce, Rodrigo Gutiérrez y Alejandro Díaz; y a los más jóvenes: Miguel Antunes, Pablo

Pardo, Navil y Fabio (otra vez vosotros dos, sí, gracias por dos). Espero haber aprendido lo máximo de los primeros para poder trasladárselo a los segundos. Gracias por hacer del sitio de trabajo un sitio agradable y divertido, donde todos aprendemos de todos y nos podemos desarrollar personalmente.

A mi familia: a mis abuelos, a mis tíos y primos. Siento admiración por todos vosotros y sois parte de mis éxitos. Porque me cuidáis y confíais en mí, y de todos vosotros aprendo algo que me ayuda a afrontar mis retos.

A mis padres, Santiago Montiel y María Dolores Marín. Las personas que soportan mi mundo cuando este se tambalea. Un auténtico motor de fuerza y esperanza cuando el viento sopla en contra. Porque han creado un ecosistema en el que puedo desarrollarme personalmente y ser totalmente sincero emocionalmente con ellos. El sitio al que acudo cuando las dudas surgen. Ya lo dijo la dedicatoria de mi Trabajo Fin de Grado: *"a mis amados padres, porque sin ellos hubiera sido imposible"*. La dedicatoria de este Trabajo Fin de Máster insiste: *"a mis amados padres, aquellos que confían en mí aún cuando ni yo mismo lo hago"*. Podría dedicarles infinitas frases, pero no serían suficientes para agradecérselo.

Por último, a ti, lector de este libro. Porque espero que el fruto de horas de trabajo que ha sido plasmado aquí te sirva para avanzar en tu carrera profesional o académica. He tratado este texto con las mejores formas que tengo y espero que te enriquezcas personalmente con esta lectura.

No estoy seguro de que haya conseguido el éxito con la terminación de este libro, porque solo es un paso más en el largo camino que queda por delante. De lo que sí estoy seguro, es de que gracias a todas las personas que he mencionado es por lo que sigo teniendo fuerzas y ganas de seguir caminando. Además, qué mejores pies para caminar que los pies de un *hobbit*. Gracias de corazón.

# Resumen

La percepción en el dominio de los vehículos autónomos es una disciplina clave para lograr la automatización de los Sistemas Inteligentes de Transporte. Por ello, este Trabajo Fin de Máster tiene como objetivo el desarrollo de una técnica de fusión sensorial para RADAR y cámara que permita crear una representación del entorno enriquecida para la Detección de Objetos 3D mediante algoritmos Deep Learning. Para ello, se parte de la idea de PointPainting [1] y se adapta a un sensor en auge, el RADAR 3+1D, donde nube de puntos RADAR e información semántica de la cámara son agregadas para generar una representación enriquecida del entorno.

**Palabras clave:** Conducción autónoma, fusión sensorial, RADAR, detección de objetos, PointPainting.

# Abstract

Perception in the domain of autonomous vehicles is a key discipline to achieve the automation of Intelligent Transport Systems. Therefore, this Master Thesis aims to develop a sensor fusion technique for RADAR and camera to create an enriched representation of the environment for 3D Object Detection using Deep Learning algorithms. To this end, the idea of PointPainting [1] is used as a starting point and is adapted to a growing sensor, the 3+1D RADAR, in which the radar point cloud is aggregated with the semantic information from the camera.

**Keywords:** Autonomous Driving, Sensor Fusion, RADAR, Object Detection, PointPainting.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| ACC | Adaptive Cruise Control. |
| AD | Autonomous Driving. |
| ADAS | Advanced Driver Assistance System. |
| ADS | Autonomous Driving Stack. |
| AEB | Assistance for Emergency Braking. |
| AI | Artificial Intelligence. |
| AOS | Average Orientation Similarity. |
| AP | Average Precision. |
| AUC | Area Under the Curve. |
| AV | Autonomous Vehicle. |
| BEV | Bird's Eye View. |
| CFAR | Constant False Alarm Rate. |
| CM | Confusion Matrix. |
| CNN | Convolutional Neural Networks. |
| CSP | Cross Spatial Partial connections. |
| CV | Computer Vision. |
| DBSCAN | Density-based Spatial Clustering for Applications with Noise. |
| DL | Deep Learning. |
| DNN | Deep Neural Networks. |
| DPM | Deformable Part-based Model. |
| DS | Driving Score. |
| EDA | Exploratory Data Analysis. |
| EKF | Extended Kalman Filter. |
| FCN | Fully Convolutional Networks. |
| FCOS | Fully Convolutional One-Stage. |
| FFT | Fast Fourier Transform. |
| FMCW | Frequency Modulated Continous Wave. |

| | |
|---|---|
| FN | False Negative. |
| FP | False Positive. |
| FPN | Feature Pyramid Networks. |
| GNN | Graph Neural Networks. |
| GNSS | Global Navigation Satellite System. |
| GPS | Global Positioning System. |
| HD Map | High-Definition Map. |
| HOG | Histogram of Oriented Gradients. |
| IMU | Inertial Measurement Unit. |
| IoU | Intersection over Union. |
| ITS | Intelligent Transportation System. |
| KPConv | Kernel Point-based Convolution. |
| LiDAR | Light Detection and Ranging. |
| LLM | Large Language Models. |
| MAE | Mean Absolute Error. |
| mAP | Mean Average Precision. |
| ML | Machine Learning. |
| MLP | Multi Layer Perceptron. |
| MPC | Model Predictive Controllers. |
| NDS | nuScenes Detection Score. |
| NLP | Natural Language Processing. |
| NMS | Non Maximum Suppression. |
| NN | Neural Networks. |
| PCA | Principal Component Analysis. |
| PCD | Point Cloud. |
| PFE | Pillar Feature Extractor. |
| R-CNN | Region Based Convolutional Neural Networks. |
| RADAR | Radio Detection and Ranging. |
| RCS | Radar Cross Section. |
| RPN | Region Proposal Networks. |
| SAE | Society of Automotive Engineers. |

SGD Stochastic Gradient Descent.
SOTA State of the Art.
SPP Spatial Pyramid Pooling.
SSD Single-Shot Detector.
STFT Short Time Fourier Transform.
SVD Singular Value Decomposition.
SVM Support Vector Machine.

TN True Negative.
TOF Time of Flight.
TP True Positive.

VRU Vulnerable Road Users.

YOLO You Only Look Once.

# Chapter 1

# Introduction

*There is no easy way from Earth to the stars.*

Lucius Annaeus Seneca, the Younger

## 1.1. Motivation

The advent of Deep Learning (DL) techniques during the last decade has meant a technological revolution. The disciplines that have embraced DL as their main tool to evolve are advancing exponentially, fulfilling achievements that looked years away, in a matter of months. Faced with this situation, makers, researchers, and companies from all over the world have put their effort and resources into these topics. In the days we are living, a maker can publish an open-source project that achieves 50,000 stars on GitHub in days; academic conferences and journals are receiving thousands of research proposals per call; and, there is no technological company without a research and development team that produces papers or novel products based on DL. Even industry is carrying out some of the most important advances in the field such as the case of Large Language Models (LLM), whose foundational models and baselines are being developed by companies such as OpenAI or Meta. This phenomenon is known as the "industrialization of research".

Out of the disciplines that have benefited from these advances, Robotics, and Autonomous Driving (AD) are noteworthy and this work will be framed into these two fields, and more specifically, within the perception and scene understanding of the environment using the onboard sensors. In fact, the state of AD has evolved from being a futuristic and technically unachievable dream to a new reality that may come within this or the next decade. Driven by the innovative capacity of automakers, technological companies, and academic researchers, Artificial Intelligence (AI) has pushed the boundaries of sensor technologies and fusion, Computer Vision (CV), and computing platforms. The introduction of these technologies in the automotive sector has led to an authentical revolution of the Intelligent Transportation System (ITS) as we know them.

## 1.2. A brief introduction to Autonomous Driving

A self-driving car or Autonomous Vehicle (AV) is an ITS that is capable of navigating through the road system with little or no human intervention. To achieve autonomous navigation in a safe and efficient way, it is necessary to perceive the environment, localize our AV with respect to an environment or a map, carry out cognitive processes to make a decision and interact with the physical devices to take action. All these tasks constitute the Autonomous Driving Stack (ADS) and will be further explained later. Continuous closed-loop execution is required to perform autonomous navigation since the environment that surrounds a vehicle is populated with multiple and dynamic road users (such as other vehicles, pedestrians, and cyclists) and different signs and traffic lights that regulate how to circulate in each specific scenario. As the environment is constantly changing, the system must reevaluate its scene understanding to ensure or abort the decision that was taken in previous cycles by the cognitive stage, and lastly, perform the corresponding actions.

The development of this technology is supported by a series of arguments with which we can conclude that its final output is highly beneficial for our society. Most of the key advantages of Autonomous Driving (AD) come from the elimination of the human factor in the driving task. Although traffic accidents in Spain have fallen over the last decade to historically low levels, which indicates that drivers are becoming more responsible, human factors are still behind 80-90% of the accidents. Driving patterns will be improved and consumption will be reduced in terms of energy and time.

Elaborating on these arguments, the following main points can be highlighted:

- **Elimination of the human factor**: as previously mentioned, non-desired human behaviours are the cause of the majority of traffic accidents. In fact, due to the development of smartphones and mobile technologies during the last decade, accidents caused by being distracted because of using an electronic device have become the top cause of accidents. Other distractions are caused by talking to the occupants of the vehicle or consuming food and drink while driving. In addition, the consumption of alcohol or narcotic substances reduces the driver's cognitive abilities. Lastly, adapting driving to the rules of the road is a decision that the driver must make consciously. When this is not done and they are driving at high speeds or in an aggressive style, the possibility of having an accident is increased. All these scenarios will be completely forgotten in a scenario of full self-driving, so there will be a major reduction in the number of accidents.

- **Optimized patterns of driving**: it could be said that AD is committed to the environment as the development of newer planning algorithms allows the adoption of optimal driving patterns in terms of gas emissions and energy consumption. Therefore, these patterns would avoid energy-consuming manoeuvres such as sudden acceleration, excessive braking or excessive speed. Also, the ITS revolution is accompanied by the advent of new energy sources to power vehicles, such as the electrization or hydrogenation of these systems.

- **Increase in traffic efficiency**: the ability to plan and adapt optimally according to traffic rules and the environment will enable the vehicle to choose the optimal decision in terms of speed and time. For example, traffic jams appear in conditions of high-volume traffic and anomalous events, such as an accident or lane closures. This provokes the drivers to perform small stops in chains, which end up in stops, trying to keep a safe distance from the next vehicle. If we could handle traffic jams in an optimal way, we could choose the optimal braking that maximizes our speed while keeping a safe distance, minimizing the number of stop-and-go events. Also, we could avoid replanning the route in real-time. Even nowadays we can plan our routes depending on the traffic information available and replan it in real-time with constant updates if the conditions of the traffic flow vary.

- **Inclusive model of mobility**: the existence of AVs will increase the possibilities of transportation for certain sectors of the population who are not able to drive. Potentially, this set includes people with disabilities such as blindness people, with reduced mobility or ageing people, being the latter a sector in constant growth in the most developed countries. All these people will have access to unrestricted transportation without needing any external support.

However, it is still necessary to beat technical challenges to adopt self-driving technologies in a mass-wide spread:

- **Critical safety requirements**: unfortunately, the consequences of a failure in an AD system can result in fatal events, such as the death of occupants or other road users. As a result, these systems need to be further developed to ensure their capability to navigate complex, high-dynamic and out-of-distribution traffic scenarios while effectively avoiding accidents.

- **Cost reductions**: at the time being, the solutions proposed by companies are vehicles that include a wide range of high-technology sensors, such as LiDARs, RADARs and cameras. The inclusion of these sensors among vehicle components greatly increases the manufacturing cost and, consequently, the sale price in the market. If Fully AD is to be achieved as a transportation model, it must be accessible to the public in economic terms.

Also, other non-technical challenges are required to be beaten. These additional, but not least important, challenges offer a new field of study in which multi-disciplinary teams are required to solve these new issues:

- **Legal regulations**: as this technology is improved and enters the road, there will be new situations that are not addressed in the actual regulatory frameworks. Moreover, in the transition time between manual vehicles and AVs, there will be a period of time when both will coexist and we will need new traffic laws to accommodate our driving to these new paradigms.

- **Ethical issues**: an AV is endowed with cognitive capacities (in the artificial meaning of the word) and automatic planning capabilities, analogously to what human drivers do. This fact exposes them to encountering road scenes in which all the possible decisions outcome in a harmful way.
- **Public acceptance**: as it happens in the incorporation of any disruptive innovation, society shows a feeling of initial rejection. Giving control to an intelligent machine where a failure can cost your life is not an idea that a non-negligible part of society accepts easily.

The advantages and challenges highlighted before offer multiple opportunities to perform research or work on topics related to this field and contribute to the ultimate goal: the implementation of a technology that will reframe the concept of transportation.

### 1.2.1. The Autonomous Driving Software Architecture

Once the reader has been contextualized into a high-level view of the AD problem, it is time to present the software components that enable autonomous navigation with vehicles, given the fact that this work is framed within the development of a software DL solution for perceiving the environment on automotive scenarios. It is worth remarking that there is no closed answer to the definition of all the modules that form an Autonomous Driving Software Architecture and the reader can find multiple proposals in the literature.

In this work, we will divide the ADS into four big modules: perception, localization, planning and control, which can be also subdivided into multiple sub-modules; in order to maintain coherence with the introduction in Section 1. In order to illustrate this architecture, we will take a look at Autoware's architecture, a well-established open-source solution for AD architecture that can serve as a baseline for this study.

#### 1.2.1.1. Perception

The first competence of the ADS is perception, which is in charge of interpreting the information from the sensors in order to understand the environment in which the vehicle is moving. This is done in three steps: obtaining the information from onboard sensors, processing it, and interpreting the environment around the AV. Due to the importance of sensors within this work, they will be further explained in Section 1.2.2, but we can mention the sensors that are typically involved within this task, which are: LiDAR, RADAR, cameras, and ultrasonic sensors. The information gathered by the sensors is presented in the form of point clouds, images or spectral signals. The processing stage transforms this raw data into more sophisticated information, such as the location, dimensions, and speed of the rest of the road users (pedestrians, cyclists, and other vehicles) or a semantic definition of the world in which every piece of information is assigned to an inanimated category (drivable road, vegetation or road obstacles). This is normally done with learning-based algorithms like the ones we treat within this book. Once this elaboration is finished, it is possible to interpret the scene adding a temporal and spatial coherence between the processed information in a timestamp, and the processed information

of past timestamps. This temporal-spatial association of information allows the system to predict which possible scenarios are likely to happen within the successive timestamps. The final output of the perception processing pipeline is used as the input of the planning stages, and it could be possible to use it within the localization stage.

The traditional approach to the perception of the dynamic road users that coexist within the environment with the AV involves the concatenation of three independent modules: detection, tracking, and prediction.

- **Object Detection**: is a task that involves identifying and localizing the road users within the space surrounding the vehicle. These algorithms are fully based on DL and can be applied over camera images or LiDAR or RADAR PCDs. Its objective is to predict bounding boxes (either in 2D or 3D space) around the objects of interest and assign a class label to each of them. Bounding boxes delimit the position and dimensions of the object of interest. Also, **Instance Segmentation** is an extension to these methods which also predicts a binary mask within the pixels or points inside the bounding box that indicates if every specific piece pixel/point belongs to the object of interest.

- **Object Tracking**: is a task that involves the temporal re-identification of the objects detected in the previous frames or timestamps. Therefore, its main objective is to maintain a consistent association between the previously tracked objects and the new detections. They are also in charge of predicting the position of previously tracked objects which do not have any new detection associated. This kind of problem is mainly approached from the probabilistic robotics and Bayesian filters point of view. Learning-based algorithms are not the main components of tracking and they are constrained to certain sub-tasks, such as re-identification. It is worth mentioning that we can distinguish between *trackers-by-detection* or *detection-free trackers*. While the first ones depend on the output of a detection algorithm, the latter ones treat joint detection and tracking as a single problem.

- **Motion Prediction**: is a task that involves the regression of the future trajectories of the objects that have been previously tracked. Taking as input the past positions of tracked objects and their motion patterns, the main objective is to forecast the future positions of these adversarial agents. The information obtained from this stage allows a complete understanding of the scene in which the vehicle is moving and the planning stage will be able to make decisions and handle the driving in a safe correct manner. In an analogous way to tracking algorithms, it is possible to distinguish between *modular* and *end-to-end* approaches. While the first ones depend on the output of the tracking stage (normally prediction algorithms also receive as additional inputs an High-Definition Map (HD Map) or raw LiDAR scans), the latter ones need the raw data and perform a unified detection, tracking, and prediction.

For example, in Autoware's architecture, the detection of dynamic road users is faced as follows. There are three different inputs: a filtered and accumulated LiDAR point cloud, an RGB camera image and an HD Map. There are also three different processing branches that

will end up merged in the tracking stage. The first branch is a detection-free tracker based on **CenterPoint** [2], whose input is a LiDAR point cloud. The second branch applies Euclidean clustering detection over LiDAR data. The third branch applies **YOLO** [3] over the RBG image to perform object detection. After several validations, filters based on the map information and fusions, the three branches are merged into the tracker of the first branch. Then, this tracker is also fed-backed by the tracks of previous timestamps. The tracker is based on an Extended Kalman Filter (EKF). Lastly, the predictions are performed over these tracks and are based on the available map and road network information, such as the possible lanes to follow.

Another important task tackled by the perception layer is the detection of road lanes, traffic lights and signals, in order to understand the road network and navigate accordingly to it. The interpretation of the road network can be supported by an HD Map or fully done by sensors. In our exemplary architecture, this task is supported by an HD Map (in which the road structure and the position of traffic lights and signals are available) and Autoware proposes a module to only detect the state of traffic lights based on a **Single-Shot Detector (SSD)** [4] network with prior information of its location.

#### 1.2.1.2. Localization

The second competence of the ADS is localization, which is in charge of positioning the vehicle with respect to the environment or the map with the highest possible accuracy, that is in the order of centimetres or decimetres. This stage is approached with methods that come from the probabilistic robotics field and Neural Networks (NN) do not have a great impact.

Localization offers a wide range of possibilities in terms of inputs to the system. Raw LiDAR scans, GNSS, IMU, HD Maps and odometry sensors are suitable inputs for a localization system. As the state of the vehicle is defined as the position of the vehicle and its derivatives (velocity, acceleration, jerk, and so on), this problem is often approached with Bayesian filters such as Kalman Filters or Monte Carlo methods.

In Autoware's architecture, the localization pipeline involves GNSS, IMU, odometry, vehicle speed and LiDAR scans. The pipeline has two main branches: firstly, one branch consumes LiDAR point clouds and the tracked localization in previous stages and performs an NDT Scan Matching whose output is the motion transform between two consecutive LiDAR point clouds. The second branch integrates the information coming from the odometer, IMU, and vehicle estimated velocity. Lastly, these two branches are combined within an EKF and the final output is the state of the vehicle.

#### 1.2.1.3. Planning

Planning is the third competence in the ADS. The main responsibility of planning is to decide how to achieve the objective of autonomous navigation. How to reach point B from point A in a coherent and safe manner. Therefore, from the scene understanding performed by perception and localization modules, the planning module has to propose a series of routes of paths with which the mission or goal can be reached.

From an automotive point of view, planning can be divided into three sub-modules (from higher to lower levels of abstraction): global or path planning, behavioural planning and local planning.

Autoware's planning modules follow the aforementioned three-level structure. Firstly, a mission planner acts as a global planner and calculates a route from the ego vehicle position to the goal position given a sequence of lanes on the input HD Map. Then, there are two behavioural planners: lane driving mode and parking mode. In lane driving mode, a high-level behaviour is chosen, such as lane following, lane change or pull-over. According to this, a velocity planner acts as a local planner, which decides which velocity profile performs depending on the surrounding environment (waiting for a traffic light, near a pedestrian crossing). Lastly, this local planner is supported by an obstacle avoidance algorithm and a motion smoother, which adapts the velocity profile.

#### 1.2.1.4. Control

The fourth component of the ADS is control. Control is the module in charge of translating the missions to commands understandable by the physical devices of the vehicle that make autonomous navigation possible. This means that it is responsible for the guidance of the vehicle when operating in autonomous mode. The decision taken in the planning stage is translated into commands in terms of linear and radial velocity applied over the wheels of the vehicle.

Normally, the input of this module is the velocity profile proportioned by the planning module. In fact, this is how Autoware does. Starting from this list of velocities and the actual vehicle state (measured velocity), controllers are applied over the lateral and longitudinal axis. Both controllers are Model Predictive Controllers (MPC) and control the process needed to reach an objective given a series of velocities while satisfying a set of physical constraints.

The combined effort of these four modules in a cyclical way and in real-time allows the completion of the autonomous navigation task. At present, the area of perception is where algorithms based on learning have been most strongly implemented, and, although in the State of the Art (SOTA) there are proposals based on learning for all layers, commercial solutions still opt for probabilistic or control theory algorithms in many cases.

### 1.2.2. The Autonomous Driving Sensor Platform

A sensor is defined as an electronic device that is in direct contact with a physical magnitude and has the capability of transforming it into an electrical magnitude (voltage, current or power). It is possible to categorize sensors according to two criteria: the origin of the information and its role with respect to the emission of energy.

- **Origin of the information**: it is said that sensors that inform about the state of the vehicle are proprioceptive (battery state, velocity of the vehicle), while the sensors that inform

about the state of the external environment are exteroceptive (distance measurement, semantic information of the environment).

- **Emission of energy**: it is said that a sensor is passive when it receives energy from the environment (cameras) and a sensor is active if it emits energy and checks its influence over the environment (RADAR, ultrasounds).

As we have seen during Section 1.2.1, the scene understanding and the success of autonomous navigation depends directly on the usage of a varied suite of sensors that act in a cooperative or complementary way, applying sensor fusion techniques. In fact, not all the sensors participate directly in all the ADS modules or tasks.

The composition of a sensor suite does not have a closed solution and, in fact, is one of the most controversial topics when talking about autonomous vehicles in the mainstream. Each sensor has its key strengths and weaknesses which make them more or less suitable for each automotive task. For example, within the domain of perception, academia has made a strong bet on LiDAR and camera fusion, while other commercial solutions bet only on camera-based perception and other sectors highlight the potential of developing new generation RADARs for this task. Also within the perception domain, there is a debate around the usage of HD Maps due to their difficulties in terms of maintainability and scalability. It is a design decision to embrace the whole spectrum of strengths that each sensor offers and apply them in the domain in which they are most reliable.

Let us engage with the sensors that are mounted onboard a standard automotive platform and conduct a concise overview of their functionalities.

- **Camera**: RGB cameras are passive and exteroceptive sensors that provide a very dense representation of the environment, a pixel-level density image $I \epsilon \mathbb{R}^{C \times H \times W}$, where $C = 3$ is the number of channels in an RGB image, $H$, and $W$ are the height and width of the image, respectively. Also, it is possible to transform this information into a semantic representation of the environment with DL methods. These two main strengths and the development of learning and camera-based techniques make them the most used sensors within the domain of perception in the automotive field. One inherent weakness of the camera is the loss of depth information since all the visualized points are projected into a 2D image plane via a pinhole camera model. This can be partially addressed using stereo systems, in which two cameras are able to reconstruct the depth dimension within a limited range. Other remarkable weaknesses are poor performance against adverse weather conditions (fog, rain, or snow) or poor illumination conditions.

- **Radio Detection and Ranging (RADAR)**: it is an active and exteroceptive sensor whose functioning is based on the Time of Flight (TOF), the Doppler effect and the Frequency Modulated Continous Wave (FMCW). RADAR emits energy in the form of electromagnetic radio waves (in the automotive domain, its frequency is approximately 80 GHz) and receives their echoes reflected off objects. By measuring the time taken and the frequency shift between the emitted signals and their echoes, RADARs are able to determine range,

velocity and angular position (3D radars: azimuth; 4D radars: azimuth and elevation). Historically, RADAR has played a major role within the automotive sector, since assistance systems that cope with collision avoidance and adaptive cruise control tasks rely on RADAR since the 1990s. The advance of RADAR-based perception has evolved thanks to the development of antenna design and signal processing algorithms. RADAR data can be consumed by the perception module in two forms: point cloud, $R_{PC}$, or RADAR cube, $R_C$. RADAR point clouds are in the form $R_{PC}\, \epsilon\, \Re^{N \times 5}$, where $N$ is the number of targets in the point cloud, and every target is defined by its spatial dimensions: range $r$, azimuth $\alpha$, and elevation $\gamma$; its radial velocity captured by Doppler effect $v_r$; and its power of reflection, which use to be expressed by the Radar Cross Section (RCS). The RADAR cube $R_C$ is a representation of the power received for every bin through the spatial, angular and velocity resolution of the sensor. In the recent literature, it has been used in 3D RADARs and therefore, it has the form $\Re^{r_{res} \times \alpha_{res} \times v_{r\,res}}$. For these reasons, the main strengths of RADAR are the extraction of velocity information and its good performance against adverse weather conditions. On the other side, its main weakness is its poor spatial resolution, which leads to worse performance when comparing LiDAR-based learning algorithms versus RADAR-based learning algorithms.

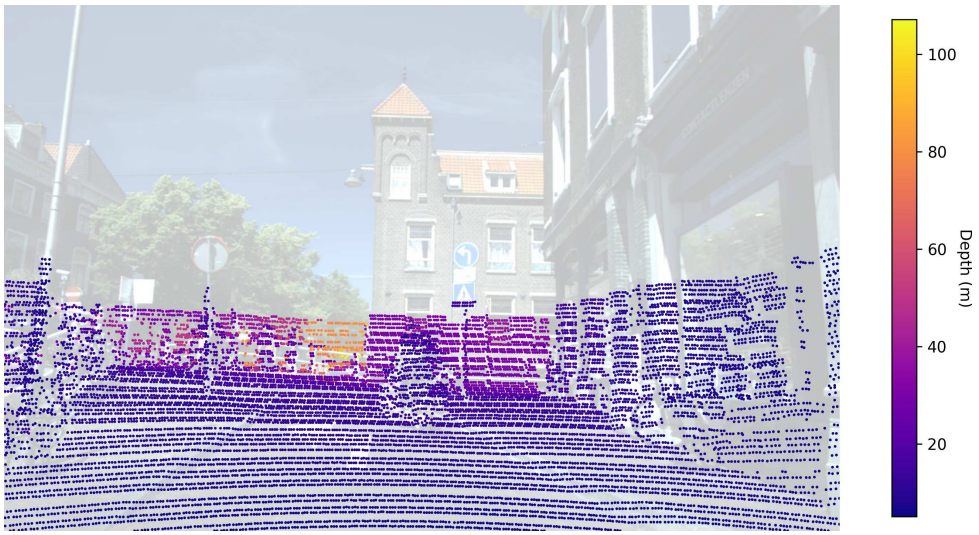
- **Light Detection and Ranging (LiDAR)**: it is an active and exteroceptive sensor whose main operating principle is TOF. LiDAR emits energy in the form of non-visible light waves which proportionate a large number of distance measurements with high precision, range and spatial resolution in the 3D domain. Typically, LiDAR information is consumed by the perception module in the form of a 4-dimensional point cloud $L_{PC}\, \epsilon\, \Re^{4 \times N}$, where $N$ is the number of points, which encodes the three spatial dimensions and the intensity of the reflection. LiDAR performs well against poor illumination conditions as it emits its own energy, but it does not perform well against adverse weather conditions. This is due to the fact of the transported energy and the frequency of the light waves. When these waves interact with water (in the form of fog or rain) its energy is mostly absorbed and a scattering process occurs, which leads to false positive distance detections. Moreover, the second big weakness of LiDAR is its heavy computational cost, since it offers a very detailed representation of the environment that then has to be processed by learning-based methods. An example of an automotive scene with camera (1.1a), RADAR (1.1b), and LiDAR (1.1c) data can be seen in Figure 1.1.

- **Ultrasonic sensors**: ultrasonic sensors are active and exteroceptive sensors that proportionate distance measurements within a very short range, whose working principle is TOF. They are widely used within the domain of robotics and in parking or blind spot detection use cases within the automotive field due to their cheapness and ease of interpretation. Their main advantage is that they are precise enough to allow parking manoeuvres within a limited working range.

- **Odometry**: is a process that allows obtaining an estimated displacement by the vehicle thanks to the measurements that come from an encoder. The odometry process is inac-

(a) Monocular camera image.


(b) RADAR Point Cloud projected to the image plane.




(c) LiDAR Point Cloud projected to the image plane.

Figure 1.1: Camera, RADAR and LiDAR for an automotive scene of View of Delft dataset.

curate since it is prone to accumulative errors and does not take into account non-ideal displacements.

- **Inertial Measurement Unit (IMU)**: it is a set of proprioceptive and exteroceptive sensors such as compass, gyroscopes and accelerometers whose main purpose is to characterize the velocity, orientation and accelerational forces that the vehicle is experimenting.
- **Global Navigation Satellite System (GNSS)**: it is a positioning system comprised of 24 satellites that orbit the Earth. Each satellite emits its position and timestamp continuously. Using the trilateration principle, a sensor can determine its position in outdoor environments if it is able to reach the information of at least 4 satellites. Positioning through GNSS has limited accuracy in the order of meters, so, in order to improve this value, there are variants such as differential Global Positioning System (GPS) (DGPS) or real-time kinematic GPS (RTK-GPS), which with centimetre accuracy can be reached. Odometry systems, IMU and GNSS can work collaboratively to characterize the state of the vehicle: position, velocities and acceleration. Therefore, these sensors are mainly used within the localization domain.

Additionally, there is a dichotomy in the automotive domain about driving with a priori detailed information or not. In this context, maps are an additional source of information that can be considered as a pseudo-sensor:

- **High-Definition Map (HD Map)**: a map is an abstract representation of the environment that only considers the features that are useful to navigate. In the case of HD Maps, they are a digital representation of the road network and its main elements, such as traffic lights, signals and drivable surfaces. As previously mentioned, the presence of an HD Map in an ADS can redefine how to process the information of the environment. When a map-based architecture is proposed, it can be integrated into the perception, localization, and planning modules.

### 1.2.3. Levels of Automation

The achievement and worldwide implementation of autonomous driving is a gradual process, which might take several years to come to an end. The concept of an Advanced Driver Assistance System (ADAS) appears in this context, due to the fact that research and companies introduce partial systems that automate a certain part of the ADS.

Vehicle manufacturers started introducing safety or convenience features such as seat belts or anti-lock brakes 1950s. However, the transition from fully manual to complete autonomous driving is a process that began in the decade of 1980 when the advent of probabilistic robotics made researchers put their eyes on autonomous navigation with vehicles. It was in the late 1990s when Toyota put into commercial production the first ADAS. Nowadays, we live in a time in which partially automated vehicles coexist with manual vehicles and the vast majority of new vehicles that are manufactured include multiple ADAS. Maybe, the following decades will witness the achievement of a fully AV.

Given the diversity of proposed and existing systems, there emerged a necessity to establish a formal measure of a vehicle's degree of partial autonomy. It was necessary to propose a simple yet effective rule set that enables the reconciliation of a technological reality with the economic and legal frameworks that rule our society. The Society of Automotive Engineers (SAE) proposed in 2014 a classification protocol known as **SAE J3016** in its document "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems" [5]. There are 6 different levels of automation which are explained in Table 1.1.

## 1.3. Objectives of this work

The main objective of this work is the study, implementation, and evaluation of a system of DL with multiple neural networks (multi-stage) for the detection of 3D objects in three-dimensional space from commercial 3+1D RADAR point cloud data and images from a monocular camera. This approach aligns with trends in recent literature, which seek to design methods for the sensor fusion of automotive radar and monocular cameras that have a performance similar to that of the LiDAR-based methods. Therefore, the final purpose of the system is the detection and classification of static and dynamic objects that surround the vehicle through DL with these two sensors following a geometrical-sequential sensor fusion scheme.

Moreover, this work is framed within the context of the **AIVATAR** project, carried out by **RobeSafe Research Group** (led by Prof. Luis Miguel Bergasa and Prof. Rafael Barea) at **Universidad de Alcalá**. The global objective of the project is to implement an AD stack with DL-based modules. This work is focused on the perception layer of the stack, in which efforts are made to include RADAR to an established perception baseline based on LiDAR and cameras.

In greater detail, we proceed with the formulation of the specific objectives of the project:

- Perform a theoretical study on the recent literature about RADAR and camera fusion in the context of Autonomous Driving.
- Perform a comparison on Instance Segmentation models for monocular cameras as the first stage of the sensor fusion pipeline, from which RADAR point clouds will be augmented and enriched with novel features coming from semantics.
- Adapt a DL training framework for LiDAR 3D Object Detection methods in order to accept modified and enriched point clouds as inputs, such as radar point clouds or augmented radar-camera point clouds.
- Evaluate the models in a dataset that include automotive RADAR point clouds and monocular image cameras.
- Improve the solution through experimentation, performing a comparison between available models, loss functions, and optimizers.
- Develop an inference pipeline that receives RADAR point clouds and images as inputs and performs new inferences using the final trained pipeline.

| Level of Automation | Characteristics and examples |
|---|---|
| Level 0<br>Momentary Driver Assistance | At this stage, the human driver is in full charge of the steering and braking tasks. It must be completely engaged and attentive to the driving situation until Level 3. At this level, warning technologies are included. Mechanisms alert the driver of any non-desired condition, but he must solve them without any automatic help.<br>**Examples**:<br>■ Forward Collision Warning (FCW)<br>■ Lane Departure Warning (LDW) |
| Level 1<br>Driver Assistance | Although the driver keeps on fully engaged and attentive, some functionalities are automated and they are enabled manually or automatically when the vehicle's perception recognize a non-desired situation. At Level 1, these mechanisms act on the steering wheel or accelerator-brake assembly, but not on both simultaneously.<br>**Examples**:<br>■ Adaptive Cruise Control (ACC)<br>■ Emergency Breaking System (ABS)<br>■ Lane Keeping Assistance (LKA) |
| Level 2<br>Additional Assistance | At Level 2, assistance systems can take control over both steering wheel and accelerator-brake at the same time. It is possible to say that this is the first level in which the human driver is not in charge of navigation.<br>However, the driver must be fully engaged and with his hands over the steering wheel, as he is monitoring the scene, not the vehicle.<br>**Example**:<br>■ Highway autopilot |
| Level 3<br>Conditional Automation | Level 3 transforms assistance into automation. The first level in which the system drives autonomously through a wide range of road situations. In this scenario, the driver can relax but he must be available to take the control upon request of the system, as it identifies a situation that can not handle correctly.<br>The only commercial solution until today is Mercedes-Benz's S Class but its operating range is limited to the German Autobahn and the state of Nevada. |
| Level 4<br>High Automation | The only difference between L3 and L4 is that the range of driving situations operated autonomously is much wider than the ones that require human intervention. Also when engaged, the driver takes only partial control of the vehicle. |
| Level 5<br>Full Automation | At Level 5, fully autonomous navigation is universally fulfilled by the system and the occupants only act as passengers. In fact, a vehicle would not require the existence of a steering wheel, accelerator, or brake. |

Table 1.1: SAE J3016 Levels of Automation framework
Source: nhtsa.gov. The Evolution of Automated Safety Technologies.

## 1.4. Overview of the document

This Section aims to explain the contents of the different Chapters that the reader will encounter during this book.

- **Chapter 1. Introduction**: The main purpose of this first chapter has been to introduce the reader to the problem of AD, providing a brief review of it. From its definition and a historical point of view, going through its main components such as the software stack and its sensor platform, to its most representative legal framework.
- **Chapter 2. Theoretical Background**: This Chapter will introduce theoretical concepts that are required to perform this work: from the Machine Learning (ML) process to definitions in the area of DL.
- **Chapter 3. State of the Art**: A comprehensive review of the recent literature about Deep Learning applied to RADAR and camera, and the fusion of both sensors that will serve to build intuitions about this project.
- **Chapter 4. Development of the work**: The core practical implementation of this work will be explained in this Chapter. From exploring the dataset to the implementation of all the modules within the detection framework, going through detailed explanations of each of them.
- **Chapter 5. Experiments and Results**: The training and experiments procedure will be introduced in this Chapter, along with all the results and comparison between the baseline and the proposed solution. Quantitative, qualitative and a time analysis compose this Chapter.
- **Chapter 6. Conclusions and Future Works**: This Chapter concludes this dissertation and includes an analysis of the most relevant findings and conclusions and the future lines of work that have been identified.

# Chapter 2

# Theoretical Background

*I know of no better life purpose than to perish in attempting the great and the impossible.*

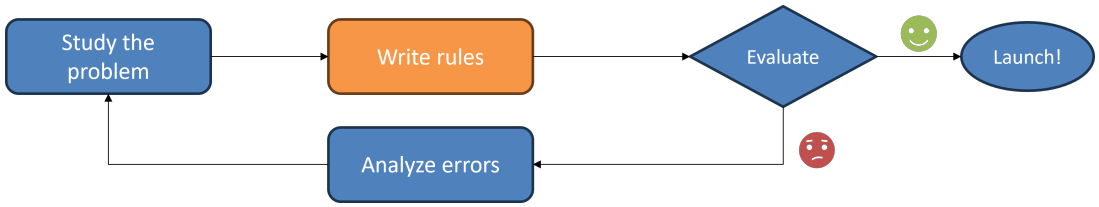
Friedrich Wilhelm Nietzsche

## 2.1. Introduction

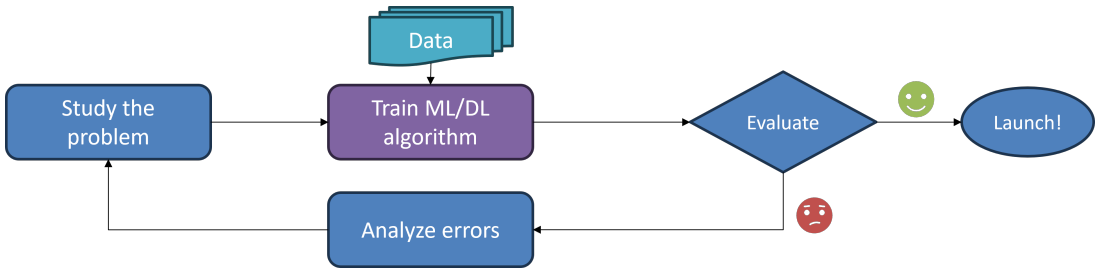
As aforementioned in Chapter 1, the algorithms used in the domain of perception in automotive environments are mainly learning-based approaches. Taking as input the information captured by onboard sensors, these algorithms must extract and process the most relevant features of the environment in order to build a higher level of abstraction knowledge, the scene understanding. That is, from raw sensor signals to bounding boxes and identifiers that create a representation of the surrounding environment, easily understandable by a human user.

The shift from traditional or rule-based approaches to data or learning-based methods has changed how algorithms are done. A high-level comparison of both can be seen in Figure 2.1. Moreover, data has become one of the key components of the pipeline. Therefore, it is important to know how the onboard sensors that provide information work and how can a programmer get the most out of the data they proportionate. Within the scope of this book, RADAR and cameras are of particular interest.

Throughout this Chapter, we will go across the Machine Learning Workflow, an end-to-end vision of how a complex problem should be addressed with a learning-based approach. Going through all the stages of the development process will create the foundations needed to cope with the practical project developed in Chapter 4. Then, we will delve into the working principles of RADAR and cameras, explaining their theoretical models and how they offer representations of the environment.




(a) Traditional or rule-based programming approach.




(b) Learning or data-based programming approach.

Figure 2.1: Rule-based and Machine Leaning programming approaches.
Source: *Hands on Machine Learning with Scikit-Learn, Keras & Tensorflow, 2.ed.* Aurélien Géron

## 2.2. The Machine Learning Workflow

As said in Chapter 1, recent progress within the Machine Learning (ML) field has allowed the incorporation of new technologies and solutions into our domains of interest: Robotics and Autonomous Vehicles. During this Section, we will deep dive into the process of building a project or solution using ML. This will also allow us to present the main components: the data, the model, the optimizer, its main parameters, etc. It is essential to clarify that the terms Machine Learning, Deep Learning and Artificial Intelligence are used interchangeably since they are interrelated areas of knowledge. In any case, there are subtle differences when it comes to defining the three disciplines. AI is a sub-discipline of Computer Science that aims to build autonomous machines that imitate human behaviour in order to complete tasks. Then, ML is a sub-discipline of AI comprised of a set of algorithms whose performance is increased with experience. In other words, they *learn* through finding patterns in the data they are shown. Lastly, DL is a sub-task of ML which is defined by the kind of algorithms that are used to learn these patterns, Deep Neural Networks (DNN). The interrelation between these disciplines can be graphically observed in Figure 2.2.

When dealing with a ML problem, the first step consists of formulating the problem in a correct manner, following the concept of task. This applies to any supervised learning problem (a problem with a target label available). Every task can be broken down into a composition of the two essential tasks in Machine Learning: classification and regression. Classification is the task of predicting a discrete class label, such as cat or dog in image classification. Regression is the task of predicting a continuous value, such as an energy value for a specific daytime in a time series.

Exemplifying this concept of defining tasks as a composition of classifications and regressions, we can talk about instance segmentation on images, which is a task we deal with in this

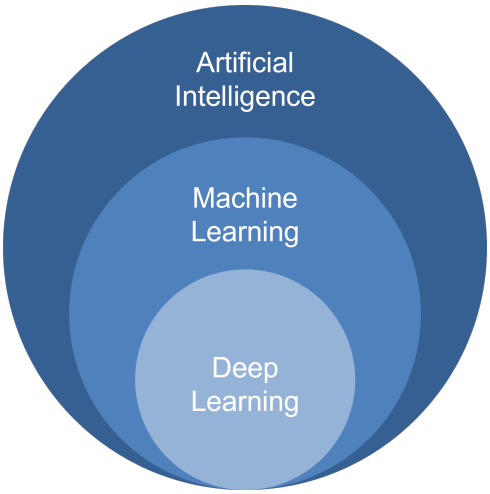



Figure 2.2: Interrelation between Artificial Intelligence, Machine Learning and Deep Learning in a Venn diagram.

project. In the literature, instance segmentation architectures are mainly designed in two ways: the first one is to inherit the main structure of a detector and add a segmentation head part [6]; the second one is to inherit a semantic segmentation network structure and add a watershed transform to identify different instances within a mask of the same class [7]. From a high level and following the first approach, this task can be seen as predicting segmentation masks over the object of interest in an image. Predicting a segmentation binary mask over a detected object can be decomposed into a pixel-wise binary classification problem. Then, localizing an object in an image can be formulated as a composition of four regressions that define a bounding box (top-left corner ($x$, $y$ coordinates), height and width) and a classification, that assigns a discrete label to the object. With this formulation, we have decomposed a complex abstract problem into single classifications and regressions. Once we proceed to analyze a new model or framework, we will expect to encounter metrics and loss functions that evaluate the performance of the goodness of a model within these smaller sub-tasks; and after we proceed to approach a new problem with different data, we can take advantage of this analysis to use some components as reusable parts.

Before starting the work, it is also important to define the objectives of the project: Where do we want to take it? Previous planning must be approached from a triple point of view: programming skills, statistical and machine learning knowledge, and domain-specific knowledge. Therefore, the influence of these three components will make the planning different depending on whether the solution should be executed in a web service, a real-time embedded system or a simulator with asynchronous execution; if it is going to be executed once a day or thousands of times per hour; if we prioritise not failing instead of predicting incorrectly; and so on. You can even encounter a situation in which you have to decide if the project is even possible to execute due to the computing power available. Being able to design the process from a high level and in all its stages is a good method to anticipate problems and increase the chances of its success. It is worth mentioning that these criteria are not fully closed and can vary during the design process due to changes in the requirements or technical impossibilities that may appear during the development phase.

In summary, a supervised problem in the domain of Machine Learning has six main components:

- A data set, composed of samples and features $X = (x_0, x_1, \ldots, x_N)$.
- A set of ground-truth labels $Y = (y_0, y_1, \ldots, y_N)$.
- An algorithm $\hat{y} = f\left(x; w_j^*\right)$, with a set of trainable parameters $w_j$ which aim to perform optimally $w_j^*$ at a determined task and returns a prediction $\hat{y}$ when an inference is made.
- A loss function $\mathcal{L}(\hat{y}, y)$, also referred to as cost or error function, which assesses the goodness of the algorithm in a *algorithm-understandable* manner.
- A set of metrics $\mathcal{M}(\hat{y}, y)$, which assesses the goodness of the algorithm in a *human-understandable* manner.
- An optimizer $\mathcal{O}$, which is in charge of adjusting the model parameter set $w_j$.

These components will be involved in the different stages that appear within the context of a Machine Learning workflow. From getting the data to deploying a final solution, and going through the training of a model.

### 2.2.1. Data acquisition and exploration

The first step in a ML project consists of getting a dataset and exploring its potential features. Depending on the nature of the problem, the data can come from various sources of information. When dealing with an automotive problem, it is necessary to install a sensor suite within a test vehicle and drive for long hours in different environments to collect data. Once the data is collected, it is necessary to generate the labels that indicate where the objects of interest are located, to which semantic class they belong or any target information that we want to infer in an automated manner.

Labelling is a time and effort-consuming task since it requires human supervision and datasets are comprised of thousands or even millions of samples. Due to this and to the fact of creating a common framework to perform a fair comparison, researchers decided to work with publicly available datasets gathered by universities or technological companies that can record data at a mass scale. Within the automotive domain, we can remark on the role of KITTI [8], nuScenes [9] and Waymo [10] datasets. Unfortunately, due to the recency of the introduction of High-Resolution 3+1D RADAR, none of the previous ones include this sensor. More recent datasets such as View-of-Delft [11] and TJ4DRadSet [12] include this specific sensor and open a new path for future research within perception based on camera and RADAR algorithms.

Before getting into algorithms, data is analyzed via Exploratory Data Analysis (EDA). EDA is a process in which a dataset is investigated and its main characteristics are summarized. During EDA, the focus is on data visualization and statistical distributions and relationships, univariable or multivariable. The main objectives of EDA are: finding relationships within variables in data,

suggesting hypotheses and developing domain-specific knowledge, supporting the selection of algorithms and techniques, and even, applying a first model to provide a baseline for further experimentation. Some of the most common tools used in EDA are: scatter plots, box plots, histograms, correlation matrices and dimensionality reduction techniques, such as Principal Component Analysis (PCA).

The realization of a good EDA allows us to obtain relevant reasoning about the data and can anticipate the appearance of certain errors related to them during later stages of development, such as:

- Insufficient quantity of training data.
- Non-representative training data.
- Poor quality data.
- Irrelevant data features
- Overfitting or underfitting the training data

### 2.2.2. Preprocessing

At a glance, the goal of this stage is to turn data into tensors that can be consumed by learning-based algorithms. In a typical ML workflow, the preprocessing stage involves three main parts: cleaning data, selecting features and transforming the data. Data cleaning is a process that aims to find outliers or noisy samples that do not incorporate relevant information about the task we are dealing with. Among samples that should be cleaned, we can find samples with missing features or non-valid values or samples with a large deviation in certain features. In the context of CV, distorted images, non-correctly labelled samples or samples with failures coming from the acquisition stage are wished to be removed. Data transformation involves the normalization or standardization of data. Ranging numerical features from 0 to 1, or normalizing the color range of the images of a dataset can help to stabilize the training process. Also, encoding categorical features into vector or embeddings permit the adaptation of a categorical concept into a mathematical one that can be handled by a ML algorithm.

Once our data has been correctly preprocessed, it is important to split the dataset into various parts. This process is known as **data splitting**. The number of splits and the amount of data dedicated to it can vary depending on the amount of data available and the nature of the problem, but a normal procedure will involve the creation of three splits: training, validation and testing.

- **Training set** (70%): the data shown to the model during the training stage.
- **Validation set** (20%): the data used to assess the performance of the model during the training stage. Feedback can be provided to the training stage. For example, the learning rate hyperparameter can be reduced depending on the performance of a network in this set.

- **Testing set** (10%): the data used to assess the performance and capabilities of the model once it has been trained. The model can not learn or have feedback from this set. The labels of the test set are not provided to the researcher in public competitions. The model must predict over the samples and the predictions are sent to a remote server (via a text file) that is in charge of the evaluation procedure.

It is important to keep these splits with different data to avoid *over-fitting*. For example, if a dataset is composed of a set of images extracted from a series of video sequences, these splits must be made taking into account that all the images that belong to the same video sequence must be stored in the same data split.

Lastly, **data augmentation** is a technique applied in the preprocessing stage that aims to obtain variants of a sample to increase the generalization capabilities of a model. This means that the network should be able to predict the same value or label independently of small variations that are produced within a sample.

### 2.2.3. Training of Machine Learning algorithms

The process in which an ML algorithm, $f\left(x; w_j^*\right)$, is fed with data, $Y$, and readjusts its parameters $w_j$, in an iterative way through an optimizer $\mathcal{O}$ is called **model training**. The training of a learning-based algorithm is enabled by two algorithms: **gradient descent** and **backpropagation**. This process consists of making predictions $\hat{Y}$ over the training set of data (also known as forward pass) and producing an output response for every sample, $\hat{y}$. Then, these predicted outputs are compared with the set of expected outputs or ground-truth labels, $Y$, by means of a loss function $\mathcal{L}\left(\hat{Y}, Y\right)$ that indicates the accuracy of the model. Then, the objective of a neural network is to minimise the value of $\mathcal{L}\left(\hat{Y}, Y\right)$ and this is done via gradient descent, an algorithm that tries to find a minimal value within a differential function. After the forward pass, the feedback provided by $\mathcal{L}$ must be propagated from the output to the input through every weight and bias of the neural network, in a process known as backward pass. This algorithm uses the Chain Rule of Calculus and traces paths from previous layers to the output using a chain of derivatives. Then, the gradient descent updates each weight $w_j$ in the negative direction of the gradient $(-\nabla f)$ with the purpose of achieving the optimal weight $w_j^*$ that makes $\mathcal{L} \to 0$. This process is repeated several times (epochs) until it converges ($\Delta\mathcal{L} \to 0$): forward pass, backward pass (both passes make up backpropagation) and weight updating via gradient descent.

Loss functions $\mathcal{L}$ must be differentiable, so we can apply these training algorithms. Therefore, they must represent the objective that a human is pursuing to solve with the model, a very high-level abstraction concept. Here, the formulation of supervised tasks as heirs of fundamental classifications and regressions plays a vital role. For example, we can define a depth estimation task as a pixel-wise regression in which every pixel represents a single depth. Then, a regression-like loss function can be formulated to train a neural network that solves it. A pixel-wise Mean Absolute Error (MAE) can be useful for this purpose. It is worth mentioning

that loss functions are another interesting topic of research, since not only does the model enable the application of DL to a certain task.

Optimizers $\mathcal{O}$ are the algorithms with which the gradient descent is computed. Some of the most common optimization algorithms used in DL are: Stochastic Gradient Descent (SGD), RMSProp, Adadelta and ADAM. The learning rate $\alpha$ is the most important hyperparameter that affects the training of a NN. It weights the step size on each iteration of the optimization progress while it is moving towards the minimum. There is some research about finding the optimal learning rate procedurally [13], but commonly, $\alpha$ ends up being tuned via experimentation.

An important concept to learn within the training stage is the **bias/variance trade-off**. It states that the total generalization error of a model is comprised of three main components: bias, variance and irreducibility.

- **Bias**: is the error made due to wrong assumptions or hypotheses during the design stage, i.e. choosing an algorithm that has not been designed to approach a specific problem. The consequence of these errors leads to **underfitting**.
- **Variance**: is the error produced when a model shows excessive sensitivity when small variations at the input occur, i.e. a big model is applied over a small dataset with much less complexity, and the model itself is able to memorize the training data (find the weight combination that solves the full training set). This phenomenon leads to **overfitting**.
- **Irreducible error**: is part of the error associated with the noisiness of the data. If this kind of error is detected, the designer should revisit the preprocessing stage and obtain more relevant features or detect and remove non-representative data (outliers).

Being able to identify these sources of error during the training or evaluation stages can lead to changes in the model, either its implementation or the decision about which model to use. As ML is an experimental science in which experiments guide intuition, a designer could tackle bias by increasing a model complexity or could tackle variance by decreasing it. This balance game is what makes this phenomenon be called *trade-off*.

### 2.2.4. Evaluation of the algorithm performance

As we have presented in Section 2.2.3, while loss functions are made to be *algorithm-understandable* (continuous and differentiable), metric functions are made to be *human-understandable*. In other words, they are made to advise the DL practitioner about the performance of the model. A loss function can perform as a metric but a metric may not be able to perform as a loss function. Metrics related to the project will be defined in Chapter 3.

In addition, metrics are not the only way to assess the goodness of a model. Depending on the objective of the project, there can be other indicators which reflect other aspects of the model. For example, if we are looking for a real-time application, the inference time or amount of computation in $TFLOPS$ can be a good indicator. If we are looking towards embedding a model in an edge device, energy consumption or occupied disk memory can be other good indicators of performance.

### 2.2.5. Improvement through experimentation

ML and DL are experimental disciplines. The development and creation of new models can be related to playing with LEGO blocks. From a set of essential blocks, the DL practitioner can propose several block combinations that lead to a more complex structure. These complex structures can also be combined to conform the whole architecture of a model. Therefore, one typical stage in the ML workflow is to build a baseline model from which differential improvements are made: introducing novel blocks, optimizing previous ones, etc. This is a process that requires certain intuition and deep knowledge of each building block. Once a modification is introduced, the new version model is trained and its performance is evaluated.

A procedure to optimize architecture is **hyperparameter tuning**. Gathering all the possible hyperparameters, a model with the same architecture is trained varying these numbers, until the optimal combination (the one that leads to better performance) is found. Two common hyperparameter tuning strategies are *random search* in which the hyperparameters are sampled randomly, and *grid search* in which the range of hyperparameters is sampled in equidistant intervals.

### 2.2.6. Model deployment

The last stage in the ML workflow is the model deployment. Depending on the context, this stage can be known as inference mode or model exploitation. During this stage, the model weights are *frozen*, meaning that they would not be updated in a training procedure. At deployment, models will only be fed with new data and return their predictions. It is important to monitor the deployed solution and trigger alert events when there is a drift in performance. This can be dealt with through online training, and retraining the model with new samples received during the deployment. Deploying a model can mean:

- Integrate the model into an embedded system for real-time operation in a robot or AD.
- Automate the model inference process within a simulator.
- Incorporate the model into a mobile or web application to perform predictions on demand.

Once the model is deployed and working correctly, the ML workflow is concluded succesfully.

## 2.3. Working Principles of RADAR and Camera

During this Section, we will cover the foundational concepts behind the two sensors that are used within the scope of this project: RADAR and camera. Beginning with a brief historical motivation, we will deep dive into the working principles and main components of these two technologies, and why they are suitable for perceiving the environment within the automotive domain.

### 2.3.1. RADAR

RADAR is a technology that was greatly developed during World War II with the purpose of detecting moving targets within the air domain. This development was driven by the early projects of Nikola Tesla, who performed the very first applications of electromagnetic waves to the detection of moving objects in the early 1900s. At the expense of the beginning of the War, the British industry made a significant investment in terms of effort and resources during the decade of 1930. From this point on, naval battleships and the British coastline were equipped with multiple radar systems that performed air surveillance tasks. Enemy aircrafts could be detected within kilometres away, allowing the possibility of getting ready for combat and eliminating the surprise factor on the enemy side. Due to the potential shown within this particular domain, all the participants in the War rapidly showed interest in introducing RADAR equipment into their armies. During the following decades, RADAR played a major role in the control of maritime and air traffic and navigation.

Then, when ADAS arose in the decade of 1990, RADAR became a suitable sensor to be mounted onboard ground vehicles and it has historically completed tasks that require a precise velocity estimation, such as Adaptive Cruise Control (ACC) or Assistance for Emergency Braking (AEB) technologies. Nevertheless, during the decade of 2010, academia started a major bet on LiDAR and camera for AV. Then, on the industry side, Tesla decided to remove RADAR from their L2+ solution, since the output data it provided was not rich enough to complement the scene understanding and decided to rely only on vision. RADAR stepped back into a secondary role. Taking into account the weaknesses observed, such as poor spatial and angular resolution, High-Resolution 3+1D RADARs appear. The industry is developing radars that are able to output denser point clouds with resolution in both, azimuthal and elevation angles. Even Tesla knows that an electromagnetic wave-based distance and velocity measurer is needed to reach fully AD technologies, and they have ensured that once Hi-Res 3+1D RADAR establishes itself in the industry, it will be incorporated again into their sensor suite. The advent of 3+1D RADARs also enables the application of DL methods, as most of the issues that were related to data (presented in Section 2.2) may be avoided.

Due to their lower costs (compared to LiDAR technologies), estimation of velocity and the advent of High-Resolution 3+1D RADARs (enabling a high spatial resolution in denser point clouds) show great potential for deployment of ADAS and AD technologies at mass-scale, working cooperatively with camera sensors.

During this section, we will review the fundamentals and basics of RADAR operation, as well as their signal processing pipeline and the output formats of radar data, that can be later used by the perception module of an AV.

#### 2.3.1.1. RADAR Physics

A RADAR is a sensor whose functioning is based on TOF principle for range estimation and the Doppler Effect for velocity estimation. In the case of TOF, the sensor can calculate the range from the reflection given the time difference between an emitted and a received echo. In the

case of the Doppler Effect, when there is a relative movement between the emitted signal and the detected target or object. The frequency difference appears between the transmitted and received signal, and from this difference, the relative speed is estimated.

Specific domain knowledge about the fundamentals of radar physics and signal processing is essential for proper decision-making in the development of radar techniques based on DL. RADARs in the automotive domain operate in the 76-81 GHz frequency band. Therefore, radar waves have wavelengths of approximately 4 mm, which gives them the ability to propagate over water drops and dust particles. This wavelength reduces the amount of scattering and leads to one of the strengths of radar: it performs well against adverse weather conditions.

The general working principle of a RADAR is described in the mathematical formulation of the *radar equation*, formulated in 2.1, which represents the physical dependencies between the transmitted signals and its returning echoes. The equation assumes ideal conditions for the propagation of electromagnetic waves, such as no scattering, and isotropic radiators which will propagate the energy in all directions forming an ideal sphere.

$$P_r = \frac{P_t \cdot G_t \cdot A_e \cdot \sigma}{(4\pi)^2 \cdot R^4} = \frac{P_t}{4\pi \cdot R^2} \cdot G_t \cdot \frac{\sigma}{4\pi \cdot R^2} \cdot Ae \tag{2.1}$$

The relationship between transmitted and received signals is formulated in terms of power, $P_t$ and $P_r$, respectively. The equation states that: the received power $P_r$ is directly proportional to four different terms: the power density at a distance $R$ from an ideal isotropic antenna, the antenna gain of the transmission antennas $G_t$, the reflectivity or amount of energy that a target reflects back (characterised by RCS or $\sigma$), and a term related to the reception antenna gain, which is characterised by the effective area of the antenna $A_e$.

From the first term, it can be observed that the transmitted power decays with the distance $R$ to the sensor in a quadratic factor, since the propagation of waves occurs in a spherical shape. The numerator $P_t$ is expressed in $W$. Then, the antenna gain $G_t$ is an adimensional value that addresses how well an input power is converted in radio waves. These two terms are related to the capabilities of the emitter and its product is the power density that is converted to radio waves that will be processed in a further stage. Subsequently, the third term expresses the amount of power that is reflected back to the sensor. As it is another isotropic wave propagation, it also decays with a quadratic factor with respect to distance $R$. Nevertheless, in this case, the reflecting capabilities of a target are expressed by means of the RCS, $\sigma$. RCS is a measure of how detectable is a target by a RADAR. In the automotive domain, the reader might be familiarised with the concept of intensity, which is the analogous term in LiDAR terminology. RCS is influenced by various factors such as the material with which the target is made, the size of the target, the incident and reflected angles, and more. This value used to be characterised experimentally in RCS diagram for a wide range of targets. As a curiosity, when stealth aircraft are produced in the aviation industry, one of the main objectives during the design process is to minimize the RCS profile. RCS is expressed in $m^2$, which leads to an adimensional third term. Lastly, the fourth term is related to the geometry of the reception antenna, as it expresses its surface in $m^2$. This term can be weighted by an efficiency factor $K$, which addresses the

imperfections of the antenna and the losses that are produced during the reception. As a rule, is a value that oscillates between 0.6 and 0.7. The product of all terms results in a received power expressed in $W$.

*Radar equation* is also used to determine the maximum range or distance at which a target can be detected. Solving the equation for $R$:

$$R_{max} = \sqrt[4]{\frac{P_t}{P_r} \cdot \frac{G_t \cdot A_e \cdot \sigma}{4\pi}} \tag{2.2}$$

From the equation, three main conclusions can be extracted:

- The relationship between the reflected and transmitted power, $P_r$ and $P_t$, decays with a factor of fourth power. Given this fact, it is expected that an output point cloud density will also decay with distances. While closer areas to the sensor are denser, the far field will be less populated.
- The reflecting capabilities of a target, expressed by means of RCS is independent with respect to distance. On the other side, this parameter is related to the shape and materials of the reflected targets and it shows certain potential for target classification. However, the statistics of certain datasets show that RCS increases with range. This is a phenomenon directly related to the previous conclusions. In the far field, targets are more difficult to solve and only the stronger targets will remain, while in the close field, there will be a higher diversity of targets with different RCS (even if they belong to the same semantic class).
- The capabilities on the radar are tightly coupled with the quality and design choices with respect to the antennas. Even without bearing with powers, the transmission antenna gain $G_t$ and receptor effective area $A_e$ are directly proportional to the reflected power $P_r$ and the maximum range of detection $R_{max}$.

As anticipated within this section, a set of electronic components are needed for a RADAR to work properly. Due to the maturity of the technology, several designs can be found in the literature. However, there are some common and foundational components that must be present in every RADAR: a wave generator, a transmitter with a set of transmission antennas, a receiver circuit with a set of multiple reception antennas, a duplexer, and a mixer to make comparisons between the transmitted and the received signal.

The transmission antennas (Tx) emit signals using linear frequency modulation (FMCW is what allows for measuring position and velocity simultaneously), commonly referred to as chirp, and the reception antennas (Rx) receive the signals that bounce back from nearby objects. The received Rx signal is combined with the Tx signal, resulting in a lower intermediate frequency signal. This signal is then captured by an analogue-to-digital converter (ADC) to provide the raw time domain data. The intermediate frequency corresponds to the distance of the reflected object from the radar. Multiple chirps are transmitted to enable the calculation of radial velocity, achieved by comparing phase shifts among received chirps.

The term *fast-time* denotes the dimension within a single chirp, as determined by the ADC sampling frequency. Conversely, the *slow time* refers to the dimension spanning across all chirps, governed by the chirp repetition time. To facilitate the estimation of the azimuth angle, indicating the direction of arrival, the Rx-antennas are horizontally spaced from one another. This arrangement ensures that the phase progression across different Rx elements is indicative of the angle at which the signal is received.

In summary, the raw time domain data comprises three dimensions: fast-time, slow-time, and Rx-element dimension, collectively capturing the radar's comprehensive information, forming a three-dimensional tensor that is sent to the next stage: the signal processing pipeline.

#### 2.3.1.2. RADAR Processing Pipeline

Taking as input the raw time domain data, the main objective of the signal processing pipeline [14] is to extract information about the ranges, angles and velocities of the relevant targets. Then, this information is represented in diverse and compact forms, because they will be the source of data for the algorithms in the perception pipeline. A diagram of the complete processing pipeline is depicted in Figure 2.3, and we will walk through it during this Section.

Firstly, a Fast Fourier Transform (FFT) is applied over the chirp index or *fast-time* dimension. This converts *fast-time* from the time domain to the frequency domain. This will lead to a frequency spectrum representation in which higher frequencies will indicate higher reflections, and therefore, possible targets. Nearby targets can be close up in the frequency spectrum and will be represented by a single peak. Here, it is where the concept of resolution comes in. The equations for range estimation and resolution are shown in Table 2.1. As seen, the resolution can be increased by increasing the bandwidth $B$, as they are inversely proportional and we are working on a centimetre scale.

Applying a Short Time Fourier Transform (STFT) over the result of the Range FFT it is possible to obtain a Micro-Doppler Spectrogram. This is a representation that does not provide spatial information but captures fine-grained detail about the temporal evolution or motion pattern of a moving target, which provides valuable insights for target classification tasks.

Then, a second FFT is applied over the range bins in the Range FFT output. It solves velocity for each range bin. As previously mentioned, the key to solving velocity is measuring the phase difference between emitted and received chirps within the mixer, so equations show different relationships with phase difference $\Delta\phi$. Range-Doppler map or spectrogram is the output of Doppler FFT. It represents the world within both axes, range and velocity. Two objects that are at the same distance, with the same relative velocity and different angles will be shown in the same bins within this representation.

Lastly, a third (or even fourth if there are two angles) FFT is applied over the Range-Doppler maps which allow us to distinguish between angles. The RADAR will need to be equipped with multiple reception antennas with a known physical distance among them and a frame of multiple chirps spatiated in time to discern the direction of arrival of the target. As shown in equations (Table 2.1) the resolution on angle estimation depends on the number of

reception antennas, $Rx$. The result of the Angle FFT is the radar cube, $R_C$. It is a 3-dimensional representation in which two dimensions (range and angle) encode the spatial disposition of the environment in polar coordinates, and the third dimension indicates the velocity of a target detected within that range-angle bin. Recent works try to exploit this representation in order to detect and classify road users. [15]

The FFT-processed radar cube may be composed of more than 32 million cells: 1024 range bins, 1024 velocity bins, and 32 angle bins, for 3D RADARs. In 4D RADARs, it is necessary to add an additional set of bins for the elevation angle. Assuming a 16-bit value precision, the data transmission rate to transfer radar cubes in real-time will reach 2.5 $Gbps$ or 10 $Gbps$. Nowadays, ingesting this amount of data seems impractical for real-time DL algorithms (or even traditional ones). Also, not all the cells within the cube contribute relevant information since the cube describes the whole space: targets, empty space and noise reflections.

| Parameter | Estimation | Resolution |
|---|---|---|
| Range | $R = \frac{c \cdot f_B}{2 \cdot S}$ | $R_{res} = \frac{c}{2 \cdot B}$ |
| Doppler velocity | $V = \frac{\lambda \cdot \Delta\phi}{4\pi \cdot T_C}$ | $V_{res} = \frac{\lambda}{2 \cdot T_f}$ |
| Angle | $\theta = \arcsin \frac{\lambda \cdot \Delta\phi}{2\pi \cdot l}$ | $\theta_{res} = \frac{\lambda}{N_{Rx} \cdot l \cdot \cos\phi}$ |

Table 2.1: Estimation and resolution equations for main radar parameters: range, Doppler velocity and angle.
Source: Towards Deep Radar Perception for Autonomous Driving: Datasets, Methods, and Challenges [16]

Therefore, methods to extract relevant information from the cube are applied. The purpose of these methods is to identify relevant peaks of energy that describe real objects in a process known as target extraction. Constant False Alarm Rate (CFAR) algorithms follow auto-describing criteria: try to maintain a constant rate of false alarms, meaning that there is a constant rate of *outliers* (noisy reflections that are considered real targets). A whole family of algorithms are created as variants of the original CFAR and all of them differ on performance, computational cost, and the manner to approach the constant false alarm rate criteria. Some of the most common methods are Cell Averaging CFAR (CA-CFAR) [17], Statistical Ordered CFAR (SO-CFAR) [18] or Trimmer Mean CFAR [19]. The result of applying CFAR over a radar cube is a point cloud $R_{PC} \epsilon \mathbb{R}^{N \times 5}$, a collection of $N$ radar targets that belong to real objects, and each of them is described by the following characteristics: distance $r$, radial velocity $v_r$, azimuth $\alpha$ and elevation $\gamma$ angles and RCS (also denoted as $\sigma$).

While most of the learning-based approaches consume point clouds as inputs, in a traditional pipeline there are two main ways to accumulate the information obtained over time and create a spatiotemporal representation of the environment. On the one hand, RADAR point cloud is consumed by a DBSCAN algorithm [20], which performs clustering over the points. At a glance, DBSCAN is a clustering algorithm that allows grouping sets of points that are close in a dimensional space and allows the presence of outliers (points that do not belong to any

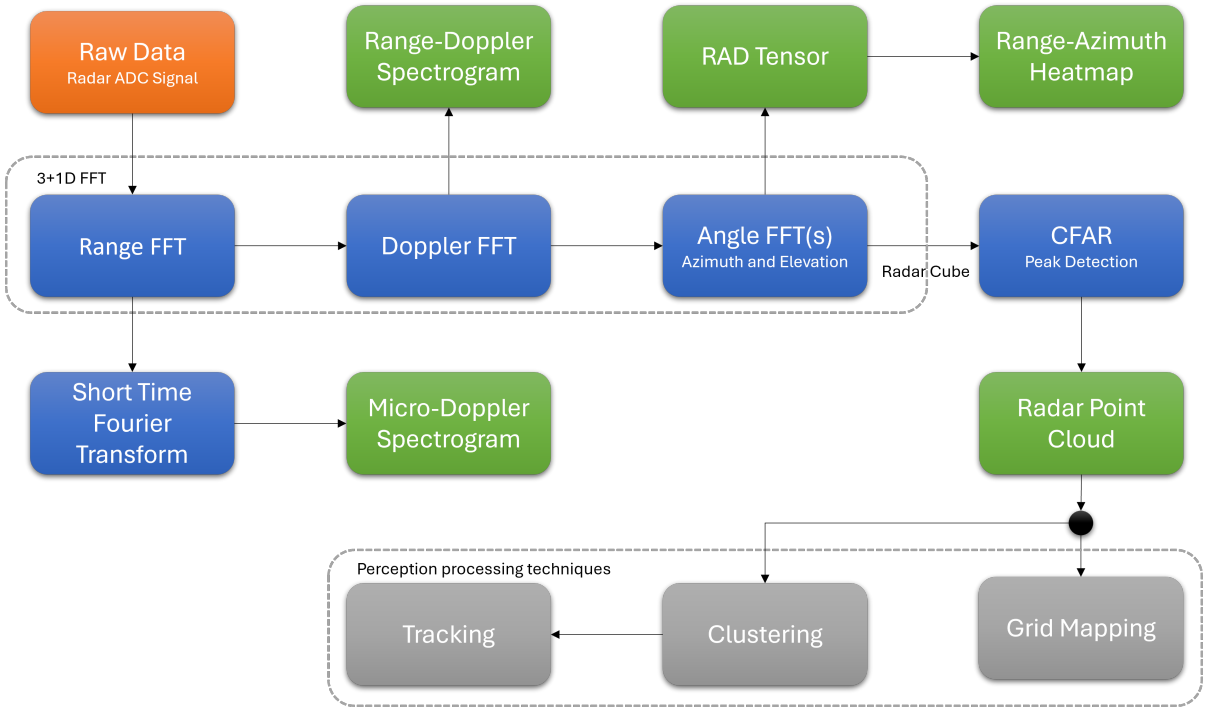



Figure 2.3: A diagram of radar data formats its processing pipeline. From the left-top part of the figure, the radar takes as input raw-data from the signals received by ADC and *Rx* antennas. Low-level processing techniques lead to intermediate representations of the radar data. Lastly, focusing on the radar point cloud, some classical perception techniques for radar are shown, such as grid mapping, clustering and tracking.

cluster) if they are located in non-dense zones, i.e. in the case of RADAR: space, velocity and RCS. The input to the clustering algorithm does not only take advantage of the spatial information. The velocity input takes a major role in separating stationary and dynamic objects that are close in the spatial dimensions. Once these clusters are separated, feature extraction is performed, i.e. extracting the dimensions, averaging velocity values and eliminating outliers. Then, clusters feed a tracking system, which is normally operated by Kalman Filters for state estimation and Hungarian algorithms for data association. On the other hand, the radar targets can feed a dynamic grid map that probabilistically accumulates the detections in spatial cells and updates this grid with new point clouds and the movement of the ego-vehicle. With any of both representations, the main objective is fulfilled: to create a spatiotemporal representation of the static and dynamic objects that surround the vehicle under test.

### 2.3.2. Camera

Cameras have played an important role during the development of ADAS and AD technologies since the beginning of the early experiments of the matter. The inclusion of camera sensors within the robotics and autonomous driving domains is driven by the fact that as humans we navigate the environment guided by our visual observations of the world. As previously mentioned, cameras are passive and exteroceptive sensors that provide a very dense representation of the surrounding environment, in a pixel-level density. Therefore, an image $I$ is a data format

in the form $\mathbb{Z}^{N \times H \times W}$, in which every pixel $i \in \mathbb{Z}$, $0 \leq I \leq 255$. $N$ is the number of channels in an image, which stands for $N = 3$ for coloured RGB images and $N = 1$ for black-and-white grayscale images. $H$ and $W$ are the height and width of the image, respectively, expressed in the number of pixels.

However, the main weakness of cameras is the loss of depth information since all the information is projected over an image plane giving place to a projective representation of the 3D world. This makes single images ambiguous since there are multiple solutions for a 3D point in the space without additional previous information. The combination of a pair of cameras within a stereo vision system permits establishing correlations between the 2D information of two cameras and projecting it to the three-dimensional world following the epipolar geometry principles, knowing a series of parameters *a priori*, such as the baseline $B$ (or distance between cameras in the pair). It is worth mentioning that not all cameras are passive sensors since there are cameras that emit structured visible light patterns to estimate 3D information of the environment within a limited range of non-visible infrared light to work in night conditions. Nevertheless, a further explanation of these types of cameras is beyond the scope of this work.

As cameras are passive sensors, they also perform weakly in scenes with poor light and adverse weather conditions. Fully AD technologies are required to solve any driving situation event, and driving at night or through tunnels are some examples of common driving situations in which a system based only on cameras leads to poor performance of the perception pipeline.

Even with these two main weaknesses, the advantages of cameras are so powerful that they can not be ignored to achieve autonomous navigation. It is the only sensor that can provide a denser representation of the environment in their field of view. Then, they also describe the environment in colour. Colour and the incidence of light over relevant objects provide valuable information about their geometries, and the type of material which leads to the potential to reveal the semantics of the scene.

The agents (pedestrians, vehicles and rest of road users) that surround a vehicle and their main characteristics, the drivable road surface, the traffic regulation (via understanding traffic lights and signals), and the capability to discern a temporal correlation of all these elements are achievable by processing only camera information. This processing is tackled by Computer Vision (CV): the science that studies the processes of acquisition, processing, analysis and interpretation of 2D images from a 3D world.

A typical processing pipeline in a CV system involves the sequential application of these tasks:

- **Imaging and acquisition**: Imaging is the process of forming an image when a sensor detects the radiation (in the form of visible light) that interacts with a physical object in the 3D world. The acquisition of images involves the adaptation of the lighting conditions of the environment and the selection of the camera sensor to take images. For example, in the automotive domain, the front lights of the vehicle allow acclimatization of the environment's lightning for data collection in night conditions. In industrial environments, lighting is easily controllable and can contribute significantly to the performance of an application.

- **Pre-processing**: It consists of applying transformations to an acquired image before it is used. It allows us to perform corrections against known camera distortions, make changes in the colour space, crop an image over a region of interest or perform any other manipulation that will increase the accuracy of the vision system in any task. For example, a normalization and standardization within the colour space will make the CV system more robust against colour variations, while rotating an image (90, 180 or 270°) that feeds a DL-based algorithm, will give it of the condition of rotate-invariant.
- **Feature extraction**: Then, there is a process to transform the raw preprocessed images into information-relevant features that can be consumed by an algorithm, in an analogous way we explained in Section 2.2. In the image domain, extracting features can involve the detection of relevant zones, such as lines, corners or contours for a single image; for a pair of images, it could involve the extraction of the optical flow, which describes the motion patterns of the objects and surfaces within both images.
- **Description and recognition**: Lastly, features are the input to algorithms that describe, recognize or localize regions or objects of interest within the image. With the advent of DL techniques, the stages of feature extraction and description and recognition are performed simultaneously within a neural network. Even though Convolutional Neural Networks (CNN) have not been explained within the scope of this work, it has been shown that there is an inner hierarchy within the learning patterns of these networks. Within [21], it was shown that early stages of a CNN extract low-level features, such as borders, corners or regions. Later stages combine these features to process higher-level features lead to the description and recognition of objects. Therefore, modern CV systems perform these two stages jointly in learning-based approaches.

Some of the most common learning tasks that are applied to images have been previously mentioned in Chapter 1 and will be further explained in Chapter 3. After this brief introduction to cameras, we will delve into their fundamentals and theoretical models.

#### 2.3.2.1. Pin-hole Camera Model

It is necessary to review some foundational concepts about optics to understand the underlying principles of cameras. Since the main objective of this work is to develop a sequential geometric sensor fusion mechanism, it is important to know how to establish relationships between a point in the 3D world and its projection in a 2D image plane.

A camera model is responsible for determining the relationship between the 3D coordinates of a point within a scene and its corresponding 2D coordinates projected to an image. The modelling of a camera is influenced by parameters that are related to the optics (lens type, focal length and distortion), illumination conditions (intensity of the illumination, reflectance of the objects) and geometric conditions (type of projection, position and orientation of the camera).

A **lens** is a transparent object, which is usually made of glass, limited by two surfaces. At least one of them is curved. The curvature of a lens provokes the refraction of light rays that come from outside and form an image in the sensor.

The **focal length**, $f$, is the distance between the optical centre of a lens and the **focal point** when focusing to infinity. It is expressed in millimetres (mm) and it is positive for converging lenses and negative for diverging lenses.

Assuming that camera lenses are thin biconvex lenses, whose thickness is approximately null, a mathematical model is formulated. This model relates the distance from the lens to the object and the distance from the image to the object. This is known as the **fundamental equation of thin lenses**:

$$\frac{n}{s'} - \frac{n}{s} = (n' - n) \cdot \left( \frac{1}{R_1} - \frac{1}{R_2} \right) \tag{2.3}$$

Where:

- $n$, $n'$ are the refraction indices of the lenses within a specific environment. It is an adimensional magnitude.
- $s$, $s'$ are the distances to the object and to the image, respectively. Expressed in meters (m).
- $R_1$, $R_2$ are the curvature radii of the surfaces of the lenses. Expressed in meters (m).

Once the concept of focal distance is introduced within the model, two mathematical assumptions are made. The first one is that the rays coming from an object in a distance $s = f$ are projected parallel to the optical axis $s' = \infty$. The second one is that a ray coming from an infinite distance $s = -\infty$ converges in the focal point of the image at a distance $s' = f'$. Solving the system of equations that if formed when considering these two situations, the fundamental equation can be reformulated in its Gaussian form:

$$\frac{1}{s'} - \frac{1}{s} = \frac{1}{f'} = \frac{-1}{f} \tag{2.4}$$

This allows drawing a similarity of triangles that relates the height of an object, $H$, in the 3D world with its projected height, $h$, on the image plane.

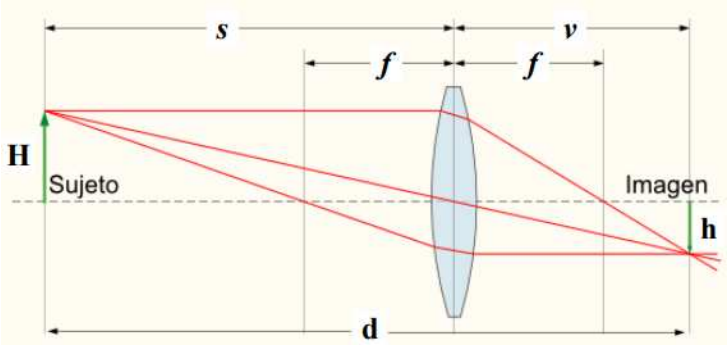



Figure 2.4: Thin lens model geometrical model.
Source: Teaching materials of **Sistemas de Percepción**. Universidad de Alcalá.

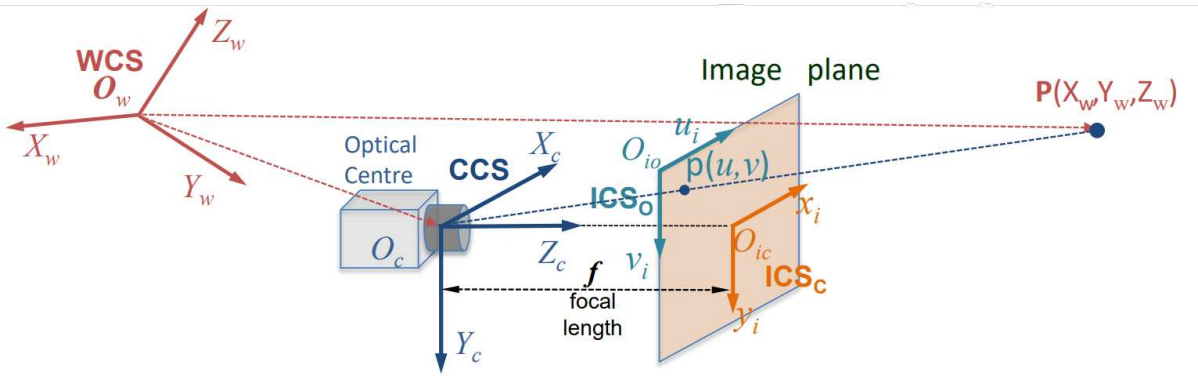



Figure 2.5: Diagram of the transform tree to project a point in the 3D world to the 2D image plane. To fulfil this objective, several geometrical transforms must be applied. A point in the 3D world is located in the World Coordinate System (WCS) and must be translated and rotated to the Camera Coordinate System (CCS). Then, a second transformation transforms the point to the 2D world in the Image Coordinate System in the centre ($ICS_C$). Lastly, a third transform displaces the point to the Image Coordinate System at the Origin ($ICS_O$). Overall, the whole pipeline is: 1. WCS/CCS $\rightarrow$ 2. CCS/$ICS_C$ $\rightarrow$ 3. $ICS_C$/$ICS_O$.

$$\frac{h}{H} = \frac{s'}{s} \tag{2.5}$$

This model implies that a scene projected in a 2D image is going to be inverted and its size is going to be reduced.

The **pin-hole camera model** is a simplification with respect to the thin lenses Gaussian equation. Two additional assumptions are made: the aperture of the lenses shrinks to zero and the image plane is always placed at a distance equal to the focal length, $f$. Any displacement of the plane image will be considered part of the distortion and will be corrected within other mechanisms. This camera model leads to the perspective projection geometry.

#### 2.3.2.2. Image Formation Geometry

Formally, the projection of a 3D point $P = (X_W, Y_W, Z_W)$ in the image plane is the intersection between the line that joins the optical centre $O_C$ with the point $P$ and the own image plane. The projection is fulfilled after making three reference changes that involve four coordinate systems.

1. World Coordinate System (WCS): defined in an arbitrary point in the 3D world. It is characterised by its origin $O_W$ and it is where the point $P$ is originally referenced to.

2. Camera Coordinate System (CCS): a coordinate system defined within the 3D world whose origin $O_C$ is placed within the physical location of the optical centre of the camera. In the automotive domain, CCS has its X component pointing to the right, its Y component pointing downwards, and its Z component pointing frontwards.

3. Image Coordinate System in the centre ($ICS_C$): a coordinate system which is defined within the 2D image plane. Its origin is located in the centre of the image $O_{ic}$. The relationship between CCS and $ICS_C$ is determined by the **focal length**, $f$.

4. Image Coordinate System at the Origin ($ICS_O$): a coplanar reference frame with $ICS_C$. They differ in their origin point, since $ICS_O$ defines its origin within the top-left corner of the image.

The first step involves a reference frame transformation between WCS and CCS frames. To represent a transformation as a matrix product it is necessary to represent the points in **homogeneous coordinates**. A point in a 3D space becomes a four-component vector in the homogeneous space $P_h = (x, y, z, w)$, where $w$ represents a scale factor and is typically fixed to $w = 1$.

In the homogeneous space, a point is composed of a position component and an orientation component. Every transformation between two points in this space can be described by a rotation $R^{3\times3}$ and a translation $T^{3\times1}$. Moreover, every rotation in the 3D space can be described by three fundamental rotations over a 1D space, its main components: $X$, $Y$ and $Z$.

Then, the translation component $T$ from WCS to CCS is:

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

Expressing the rotation $R$ as three fundamental rotations, $\alpha$ radians for X axis, $\beta$ for Y axis and $\gamma$ for Z axis.

$$\mathrm{R}_X(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

$$\mathrm{R}_Y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

$$\mathrm{R}_Z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

The total transformation is described as $M = R_X(\alpha) \cdot R_Y(\beta) \cdot R_Z(\gamma) \cdot T$. The resulting matrix $M$ is:

$$M = \begin{pmatrix} R^{3\times3} & T^{3\times1} \\ 0^{1\times3} & 1^{1\times1} \end{pmatrix} \tag{2.10}$$

Where $M_{ext} = \begin{pmatrix} R^{3\times3} & T^{3\times1} \end{pmatrix}$ is known as the **extrinsic parameters matrix**.

The second step involves a transformation between CCS and $ICS_C$ which is also known as the perspective projection equations. Applying the similarity of triangles, a 3D point $P = (X_C, Y_C\, Z_C)$ referenced to the optical centre $O_C$ can be projected into the image plane. In this step is where the loss of depth information occurs, since this model assumes that all points are at a distance $z = f$. The triangles obtained for both axes are:

$$\frac{y}{f} = \frac{Y_C}{Z_C} \rightarrow y = f \cdot \frac{Y_C}{Z_C} \tag{2.11}$$

$$\frac{x}{f} = \frac{X_C}{Z_C} \rightarrow x = f \cdot \frac{X_C}{Z_C} \tag{2.12}$$

Alternatively, this can be expressed in its matricial form:

$$\begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{pmatrix} \tag{2.13}$$

The third and last step involves the reference frame change between $ICS_C$ and $ICS_O$ in a conversion from a metrical world to a pixel world, and an offset that moves the origin point from the centre of the image to its top-left corner.

While a point in $ICS_C$ was in a metrical form $p = (x, y)$, a point in $ICS_O$ is in a pixel form $p = (u, v)$. In $ICS_O$, $(u_0, v_0)$ represents the pixel coordinates of the intersection between the optical centre and the image plane, and $(dx, dy)$ represents the dimensions in millimetres of a unitary pixel. Applying the similarity of triangles:

$$u = \frac{x}{dx} + u_0 \tag{2.14}$$

$$v = \frac{y}{dy} + v_0 \tag{2.15}$$

And expressed in matrix way:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = \begin{pmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} \tag{2.16}$$

The set of parameters $(u_0, v_0, f, dx, dy)$ is known as **intrinsic parameters** and describes the internal behaviour of the camera sensor. These parameters can be also expressed in matrix form via the **intrinsic parameters matrix**, $M_{int}$.

$$M_{int} = \begin{pmatrix} \frac{f}{dx} & 0 & u_0 \\ 0 & \frac{f}{dy} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.17}$$

Finally, to tie up all the loose ends explained in this section, the whole projection $M$ is the product of the intrinsic and extrinsic parameters matrices, resulting in:

$$\begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = M_{int} \cdot M_{ext} \cdot \begin{pmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{2.18}$$

#### 2.3.2.3. Stereo Camera Model

To end up with the introduction to camera and images, it is necessary to briefly explain some concepts about **stereo vision**, since the calibration models in real automotive datasets are a hybrid between a typical monocular calibration and a stereo one. A stereo vision system consists of a pair of cameras and their development is motivated by the fact that correspondences between images (a point $p$ for an image and its correspondent partner $p'$ in another image) can lead to reconstructing the scene geometry: to find the 3D coordinates of a point from multiple projections on different images.

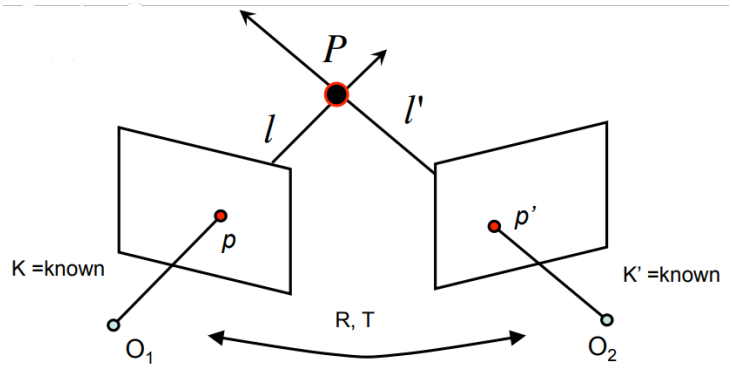



Figure 2.6: Triangulation process between a pair of images of a stereo vision system. Source: Teaching materials of **Sistemas de Percepción**. Universidad de Alcalá.

Correspondences between points are found via triangulation, which is shown in 2.6. **Epipolar geometry** sets a series of geometrical relations that enable the correspondence finding. The most important concepts of epipolar geometry are:

- Baseline ($B$): is the distance between the optical centres of the cameras.

- Epipoles ($e_1$, $e_2$): are the points in which the baseline intersects with the image planes.
- Epipolar plane: is the plane in the 3D space which contains the baseline and the 3D point for which correspondences are been searched. This plane allows to establish a geometrical relation between the two cameras and the point.
- Epipolar line: is the line that is generated between the two projections over their corresponding image plane of a point in the 3D space.

The development of this geometry allows to recover the 3D information of a point that was lost using a single camera. Some other important concepts are the **Essential** $E$ and **Fundamental** $F$ matrices. Since this discipline involves heavy matrices computation, it is helpful to use techniques such as Singular Value Decomposition (SVD). Both of these matrices allow the mapping of correspondences from one image plane to another. While the first one performs the correspondence in the pixel world, the second one is thought for the metric world. Computing all correspondences among possible points leads to the calculation of a disparity map (D), which represents the variation in pixel coordinates of all points present in both images from one plane to another.

In order to compute depth values from a stereo vision system, the image planes of both lenses must be aligned within both $u$ and $v$ axes. Therefore, they must be parallel, as shown in Fig. 2.7. Then, knowing the baseline (B), the focal length (f) from the optical centres (O, O') to the image planes and the disparity map (D), the depth values (Z) can be computed as follows:

$$Z\,(u,\,v) = \frac{f_x \cdot B}{D\,(u,\,v)} \tag{2.19}$$

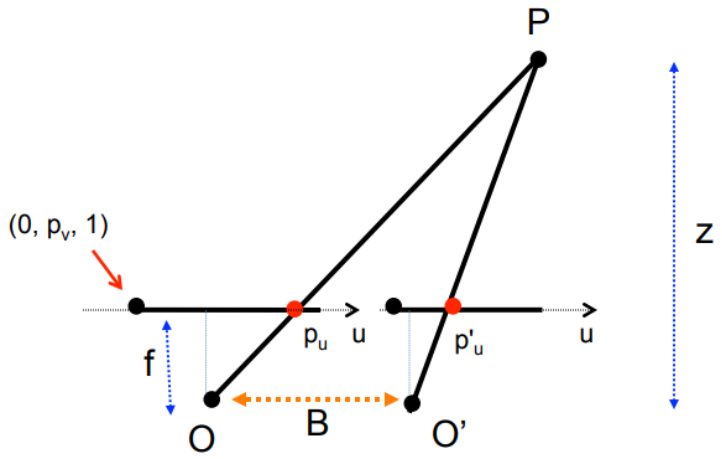



Figure 2.7: Computing depth value from a disparity map in a stereo vision system. Source: Teaching materials of **Sistemas de Percepción**. Universidad de Alcalá.

# Chapter 3

# State of the Art

*Don't count the days, make the days count.*

Muhammad Ali

## 3.1. Introduction

The purpose of this Chapter is to perform a systematic review of the most influential DL architectures that appeared during the last years and tackle the two main tasks involved in this project: 2D Object Detection/Instance Segmentation for images and 3D Object Detection for point clouds, the so-called State of the Art (SOTA). This will serve to introduce the reader to the inner workings of these architectures.

## 3.2. 2D Object Detection and Instance Segmentation

Understanding the fundamental concepts behind the tasks we are dealing with in a supervised learning problem has allowed us to characterize how the inputs and outputs of the networks will be. Delving within the nature of the task will also allow us to characterize the anatomy of an architecture, which is shared by most of the SOTA methods. Then, the aim of recent methods is to improve these components in order to obtain a better overall performance. Remembering, an Object Detection problem expects as input an RGB image and returns as its output this data structure for each of $D$ detections:

- A **probability class** vector that indicates to which semantic class the object belongs. Its form is $N \times 1$, being $N$ the number of classes in a dataset. All probabilities must sum up to 1. With this vector, two outputs are computed: the object class (applying `argmax` operation) and the confidence score (applying `max` operation) which is a measure of the likelihood of the detection.
- A **bounding box** that indicates the localization of the object of interest. Its form is $4 \times 1$, being these 4 values the top-left corner of the box ($x$ and $y$), its width and height.

The extension of an object detection problem to an instance segmentation one involves the prediction of a **mask** that encircles the object of interest, so the final output will have another component.

- A *black-and-white image* in which white pixels indicate the presence of the **mask**, whose form is $H \times W$.

Given the input RGB image and the structure of outputs desired, and taking into account that the objective is to localize objects (which are smaller regions of the image), the anatomy of a DL architecture for both of these tasks will be composed of:

1. A **backbone**, which will be in charge of extracting features to comprehend the context of the image.
2. A **neck**, which will be focused on proposing particular regions of interest given the extracted features and obtaining multi-scale representations of the image.
3. A **head**, which will compute the output data structure recently explained: class, confidence, bounding box and mask.

Provided with all the fundamental concepts, the intuitions behind the problems and all the necessary building blocks to compose an architecture, we will delve into a brief historical and SOTA review of the most relevant methods.

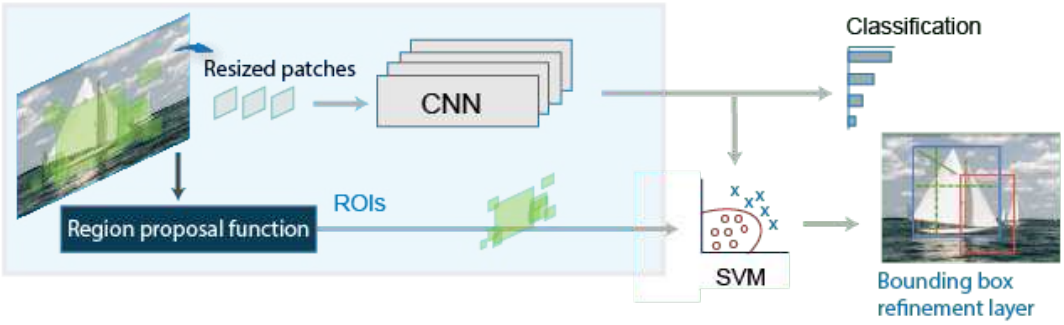
### 3.2.1. Methods

The history of Object Detection methods [22] starts in the pre-neural era (around 2000). These algorithms relied on traditional image processing techniques and the three most relevant were: the Viola-Jones detector [23], the Histogram of Oriented Gradients (HOG) detector [24] and the Deformable Part-based Model (DPM) [25]. A common characteristic among them is that they are specialized in detecting certain parts: faces or persons. They are able to introduce domain-specific knowledge and rules to perform well in a little niche task, but they lack generalization or capabilities to detect other types of objects, a reflection made in Chapter 2.

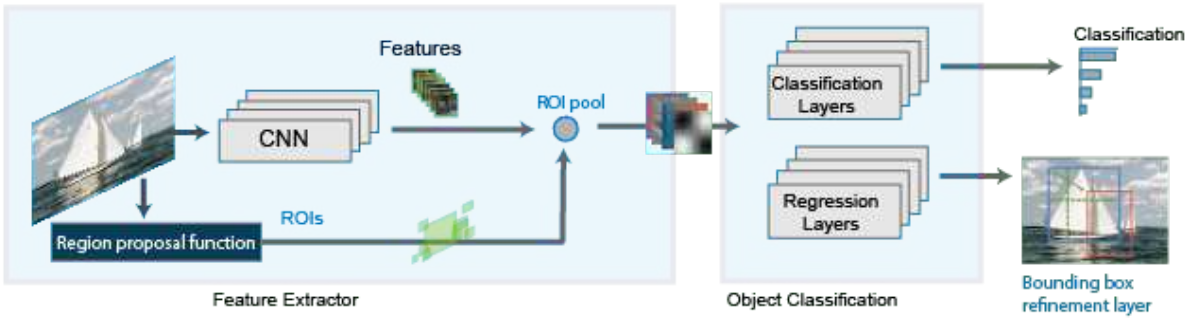
Then, the success of DL architectures applied to image classification at the ImageNet 2012 challenge made the research community put their hands on creating architectures for other tasks, such as Object Detection. This led to two research directions: single-stage detectors and two-stage detectors.

#### 3.2.1.1. Two-stage detectors: from R-CNN to Mask R-CNN.

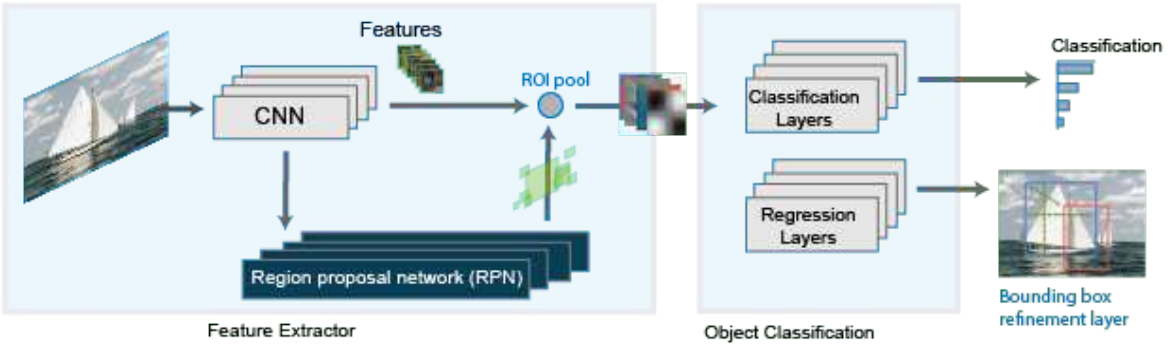
The Deep Learning era of Object Detection started in 2014 when Region Based Convolutional Neural Networks (R-CNN) [26] rise. An overview of R-CNN architecture can be seen in Figure 3.1a. Girschik *et al.* propose to extract a set of regions that might contain an object, called region



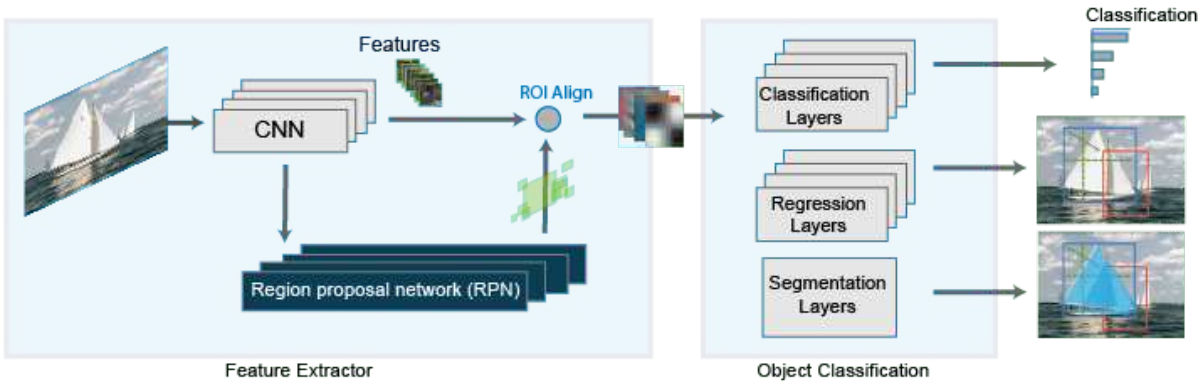

(a) General overview of R-CNN.




(b) General overview of Fast R-CNN.




(c) General overview of Faster R-CNN.




(d) General overview of Mask R-CNN.

Figure 3.1: The evolution of R-CNN family: from R-CNN to Mask R-CNN.
Source: `mathworks.com`. Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN (a, b, c) and Getting Started with Mask R-CNN for Instance Segmentation (d).

proposals, using a Selective Search [27] or an Edge Boxes algorithm [28]. Then, these regions are cropped and resized and they feed a CNN that acts as **backbone** and extract features. Features feed a set of class-specific linear Support Vector Machine (SVM) classifiers and they predict a bunch of bounding boxes that are filtered by a Non Maximum Suppression (NMS) algorithm. The paper reported that during test-time evaluation, 2000 region proposals were extracted, which led to an extremely low implementation with an inference time of approximately 14 seconds per image using GPU. Moreover, the proposed method reported an increment of 16% in mAP (a metric which will be explained in detail later) in the reference dataset of that time, PASCAL VOC 2007 [29].

In 2015, Girschik [30] proposed an evolution of its preliminary architecture, directly dealing with the main disadvantages shown. There are two main novelties in Fast R-CNN. Firstly, the introduction of a modern **head** based on NN layers. The SVM-based mechanism is substituted by a bounding box regressor, which follows up the modern output data structure explained before. Secondly, the backbone extracts features of the whole image instead of the region proposals separately. Therefore, a region of interest pooling mechanism is introduced to align the features with the region proposals. The two novel mechanisms led to an inference time 200 times faster and an improvement of 15.5% mAP in PASCAL VOC 2007, reaching a total score of 70.0% mAP.

Later that year, Ren *et al.* [31] addressed the question: *Can we generate region proposals with a CNN model?*. This is where Region Proposal Networks (RPN) are proposed. This led to another improvement in both run-time (17 fps) and metrics (mAP = 73.2% in PASCAL VOC 2007). The three main components of a detector were based on NNs and they become an end-to-end DL structure. Moreover, modern detectors integrated RPNs within another component: the **neck**. In 2017, Lin *et al.* [32] proposed Feature Pyramid Networks (FPN). Object detection has to deal with both: classification and box regression. While the deeper layers of the backbone have considerable classification potential, the resolution is decreased and the potential for box regression is impoverished. While previous approaches only performed detection on the final layer of the backbone, FPNs route lateral connections from previous layers to the head. This allows captioning details in multiple scales. The overall improvement was clear, from 42.7% mAP@0.5 in vanilla Faster R-CNN to 59.1% in Faster R-CNN with FPN neck in the COCO dataset. From this day on, the typical structure of a detector follows this approach: backbone, neck and head.

Lastly, He *et al.* [6] introduced a mask regressor within the head of Faster R-CNN, allowing Instance Segmentation with the same architecture, and presented Mask R-CNN. Experiments in this paper were made with ResNet [33] and ResNeXt [34] backbones over the COCO dataset. The method outperformed the previous approaches not only within instance segmentation but also in human key-point estimation.

Overall, the **two-stage** naming comes from the combination of RPN with the region of interest pooling or align mechanisms that are in charge of generating the bounding box proposals. Nowadays, Mask R-CNN is one of the most methods used methods for instance segmentation tasks. Some other variants are also popular, but are more focused on accuracy and have slower

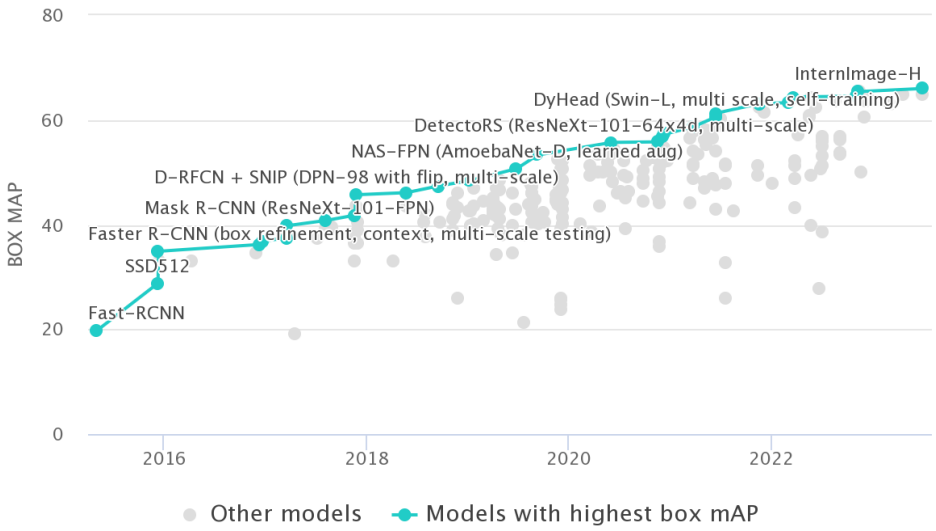



Figure 3.2: Evolution of mAP with respect to time for 2D Object Detection on COCO test-dev dataset.
Source: paperswithcode.com. Object Detection on COCO test-dev.

inference times such as Cascade Mask R-CNN and Hybrid Task Cascade R-CNN. As mentioned before, improvements in Mask R-CNN have been done from its release on as a consequence of the release of more modern building blocks that are introduced as backbones or necks.

#### 3.2.1.2. Single-stage detectors: the YOLO family.

One of the main disadvantages of two-stage detectors is that their inference time is not enough to be deployed in real-time environments, such as embedded devices, which require between 25 and 30 Hz at minimum. In response to this requirement, Redmon *et al.* [35] proposed the first version of one of the most relevant family of architectures in DL history: YOLO. The following approach models detection as a regression problem: the image is divided in a $S \times S$ grid and predicts simultaneously $B$ bounding boxes with confidences and a class probability for each grid cell. Each bounding box encodes its coordinates and the confidence of its prediction: $B = (x, y, w, h, c)$ Consequently, the final predictions are encoded in a $S \times S \times (5 \cdot B + C)$. This is exemplified with PASCAL VOC 2007 dataset, where there are 20 labelled classes $C = 20$ and the authors chose a square grid of 7 cells and 2 bounding boxes per cell, $S = 7$; $B = 2$, the output tensor is $7 \times 7 \times 30$. This behaviour is graphically represented in Figure 3.3.

These decision choices lead to a fixed number of bounding boxes equal to $S^2 \times B$. As there is not a fixed number of objects in an image, we could consider this number as the maximum number of possible detections, since a post-processing stage is applied to take only the relevant detections. Due to the nature of the network, there will be multiple overlapping predictions over the same object of interest and *trash* predictions in zones with no particular objects. This stage involves the application of confidence thresholds and Non Maximum Suppression to the set of detections and after applying both there will remain only the relevant predictions.

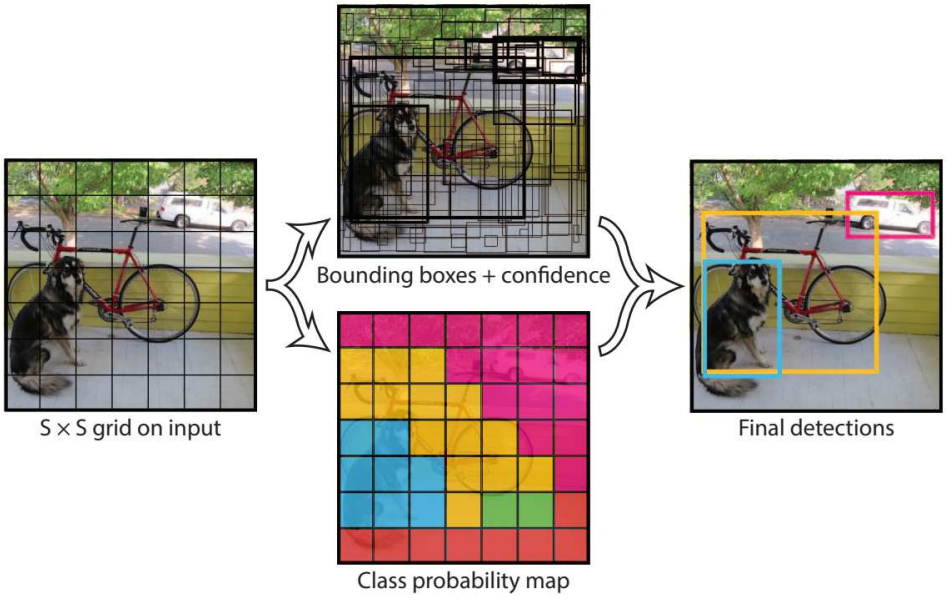



Figure 3.3: How YOLO works: simultaneous regression of bounding boxes and semantic classes per grid cell.
Source: You Only Look Once: Unified, Real-Time Object Detection [35].

The architecture of YOLO is a succession of 24 convolutional layers which end up with 2 dense or fully connected layers. Leaky RELU activation functions were applied to all layers except for the last one, which is a regression, so they applied a linear activation function. It is designed to support as input square RGB images whose height and width are 448. The use of dense layers at the final stage of the network fixes the input size. One of the major disadvantages of YOLO is that it had difficulties in predicting objects with different aspect ratios to the ones seen during the training stage.

This proposal leads to a very lightweight implementation that ran at 45 Hz in its early days. More modern versions of YOLO can reach inference frequencies of 120-160 Hz. Inference time and accuracy are two parameters that suffer from a *trade-off* in these architectures. Generally, faster methods will perform slightly worse than the best-performing methods, which will not be able to reach real-time inference speeds.

The same author proposed YOLOv2, YOLO9000 [36] and YOLOv3 [37]. YOLO9000 was capable of detecting 9000 different semantic classes using a hierarchical classification method, based on the applications of multiple softmax activation functions over a tree-like structure until a final leaf is reached. The improvements done in these two versions include the introduction of batch normalization for regularization, the removal of the dense layers, and the introduction of anchor boxes and prior dimension clusters. Also, YOLOv3 included a new backbone, Darknet53, a Spatial Pyramid Pooling (SPP) block which increased the receptive field of the network, and a new decouple head that performed multi-scale predictions.

The multi-scale prediction mechanism makes the output have three branches. Each branch is connected to a different feature map, each one with different resolutions so their predictions will be focused on objects of different sizes in the original images. Final predictions are merged using NMS.

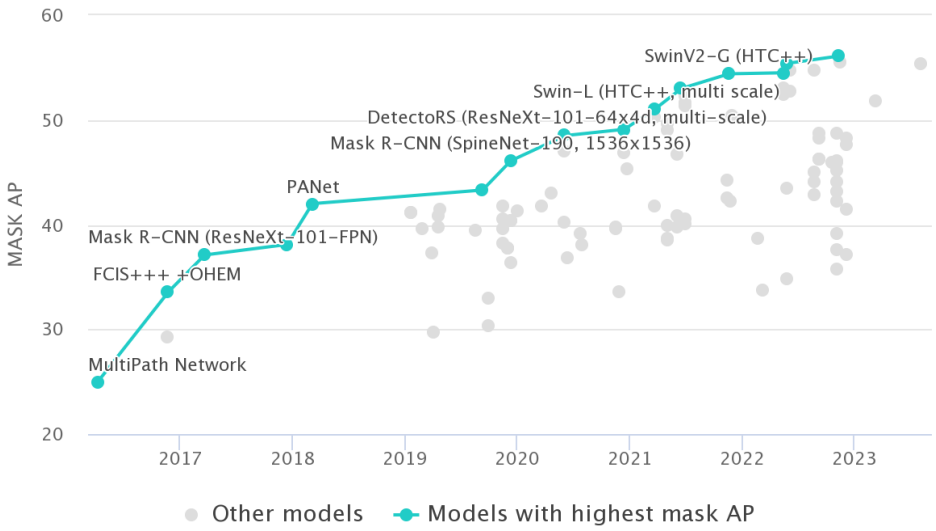



Figure 3.4: Evolution of mAP with respect to time for 2D Instance Segmentation on COCO test-dev dataset.
Source: paperswithcode.com. Instance Segmentation on COCO test-dev.

Discontinued by the original authors, several parallel research branches appeared. During the rest of this review, we will focus on YOLOv5 and YOLOv8. Glenn Jocher and its company, Ultralytics, started their work rewriting YOLOv3 in modern PyTorch. The architecture was slightly modified and the main improvements were focused on reducing memory and computational costs: stride convolutions in a new backbone (CSPDarknet53), a fast SPP block and some modernizations such as SiLU activation functions and batch normalization. Lots of work was put into engineering tasks, such as including a pre-training phase with an *auto-anchor* mechanism, several data augmentations and customizable parameterizations to improve the training stage and several export tools to maximize the number of possible devices in which the model can be deployed. YOLOv5 became easy to train, use and deploy by every DL practitioner so it became one of the most famous architectures. It offered 5 model versions that vary in size: *nano* (n), *small* (s), *medium* (m), *large* (l) and *extra-large* (l) with two pre-trained checkpoints for each one in 640 *px* and 1280 *px*. In addition, YOLOv5 began a task expansion project and they started supporting not only object detection but also instance segmentation and image classification.

Lastly, YOLOv8 was released in January 2023 and offers again 5 size variants (with one only pre-trained checkpoint for 640 *px*) and support for multiple vision tasks: classification, object detection, instance segmentation, key-points pose estimation and object tracking. At the time of writing this document, work is put on supporting more tasks such as rotated/oriented object detection. Among the main improvements of YOLOv8, you can find:

- Anchor-free detections. From 2023 on, there is a trend for real-time detectors to remove anchor mechanisms.
- Use of novel loss functions for training, such as Complete IoU (CIoU) and Dual Focal Loss (DFL) for bounding box and classification losses. Removal of confidence loss.

- YOLOv8 for segmentation includes an FCN-like decoder called Proto and a segmentation head that receives as input the boxes filtered by NMS.

After all this review, Figures 3.2 and 3.4 reflect the evolution of SOTA with respect to time for object detection and instance segmentation, respectively. As can be observed in both charts, two-stage detectors such as Mask R-CNN were leading and top-performing during the early days of DL for CV. Several variants were Top 1 leaders, delving into the idea we have seen during this Chapter: once an architecture is established, there are incremental improvements attacking their three main components (backbone, neck and head). In parallel, attention mechanisms [38] were being developed within the Natural Language Processing (NLP) domain. With the advent of Vision Transformers [39] and Swin Transformers backbones [40] all the top-performing (in consequence, slightly slower) methods decided to go through the path of including attention-based building blocks in their architectures. Attention mechanisms for object detection will be the natural continuation of this review, but as we will not cope with them, we postpone it for future works.

### 3.2.2. Metrics

#### Mean Average Precision (mAP)

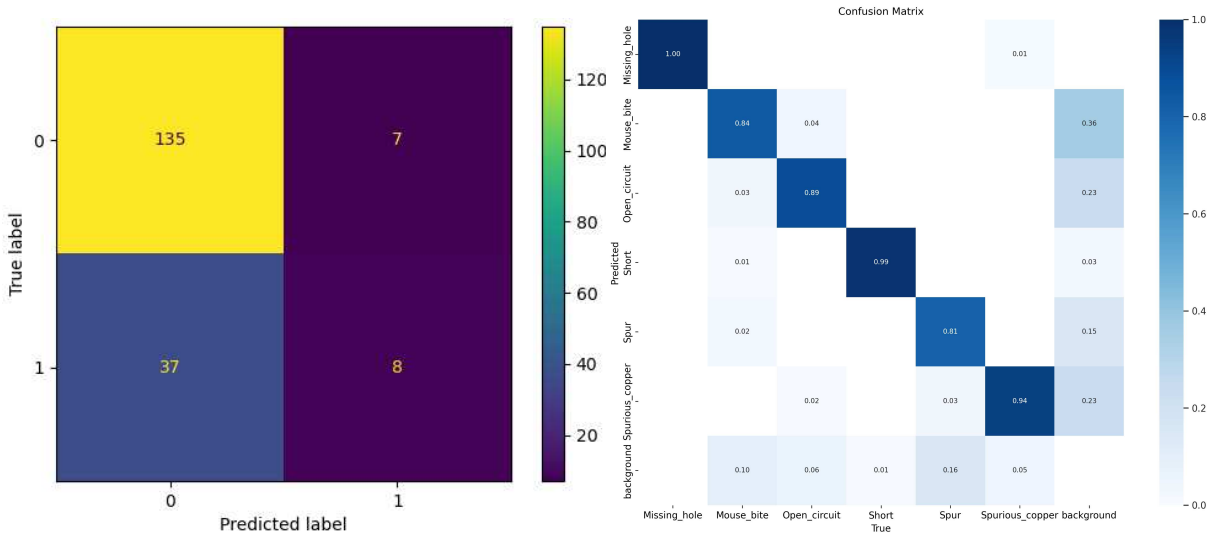
Mean Average Precision (mAP) [41] is the most common metric used to assess the goodness or performance of object detection and instance segmentation models in images. It is formed by the combination of several different metrics: Confusion Matrix (CM), Intersection over Union (IoU), **precision**, **recall** and **precision-recall curve**. In a nutshell, to compute mAP it is necessary to follow a seven-step procedure, each of these steps is:

1. Perform an inference over all the samples in the dataset and compute the prediction scores.
2. Transform the prediction scores to class labels.
3. Build the Confusion Matrix comparing the set of predictions with the set of ground-truth objects via Intersection over Union.
4. Compute precision and recall.
5. Compute the Area Under the Curve (AUC) for the precision-recall curve.
6. Measure the Average Precision (AP).
7. Weight the average precision for each class over the total number of classes.

We will dive through all these steps, as mAP is at the same time, it is one of the most used metrics and whose internal workings are less known. Firstly, it is necessary to perform an inference over all the samples within the data split that want to be evaluated. The output of a model for an object detection task includes a uni-dimensional array with the probabilities to belong to each class present in the dataset. All these probabilities must sum up to 1.

Step 2 involves taking the index of the class with maximum probability (through an `argmax` operation), so the prediction score is transformed into a class label.

A Confusion Matrix or Error Matrix is a graphical representation of the performance of an algorithm dealing with a supervised classification task and applied over a determined dataset. For a binary classification task, given a set of samples, where we have a label-prediction pair for each, we must classify each pair into four categories: True Positive (TP) if a prediction and labels are positive, True Negative (TN) if both are negative (which is not possible in an object detection task since the absence of label implies the absence of a prediction for this particular object), False Positive (FP) if a prediction is done for no label, and False Negative (FN) for a label without matching correct prediction. These four categories are graphically arranged in a $2 \times 2$ matrix. From the hypothesis testing point of view, we can consider FPs as Type I errors and FNs as Type II errors. Expanding the CM to a multi-class classification problem, the information will be arranged in a $N + 1 \times N + 1$ matrix, being $N$ the number of classes in a dataset. Therefore, with a multi-class CM is also possible to distinguish those sets of classes that are more prone to being confused between them. An example of a binary classification matrix (3.5a) and a multi-class classification confusion matrix (3.5b) can be seen in Figure 3.5.




(a) Binary confusion matrix (b) Multi-class confusion matrix

Figure 3.5: An example of confusion matrices for binary and multi-class classification tasks.

But, in the context of object detection, it is mandatory to define an additional criterion to confirm that a prediction corresponds to a label. IoU is a geometrical concept that represents the quotient between the overlapped polygon and the polygon formed by the union of a pair of bounding boxes. The IoU for a bounding box with itself (perfect overlap) will output 1, whilst the IoU for a pair of boxes with no overlap region will output 0.

$$IoU\,(A,\,B) = \frac{|A \cap B|}{|A| \cup |B|} \tag{3.1}$$

Introducing IoU into mAP leads to new variants of the metric. For example, mAP@0.5 represents the metric with a single validation threshold at IoU $= 0.5$. Another example is

mAP@ $[0.5 : 0.05 : 0.95]$, which computes the metrics for a range of thresholds, from 0.5 to 0.95 with a step of 0.05. It is worth mentioning that the second variant is much stricter than the first one since it takes into account a higher value of IoU, which requires the model to make *quasi-perfect* overlapped predictions.

Then, precision and recall can be computed for each class separately attending to their definitions. On the one hand, **precision** is the quotient between the set of correct predictions and the set of positive labels for that class. On the other hand, **recall** is the quotient between the set of correct predictions and the set of positive predictions (correct and incorrect ones).

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

From a higher-level abstraction, precision answers the question *"How often does the model predict correctly?"*, whilst recall answers *"Has the model predicted what it was supposed to predict?"*. While the first one focuses on the success rate, the second one focuses on the absence of errors.

With precision and recall values it is possible to build the **precision-recall curve** for each class. The construction of the curve requires as input the list of predictions for a single class and being classified in TP or FP (i.e. via an IoU threshold). Firstly, the predictions must be sorted by decreasing confidence, from the maximum to the minimum. Then, the aggregated values of precision and recall are calculated following this order. This operation will output $P$ pairs of precision-recall values, being $P$ the number of predictions that a model has made over the whole dataset. An important point is that the recall will follow a monotonical increasing tendency. Precision will increase and decrease without any pattern, a phenomenon that will produce sharp variations in the curve that highly increase the sensitivity of the computation. These pairs are plotted into a two-dimensional axis in which the X-axis represents recall and the Y-axis represents precision, forming the curve. Then, the ideal Area Under the Curve (AUC) is computed by taking integrals from 0 to 1.

$$AUC = \int_0^1 Precision\,(Recall) \cdot d\,(Recall) \tag{3.4}$$

To alleviate computational demand and to reduce the impact of sharp variations in the curve, ideal AUC is not used in practice. Instead, Average Precision (AP) is an approximation that interpolates the curve within 11 or 40 points. The original approach was to use 11 points from 0 to 1 in steps of 0.1. Most modern approaches divide the range from 0 to 1 in 40 equidistant steps. The interpolated precision values are obtained by taking the maximum precision whose recall value is greater than its current recall.

With all these elements, AP can be computed.

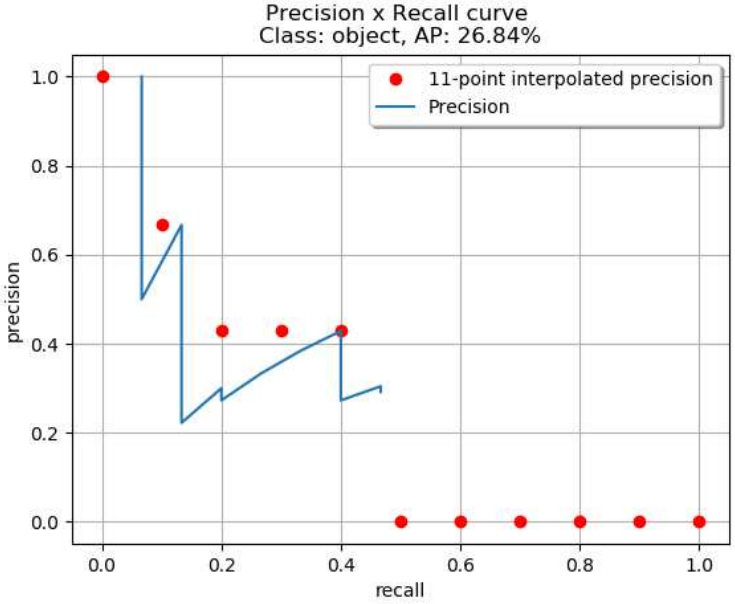



Figure 3.6: Original precision-recall curve vs. 11-point interpolation curve.

$$AP = \frac{1}{11} \sum_{k=0}^{k=10} Precision\,(0.1 \cdot k) \tag{3.5}$$

And lastly, averaging through all classes, mAP is also computed as:

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \tag{3.6}$$

As a last remark, when we introduced this metric it was stated that is one of the most commonly used among object detection and instance segmentation tasks. Nevertheless, when mAP is applied in instance segmentation, the criterion to consider a prediction as TP varies. Instead of using IoU among pairs of boxes, IoU is applied pixel-wise between pairs of masks. To differ between them, modern datasets report mAP(B) for object detection and mAP(M) for instance segmentation.

## 3.3. 3D Object Detection in Point Clouds

Once we have gone through an extensive review of object detection for images, it will be relatively easier to introduce the task of object detection for 3D point clouds. These point clouds will come from LiDAR or RADAR sensors. The research on methods for this task has been mostly done with LiDAR data. The work with RADAR data was particularly limited due to the low resolution of the previous generation sensors. Recently, a study has tested the methods developed for LiDAR on a 4D RADAR dataset. Alternatively, there are lines of study that focus on specific architectures for traditional RADAR point clouds. The gap between using LiDAR

or RADAR data is very large, so the advent of 4D RADAR represents an opportunity for new architectures that have a higher performance from better quality data with velocity information.

Therefore, this task starts taking an input from a point cloud, which is a tensor in the form $N \times F$, where $N$ is the number of points and $F$ is the number of features. Features will include spatial localization in $x$, $y$, $z$ coordinates, the intensity of the reflection or its RADAR equivalent, RCS, radial velocity in the case of RADAR, and a time dimension which will reflect the temporal relations between points if point clouds from consecutive time stamps are accumulated. The output will be a set of oriented bounding boxes in the 3D space, in the form $B = (x, y, z, dx, dy, dz, \theta)$, where $(x, y, z)$ are the coordinates of the centroid of the bounding box, $(dx, dy, dz)$ are the dimensions of the box and $\theta$ is the orientation of the box with respect to the $Z$, upwards, axis.

### 3.3.1. Methods

Once the inputs and outputs of the task have been characterised, a selection of existing methods of the current literature will be reviewed. Methods for 3D object detection can be categorised by the way they represent the point clouds, which will lead to a specific set of models: projection-based, volumetric and point-based methods. Additionally, a fourth research brand focused on 2+1D RADAR will be mentioned.

- **Projection-based methods**: As a more complex task and appearing later in time, this research line aims to take advantage of the developments made within 2D object detection and projects the point cloud into an image plane via perspective, cylindrical or spherical projections. On one hand, fast real-time computations are enabled because it is possible to apply variants of the known 2D object detection one-stage architectures. On the other hand, the compression of depth into an image plane difficult the estimation of shape and orientation for the bounding boxes. Another disadvantage is that the detection of occluded objects is difficult within a 2D image plane. Another possibility is to project the point cloud into a Bird's Eye View (BEV) projection and encode the height into the image channels.

- **Volumetric (voxel-based) methods**: The key concept behind this research line is that an unstructured input point cloud is discretized into a volumetric 3D grid, in which each unitary cell is known as **voxel** (**vol**umetric pix**el**). Therefore, the whole 3D space is discretized and each voxel is populated with the points that lie inside it. After populating the voxels, each one will be represented by a unique vector of features, which are extracted by manipulating the data points, and their spatial coordinates. The main advantage of these methods is that they maintain a spatially coherent representation of the environment with minimal information loss. However, there are two big disadvantages. Firstly, due to the nature of LiDAR point clouds (and even more accentuated on RADAR ones), the information is sparsely distributed through the space. This means that a high percentage of voxels will be empty and will not contribute relevant information. Secondly, applying

CNNs to a 3D data structure involves the usage of 3D convolutions, which are heavily costly in terms of computation effort.

- **Point-based methods**: Lastly, point-based detection methods consume the point cloud in raw format, as a tensor, without any spatial information loss. They will present high sparsity and computational cost due to the usage of 3D convolutional layers, as happened within the voxel-based methods.

During this review, we will observe that some architectures try to combine the advantages of various of these research lines (i.e. point-based and voxel-based), creating hybrid approaches. Moreover, analogously to 2D detectors, we can categorise 3D detectors into one-stage or two-stage detectors.

#### 3.3.1.1. Projection-based methods

**Complex YOLO** [42] expanded the work made in YOLOv2 by adding a regression strategy to estimate multi-class 3D bounding boxes in the Cartesian space. In this case, a LiDAR point cloud is colourized by back-projection of the image and projected to BEV. Then, a CNN based on YOLOv2 with a modified E-RPN to propose regions in the Euler space performs detection in a grid-like system (as explained in previous sections). The post-processing stage performs a regression for the angle of the oriented bounding box. The main contribution of this paper is that they achieved the first architecture to process LiDAR point clouds in real-time, at 50 Hz. The model is trained and evaluated in KITTI [8]. Another approach based on YOLOv2 for 3D-oriented object detection is **YOLO3D** [43].

Beltrán *et al.* proposed a framework called **BirdNET** [44] in 2018. It consisted of the application of a 2D object detector, Faster R-CNN with RPN, over a LiDAR point cloud projected to BEV. The main contribution of the paper lies in the cell encoding protocol, whose purpose is to perform a density normalization over the beams. Moreover, the head component of the detector is modified to support a third branch that estimates the possible orientation of the bounding boxes. Later, a preprocessing stage refines the bounding box orientation and estimates the heights of the object, supported by a ground plane estimate. Later, Barrera *et al.* extended this work in **BirdNet+** [45] performing end-to-end detection relying solely on Faster R-CNN and no post-processing stage. Both models were trained and tested on KITTI [8].

In order to take advantage of multiple projections of the LiDAR, *multi-view* approaches appear. Chen *et al.* proposed **MVLidarNet** [46] in 2020 consisting of two consecutive networks schema: a semantic segmentation of the point cloud within a spherical projection within the image plane and a BEV projection of the segmented point cloud. The first network is an optimized and lighter encoder-decoder based on FPN. The second network receives as input the sampled BEV image and the semantic-coloured image in two different branches that are fused and share the decoder. The term *encoder-decoder architecture* is another way to express the anatomy of a NN. It receives that name because it receives as input an image-like tensor and the backbone or encoder is in charge of compressing the image and extracting the most relevant

features, as usual. The decoder varies from the neck-head concept explained before. This type of network also outputs an image-like tensor and is in charge of decompressing or decoding the features to create the desired representation. The output of the second network is a BEV image with features that may represent a bounding box. Then, DBSCAN clustering is applied over this image to extract the relevant objects and, also, the free-driving space. For this work, the authors trained the first stage in SemanticKITTI, and the second one in KITTI, [8]. Running both networks has an inference time of 7 ms and performs competitively alongside more expensive architectures in KITTI.

Overall, the projection-based methods aim to obtain higher inference times at the expense of slightly worse performance. Moreover, a research trend in which novel methods from 2D Object Detection are imported into this task has been identified. This practice allows the fast adoption of the spearhead of research for this particular application. Another example of this trend can be seen in the case of Fully Convolutional One-Stage (FCOS) [47], a novel architecture that treats object detection as a per-pixel task (similar to semantic segmentation). The same authors released the original method in 2019 and they adapted it to 3D object detection with LiDAR in a perspective projection view in 2022 [48].

#### 3.3.1.2. Voxel-based methods

Object detection in point clouds is intrinsically a three-dimensional problem. Hence, it is natural to deploy networks based on 3D convolutions for this problem. The first voxel-based methods were also imported from the 2D vision domain. In this case, Li presented **3DFCN** [49], an adaptation of FCNs to the automotive field with LiDAR point clouds. The method takes a discretized voxel representation of the point cloud as input and outputs two similar representations: a binary objectness map (which indicates if the voxel belongs to a vehicle or to the foreground) and a bounding box map (which is in charge of predicting its coordinates). To deal with voxels, this is the first approach that applies 3D convolutions, which leads to an increase in accuracy but at the cost of a significant inference time and computational effort. The results of the paper were promising but the subsequent papers will try to mitigate the effect of convolving a 3D tensor.

As a successor of this idea, Zhou and Tuzel proposed **VoxelNet** [50]. The voxel-based representation samples the point cloud taking into account that the variety of density in a LiDAR point cloud leads to an over-representation in nearer voxels. To cope with this issue, they randomly sample the voxels that exceed a maximum number of points. This sampling strategy reduces the computational efforts and augments the randomness during training, which leads to a higher generalization. They propose an architecture based on three networks: a voxel feature extractor, a set of stacked 3D convolutional layers and a 2D RPN as the last component. The inclusion of a RPN based on 2D CNNs also alleviates the computational load. The method is trained on KITTI [8].

Later on, **SECOND** [51] adopted the same structure as VoxelNet. The main contribution of this paper was the introduction of *sparse convolutions* to substitute the costly 3D convolution. This made it possible to run voxel-based architectures between 20 and 40 Hz in real-time.

Bringing together the knowledge acquired by the two previous proposals, **PointPillars** [52] is proposed in 2019 offering a novel and lightweight representation, *pseudo-images*. Pseudo-images are a variant of voxel-based sampling in which all voxels have a height equal to the total height range of the point cloud and are called *pillars*. Then, the architecture has three main components, similar to **VoxelNet** and **SECOND**'s: a pillars-based feature encoder, a 2D convolutional backbone and a detection head based on the work of **SSD** [4]. Therefore, the main contribution is a Pillar Feature Extractor (PFE), which is the component that allows a 2D backbone component and the reach of real-time performance (16.2 ms runtime in an Intel i7 CPU and NVIDIA GTX1080Ti GPU) in consequence.

#### 3.3.1.3. Point-based methods

The review of research lines for object detection in point clouds will end up with point-based methods. **PointNet** [53] and **PointNet++** [54] are proposed by Qi *et al.* and establish a standard for point-to-point feature extraction in point clouds. These two methods perform point cloud classification and segmentation. Therefore, Shi *et al.* transfer the acquired knowledge to the task of object detection in point clouds with **PointRCNN** [55]. It is a two-stage detection framework which consists of a 3D proposal generation stage and a bounding box refinement stage. The first stage is comprised of **PointNet++** with multi-scale grouping. After obtaining point-wise features, there are two outputs. one is in charge of doing foreground point segmentation and the second one is in charge of doing a bin-based 3D bounding box generation. The method is trained and evaluated on KITTI [8].

Due to the importance of the method, it is interesting to review how **PointNet** established a method to extract point-wise characteristics. The methods treat point clouds as point sets in an Euclidean space. Sets have these three properties:

1. Sets are unordered. Then, the network must be invariant to permutations within the input tensor.

2. Interaction among points occurs. Then, a point is not isolated and can interact with its neighbours, forming meaningful subsets.

3. Sets should be invariant under transformations. As a representation of a geometric object, the meaning of the object must not change independently of if the points are translated and rotated as a whole.

The feature extraction relies on three modules: a max pooling that aggregates features from all points, a local and global information structure based on MLPs, and two joint alignment networks that align the input point and point features based on subnetworks that predict an affine transformation matrix. Then, **PointNet++** adds a hierarchical mechanism to apply multiple PointNets to specific subsets and capture better the relationship among neighbour points. Most of the point-based methods are related to these two in one way or another.

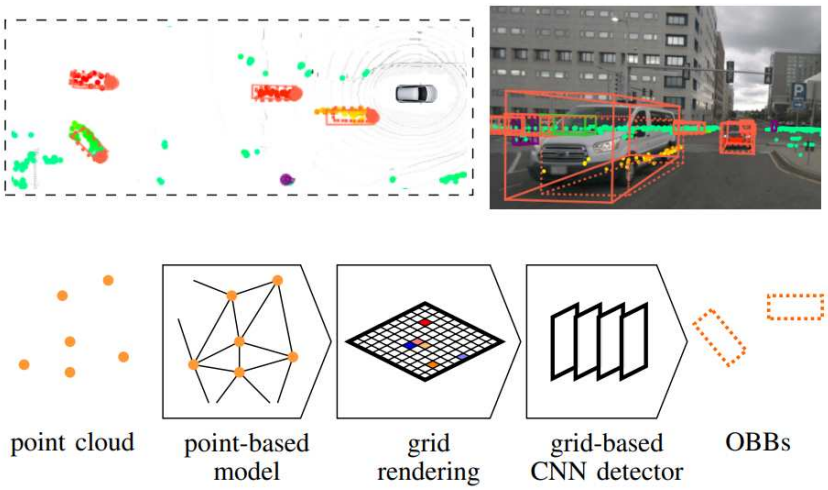



Figure 3.7: Radar-only detection framework based on PointNets for feature extraction, pillar rendering and per-pixel 2D oriented object detector.
Source: Improved Orientation Estimation and Detection with Hybrid Object Detection Networks for Automotive Radar [56]

#### 3.3.1.4. Object Detection in Sparse RADAR Point Clouds

As an additional research line, we will review the progress of RADAR only point cloud-based methods on nuScenes [9] dataset. The vehicle platform with which this dataset was recorded was equipped with 5 Continental ARS408 radars, offering a 360º degree coverage. As this hardware is considered old-gen and provides only 2+1D output point clouds with a very sparse density, literature on this topic is scarce.

However, Ulrich *et al.* [56] proposed in 2022 two novel architectures that combined point- and grid-based approaches to perform object detection and orientation, as shown in Fig. 3.7. Both networks are composed of five main components: a point-based feature extractor, a grid renderer to sample the feature among the corresponding pillars, a 2D convolutional backbone, an FPN as neck and a per-pixel head for oriented bounding boxes. With this schema, the authors are able to merge the advantages of different research trends (studied during this review) into a novel application in RADAR point clouds: feature extraction with **PointNets**, pillar-rendering like **PointPillars**, and a BEV-projected single stage object detector like **FCOS**. Both networks differ in how they extract point-based features: GNNs or Kernel Point-based Convolution (KPConv). Both models are trained on nuScenes, obtaining an mAP of 24.6% and 26.2% on the validation set, respectively. Given these results, the KPConv-based model is submitted to the evaluation server of nuScenes, ranking first on RADAR only category with 29.2% mAP.

The research line is followed up by Köhler *et al.* [57] and Lippke *et al.* [58]. In [57], the authors propose an incremental improvement of [56]: the application of a multi-grid rendering mechanism to take advantage of multiple dense feature maps. In [58], the point-based feature extraction component is composed of two different branches: a KPConv-based branch and a submanifold sparse convolutions branch which aims to obtain computational benefits from the natural sparsity of RADAR point clouds.

| | Model name | Performance on nuScenes | | Performance on KITTI 3D, Medium difficulty | | |
|---|---|---|---|---|---|---|
| | | NDS | mAP | AP Car | AP Ped. | AP Cyc. |
| **Projection** | BirdNet [44] | – | – | 37.1 | 35.3 | 31.9 |
| | BirdNet+ [45] | – | – | 55.6 | 52.4 | 42.6 |
| | FCOS-LiDAR [48] | 0.632 | 57.0 | – | – | – |
| **Voxel** | VoxelNet [50] | – | – | 65.11 | 33.69 | 48.36 |
| | SECOND [51] | – | – | 73.66 | 42.56 | 53.85 |
| | PointPillars [52] | 0.491 | 34.33 | 75.75 | 34.41 | 55.67 |
| **Point** | PointRCNN [55] | – | – | 74.99 | 43.53 | 59.07 |
| **RADAR** | GNNConvPillars [56] | – | *24.6 (Car)* | – | – | – |
| | KPConvPillars [56] | 0.139 | 4.9(All)/29.2(Car) | – | – | – |
| | Köhler et al. [57] | – | *26.42 (Car)* | – | – | – |
| | Lippke et al.[58] | – | *25.98 (Car)* | – | – | – |

Table 3.1: Comparison among State of the Art methods presented. Results are taken from their respective publications. *Italics* means results are taken from nuScenes validation set.

As can be seen in Table 3.1, the methods presented during this Chapter are compared quantitatively. Results are taken from their respective papers within the datasets KITTI [8] and/or nuScenes [9]. We remark the performance of **PointPillars** in both terms of performance on KITTI and their belonging to the voxel-based models family, which will provide it of a faster inference time than **PointRCNN**, a voxel-based method that performs on pair.

### 3.3.2. Metrics

In an analogous way to Section 3.2, a review of the most common metrics will be done. It is worth mentioning that mAP is the most used metric for 3D Object Detection for Point Clouds, as it was in 2D Object Detection for Images. In this case, the criteria to consider a detection as positive is by means of an evaluation of IoU over the three-dimensional space. Now, IoU thresholds depend on the category, i.e. IoU $\geq 0.5$ for Cars, while IoU $\geq 0.25$ for Vulnerable Road Users (VRU) (pedestrians and cyclists).

Additionally, during this Section two more metrics will be presented: Average Orientation Similarity (AOS) and nuScenes Detection Score (NDS).

#### Average Orientation Similarity (AOS)

Object detection in 3D point clouds requires oriented bounding boxes at outputs. Therefore, a metric that jointly assesses the performance of object detection and orientation is necessary. As detection is covered by mAP, AOS introduce the orientation component through a concept known as *cosine similarity*. The orientation similarity is a variant of cosine similarity that is ranged in $[0, 1]$ and is defined as:

$$s(r) = \frac{1}{\mathcal{D}(r)} \cdot \sum_i \frac{1 + \cos \Delta_\theta^{(i)}}{2} \cdot \delta_i \tag{3.7}$$

Where $\mathcal{D}(r)$ is the set of detection for a certain recall value $r$ and $\Delta_\theta^{(i)}$ is the difference on angle $\theta$ estimation and ground-truth for a matched detection $i$. Additionally, $\delta_i$ acts a penalization mechanism: $\delta_i = 1$ if $\text{IoU}_i \geq 0.5$, or $\delta_i = 0$, otherwise.

Then, the calculation of AOS is performed taking orientation similarity for 11 points on the recall curve, in an analogous way to mAP. It is expressed as:

$$AOS = \frac{1}{11} \sum_{k=0}^{k=10} \max s\,(0.1 \cdot r) \tag{3.8}$$

#### NuScenes Detection Score (NDS)

nuScenes [9] has become the dataset of reference in the domain of DL for perception. The spearhead of research from 2021 on has embraced this dataset arguing that KITTI [8] was not challenging enough to discern which of the most novel methods is the better-performing one. The dataset proposed one of the most comprehensive metrics that have been designed for AD yet: **nuScenes Detection Score**. NDS is the weighted sum of six sources of error:

- **Average Precision (AP)**. Explained in Section 3.2.
- **Average Translation Error (ATE)**. Euclidean distance in $x$-$y$ plane between the centroids of the detection and its matched ground-truth.
- **Average Scale Error (ASE)**. Aligning the centres and orientation of the detection and its matched ground-truth, it is computed as $1-\text{IoU}$.
- **Average Orientation Error (AOE)**. Angle difference between the prediction and the ground-truth.
- **Average Velocity Error (AVE)**. Absolute velocity error in m/s.
- **Average Attribute Error (AAE)**. Calculated as $1 - c$, being $c$ the confidence in the class prediction.

All these errors are positive, except for translation and velocity, which can be put in absolute values. Then, these errors are calculated per class and their mean versions are computed. Then, the weighted sum assigns the following ponderations:

$$TP_{error} = 5 \cdot mAP + mATE + mASE + mAOE + mAVE + mAAE \tag{3.9}$$

Finally, NDS is the complementary version of $TP_{error}$, and it is bounded in the range $[0,\ 1]$.

$$NDS = \max\,(1 - TP_{error},\ 0) \tag{3.10}$$

Moreover, in a recent study published in August 2023, Schreier *et al.* [59] made a comparison between perception-focused metrics and holistic-focused metrics in an offline evaluation in

CARLA Simulator [60]. The term *holistic* in Autonomous Driving refers to the navigation as a whole, as a higher-level abstraction. In this study, NDS metric was the most correlated metric with the holistic metric of reference, Driving Score (DS), nearly followed by mAP. This means that both metrics capture the essence of the most relevant aspects of the perception stage to perform autonomous navigation with success.

## 3.4. RADAR-Camera Fusion

As previously introduced in Chapter 1, sensor fusion aims to fuse the data of different modalities or sensors taking advantage of the synergies that can outcome from the combination of them to fulfil a specific task. In the context of Autonomous Driving, sensor fusion improves the robustness and reduces the uncertainty of perception systems. During previous sections of this review, only single-sensor methods have been presented but the most common approach is to fuse LiDAR, RADAR or cameras and any possible combination with this set of sensors have been explored within the recent literature.

Therefore, during this Section, we will define how sensor fusion can be performed by answering four main questions: *what to fuse*, *where to fuse*, *when to fuse* and *how to fuse* [61]. Then, a set of selected methods based on camera and automotive RADAR will be revised.

### 3.4.1. Key Questions on Sensor Fusion

RADAR and camera are two sensors that excel at complementary features. While RADARs provide depth cues, accurate velocity estimation, and are robust against adverse weather conditions, cameras provide a dense, semantic representation. Coincidentally, the strong points of the first are the weak points of the second, and vice versa. Moreover, a perception system based on RADAR and camera is cheaper than a system based on LiDAR, thus mass-scale production is enabled. Therefore, the two main advantages of this fusion are: that the combination of strong points embraces the whole range of capabilities needed for autonomous navigation and that the reduced price of the sensor suite allows mass-scale implementation. Having justified the fusion, let's answer the key questions for fusing correctly.

- **What to Fuse?** RADAR provides a wide range of modalities to fuse that have been reviewed in Section 2.3.1: from raw ADC to the point cloud representation, going through spectrograms and radar cubes. Each one of them offers a different set of advantages and weak points. On the side of the camera, there is less diversity of representations. Among camera modalities, we can find: RGB or gray-scale images for visible light cameras, and infrared or thermal representations.

- **Where to Fuse?** Spatial coherency must exist when fusing data from various sources. Data must be referenced to the same coordinate frame. To tackle this, there are two main options in RADAR-camera fusion: in the perspective projection of the image plane, or in the bird's eye view space. Fusing in the image plane involves the projection of the radar

tensors or point clouds via calibration matrices, forming sparse radar images. Fusing in BEV plane involves the lifting of camera-based features inside a neural network due to the fact that geometric projections under-perform in this task.

- **How to Fuse?** Fusing two sensor modalities involves keeping spatial and temporal coherence or alignment. Temporal misalignment can occur when the latencies are not controlled or compensated. Spatial misalignment occurs when calibration methods are not accurate. Fulfilling these conditions, fusion is done by means of fusion operators. Fusion operators are: transformation matrices for data-level fusions, and additions, multiplication, concatenation and attention mechanisms for feature-level fusions.

- **When to Fuse?** Depending on the stage of the perception pipeline we can classify fusion schemes in these levels: object-level, data-level, feature-level and any combination of them (hybrid-level). Object-level fusion involves two separate independent methods that return their respective detections and a later fusion mechanism. In this case, Bayesian Theory, Kalman Filters and matching algorithms are used to associate detections spatially. The data-level fusion involves the creation of a new representation of the environment before any detection network or method, i.e., projecting the radar features to the image plane and concatenating those channels with the original RGB image, resulting in a different tensor that will feed an NN. The feature-level fusion involves the combination of radar features maps and camera feature maps inside an NN applying any fusion operator (keeping spatial coherence). Then, hybrid approaches can make combinations of any of the previous ones applying different fusion operators.

### 3.4.2. Methods

Having reviewed the main components of sensor fusion, let's review some spearhead methods for RADAR-camera fusion. Although RADAR sensors have been applied to ADAS since many decades ago, there are few studies that tackle the fusion of RADAR and camera information. This research has remained unexplored since the advent of DL, and it has been in the decade of 2020 when a rising interest in the topic has been detected.

As previously introduced, nuScenes [9] has become the dataset of reference for AV applications and has been the first big-scale benchmark to introduce RADAR data. One of the first methods ranked in nuScenes at Camera and Radar Detection Track was **CenterFusion** [62], whose architecture is framed in Figure 3.8. Nabati and Qi propose a feature-level fusion scheme with two main branches. The imaging branch uses **CenterNet** [63] as the backbone. The RADAR branch opts for a pillar rendering mechanism, similar to KPConv. The head is split into two parts: primary and secondary. The primary head obtains preliminary 3D bounding boxes. Pillar features and preliminary boxes are associated through a frustum-based mechanism and a radar feature map is generated. Image and RADAR feature maps are concatenated and are the input to the secondary head, that refines the preliminary proposals. They performed a performance comparison in which the camera-only KPConv scored an NDS = 0.328 and the

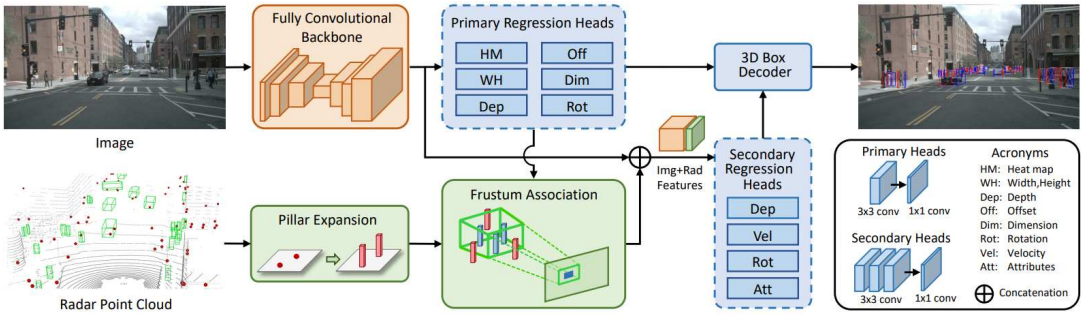



Figure 3.8: CenterFusion network architecture.
Source: CenterFusion: Center-based Radar and Camera Fusion for 3D Object Detection [62]

fusion of both sensors scored NDS = 0.453, an improvement of 0.125 due to the fact of including radar data information.

Nobis *et al.* [64] proposed a method to fuse RADAR and camera information in the image plane, extending the image with radar channels that include range, velocity and RCS information. The method is built based on **RetinaNet** [65] with two branches. The camera image follows a VGG [66] backbone and the radar branch applies max pooling operations to preserve the maximum information possible when the dimensionality is reduced, as it is a sparse image. Then, feature maps from radar and image branches are concatenated at multiple scales. An FPN and a detection head end the architecture. The method introduced a novel augmentation method called *BlackIn*. *BlackIn* shut down the neurons at the input of the camera image, so the network must rely on the RADAR channels to obtain the information.

A remarkable network that opted for a data-level approach is **RadSegNet** [67]. The method uses a 2D detector for oriented bounding boxes. In this case, a BEV grid map is generated from the RADAR point cloud. The grid is augmented with semantic information coming from the camera. A 2D semantic segmentation network is applied to the image. Then, a semantics-to-radar module projects the information from the image plane to the BEV grid map. The final grid map is composed of multiple channels: the grid map, point-based feature maps and semantic maps, leading to a 22-dimensional tensor. The architecture for semantic segmentation is a DeepLabV3+ with a ResNet-101 backbone trained in CityScapes [68], due to the absence of semantic labels in RADAR-camera-based datasets. The architecture for object detection is based on RADAR, treating detection as a per-pixel task, trained on Astyx dataset [69].

# Chapter 4

# Work Development

*We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard.*

John F. Kennedy.

The main purpose of this Chapter is to explain in detail the practical project developed within the scope of this work. A 3D Object Detection framework based on the fusion of monocular camera images and automotive radar point clouds is developed using multiple DL networks for autonomous driving applications.

Therefore, the main objective of this work is to implement a sequential geometrical sensor fusion scheme, **PointPainting** [1], which was proposed for fusing LiDAR and camera, but in RADAR and camera images. The LiDAR version was evaluated on KITTI [8] and nuScenes [9] datasets, in which the method showed a delta improvement of 2.94 and 6.3 mAP, respectively. To the best of our knowledge, this method has not been adapted to RADAR yet and there is no literature related to this type of fusion schemes. We aim to analyze the impact of camera colours and semantics additional information into RADAR point clouds. Additionally, we propose a heuristic set of rules to deal with one of the main disadvantages of the method, the *smearing effect*, which will be further explained.

As for the organization of this Chapter, it will be divided into two sections. Firstly, the dataset used to validate our methods will be explored and analysed, providing insights about the sensors, the recording platform, the format in which the data is presented and the distribution of some important variables. Secondly, the implementation of the aforementioned methods will be explained in detail.

As a remark, during this Chapter, we will go through concepts and methods explained in the previous Chapters. These Chapters served as an introduction and contextualization to the reader to the topics that are been dealt with in this work.

## 4.1. Dataset

Due to the recent advent of High-Resolution 3+1D RADAR, and been RADAR the least explored sensor among DL-based perception techniques, there is a relatively low number of multi-modal datasets that include this sensor[1]. Moreover, it is difficult for any of the available datasets to include annotations for a wide range of tasks. In this context, the **View of Delft (VoD)** [11] was released in Q2 2022, which is a collection of urban sequences recorded in the Dutch city of Delft. An example frame of the dataset and the sensor setup of the recording platform can be seen in Figure 4.1.


(a) Example frame at View of Delft dataset.

(b) Sensor setup at View of Delft dataset.

Figure 4.1: The View of Delft dataset.

The dataset is composed of 8682 individual frames and has official training/validation/testing splits with 5139/1296/2247 frames respectively. The training and validation sets are available to download to academic institutions and the testing set is accessed through a request to the authors. Each frame is composed of a 64-beam LiDAR point cloud, a 3+1D RADAR point cloud, a single image from a stereo vision system, the calibration matrices to transform between reference frames, odometry and GPS information. In addition, every frame is annotated for both tasks: 2D and 3D object detection.

Every LiDAR point cloud is a vector $L_{PC} \epsilon \mathbb{R}^{N \times 4}$, where $N \approx 180.000$ is the number of points. Each LiDAR point is composed by $(x, y, z, i)$, spatial location and intensity of the reflection. Every RADAR point cloud is a vector $R_{PC} \epsilon \mathbb{R}^{N \times 7}$. In this case, the RADAR data is presented in three versions accumulated over time: single-frame and multi-frame (3 frames and 5 frames). Therefore, $N \approx 250$ for the single frame version, and 750, 1250 for the 3-frame and 5-frame versions, respectively. Each radar point is a vector $(x, y, z, v_r, v_{rcomp}, \sigma, t)$, where $(x, y, z)$ is the spatial location of the reflection, $(v_r, v_{rcomp})$ are the radial velocity and its compensated-by-ego-motion version, $\sigma$ is RCS, and $t$ is a time variable which indicates the number of timestamps since that point cloud was captured. The camera information is presented as an RGB image with size $I \epsilon \mathbb{Z}^{3 \times 1216 \times 1936}$.

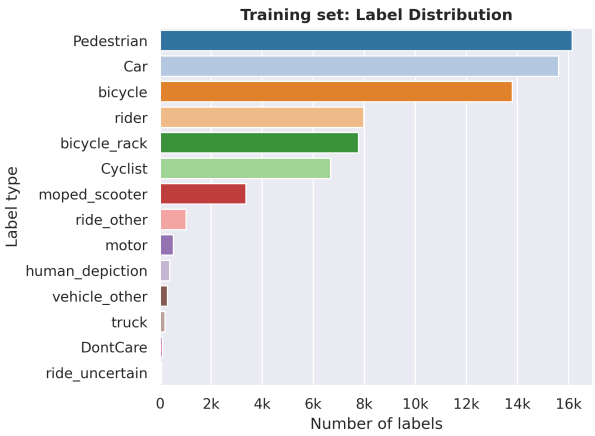
[1] Awesome Radar Perception is a GitHub repository that collects the main developments on RADAR-based DL techniques and datasets.

As previously mentioned, every frame is annotated for 2D and 3D object detection tasks. Frames are annotated in a KITTI-like format. There are 14 label types available: *Pedestrian*, *Car*, *bicycle*, *rider*, *bicycle rack*, *Cyclist*, *uncertain riders*, *moped scooter*, *other riders*, *motorbikes*, *human depictions*, *trucks* and *other vehicles*. The label distributions for training and validation sets are shown in Figure 4.2a and 4.2b, respectively. The labels belonging to the same frame are contained within a text file. In these files, every label is contained in one text line. Each label has 15 values separated by a space.

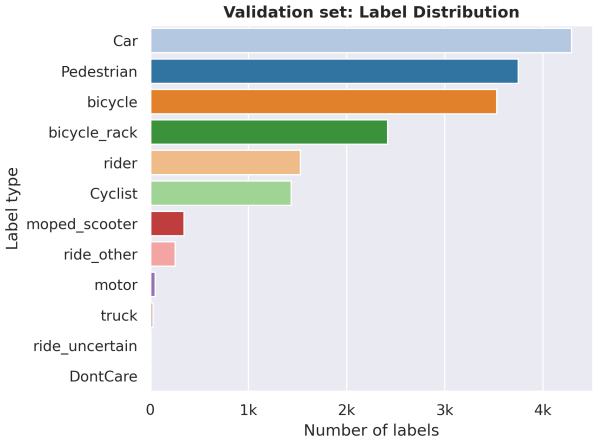
- **Type**: a text string that describes the type of the object.
- **Moving**: an integer that describes if the object is static (0) or dynamic (1). As a remark, this field differs from the original KITTI format.
- **Occluded**: an integer that indicates if an object is fully visible (0), partly occluded (1), largely occluded (2) or unknown (3).
- **Alpha**: a float number ranged in $[-\pi,\ \pi]$ that represent the observation angle of the object.
- **Bounding box 2D**: four integer numbers that represent the left, top, right, and bottom pixels image coordinates of the object.
- **Dimensions**: three float numbers in meters that indicate the dimensions of the 3D bounding box that encloses the object: height, width and length.
- **Location**: three float numbers in meters that indicate the spatial location of the centre of the 3D bounding box in camera coordinates.
- **Rotation**: angle of rotation around the downwards-pointing axis (Y-axis in camera coordinates), ranging in $[-\pi,\ \pi]$.

Additionally, detections can be expressed in the same format but with an additional **score** field, which indicates the confidence of the detection, ranging in $[0,\ 1]$. The annotated zone covers an area of $(x_{min},\ x_{max},\ y_{min},\ y_{max}) = (0,\ 50,\ -25,\ 25)\ m$. Lastly, the sensor setup of the recording platform, a Toyota Prius vehicle, and its main specification must be described.

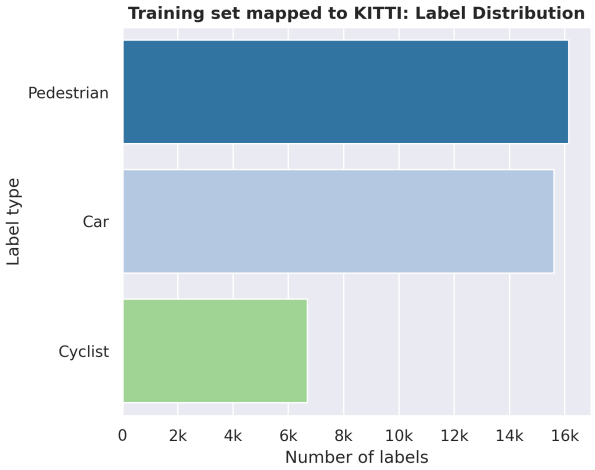
- Camera: The vision system is an IDS stereo camera that provides coloured and rectified images of 1216×1936 pixels at 30 Hz approximately. It has a horizontal FoV of 64º (±32º) and vertical FoV of 44º (±22º). It is located in the driver's position behind the main glass.
- RADAR: The RADAR sensor is a ZR FRGen21 3+1D, which output point clouds at approximately 13 Hz. It provides measurements up to 100 meters in this dataset, as can be seen in Figure 4.3, covering the frontal part as its field of view. It is mounted on the front bumper of the vehicle.
- LiDAR: The LiDAR system is a Velodyne HDL-64 S3, that output measurements every 10 Hz. It is placed at the top roof of the vehicle and it covers a horizontal FoV of 360º and a vertical FoV of approximately 26º.



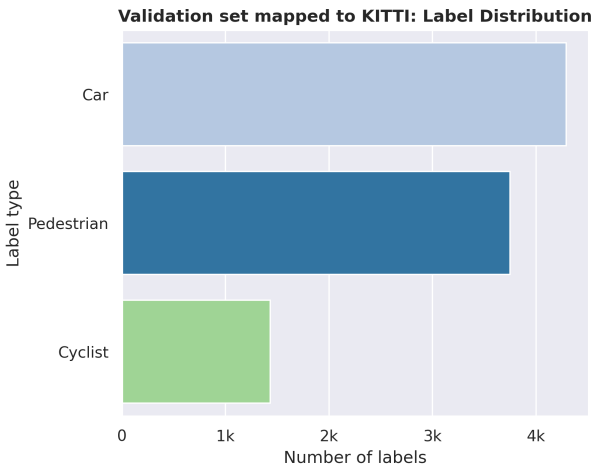

(a) Label distribution in training set




(b) Label distribution in validation set




(c) Label distribution in training set mapped to KITTI categories




(d) Label distribution in validation set mapped to KITTI categories

Figure 4.2: Label distribution in View of Delft dataset.

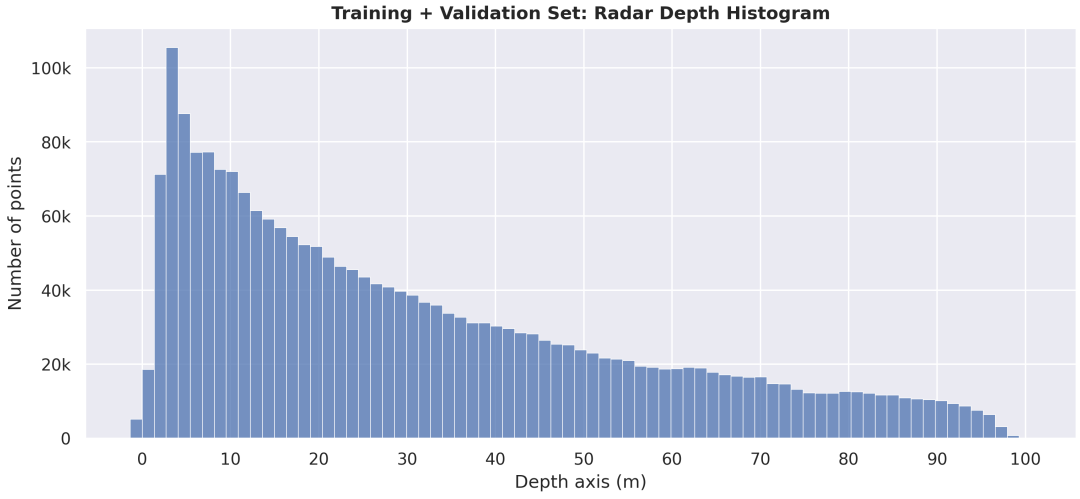



Figure 4.3: Radar depth histogram.

## 4.2. Methods

The proposed method aims to solve the task of 3D Object Detection on RADAR point clouds with additional semantic information coming from a camera image. To solve the task, a framework that combines multiple neural networks and geometrical algorithms is proposed. The four key components that comprise the framework are:

- An instance segmentation network for images using **YOLOv8-seg**.
- A sequential geometrical fusion for point cloud colouring algorithm, known as **Point-Painting**.
- A rule-based cluster refinement algorithm.
- A 3D object detection network for point clouds using **PointPillars**.

This section will perform a theoretical-practical explanation of the four main components. Delving into the inner details of the structure of the neural networks or algorithms and explaining the steps that are needed for their start-up. An overview of the full framework can be seen in Figure 4.4.

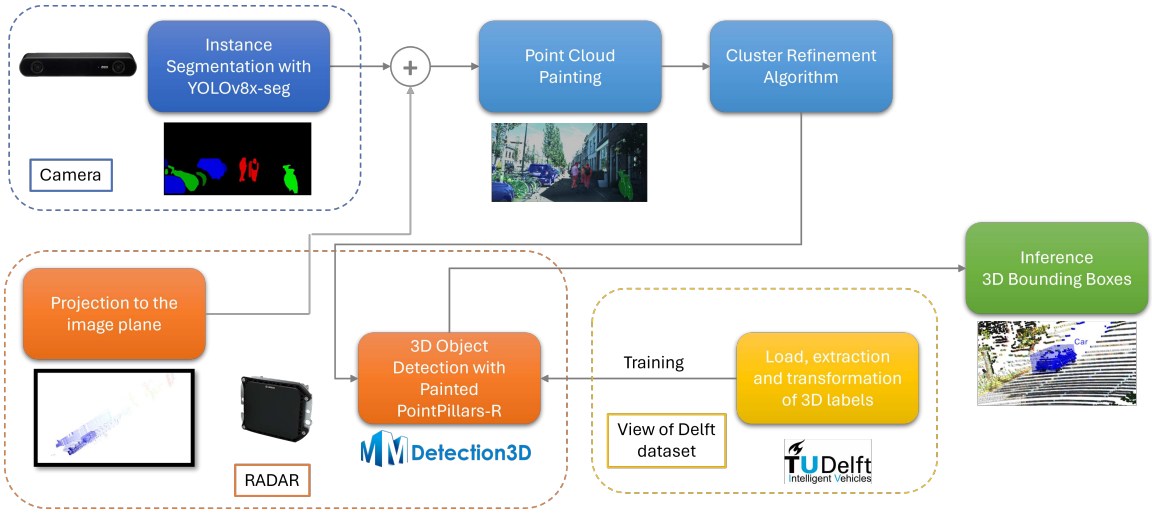



Figure 4.4: Full framework overview.

The development environment is encapsulated in a Docker[2] container. The utilization of Docker containers to develop software applications allows the packaging of all the necessary dependencies needed to develop, deploy and execute the application independently of the working machine and hardware drivers. This approach is particularly useful for research applications where reproducibility and flexibility to work with novel components and tools are important factors. Due to this fact, **NVIDIA** has developed its own registry, the `NGC Catalog`, where tool-specific *images* are stored and allows an easy startup of a project. For the realization of this project, the base image is `nvcr.io/nvidia/pytorch:23.05-py3`[3].

[2] Docker official website.
[3] PyTorch-optimized Docker image at NVIDIA Container Registry.

### 4.2.1. YOLOv8 for Instance Segmentation

The first step of the framework consists of the application of a DL architecture to the camera image in order to obtain the semantics of the scene. For this task, we employ the novel **YOLOv8-seg** architecture implemented in `ultralytics`[4] framework.

#### 4.2.1.1. Architecture

During Chapter 3, a review of the YOLO family algorithms was done, from the original YOLO [35] to the last YOLOv8. In this Section, we will focus on the inner details of YOLOv8 and how to evolve from the object detection version to the instance segmentation one. As a starting point and due to the lack of official diagrams of the architecture, we will refer to Figures 4.5 and 4.6 through this explanation. The first one is extracted from `mmyolo`, an *OpenMMLab* package to implement the YOLO family architectures, and the second one is extracted from **FastSAM** [70], an efficient method to perform prompt universal segmentation based on the **YOLOv8-seg** architecture.

As explained before, YOLO family architectures are categorised as one-stage detectors. Among their main advantages, they offer adequate performance while preserving inference times suitable for real-time applications. YOLOv8 architecture is divided into three main components: backbone, neck and head. The head is divided into two sub-branches, one for object detection and the second one for segmentation. A post-processing stage ensemble both head outputs to provide an instance segmentation representation of the environment.

- **CSP-Darknet backbone**. The backbone of YOLOv8 is based on Darknet and adds Cross Spatial Partial connections (CSP) to reorganize the flow of information through the network. The stem is a convolutional block composed of a convolutional layer, batch normalization and SiLU activation functions. Following there are four blocks composed of a convolutional block and a CSP block with two convolutional blocks inside. The last of these four also includes a Spatial Pyramid Pooling (SPP) block. Spatial Pyramid Pooling (SPP) block allows the network to perform inference with different image sizes. The information flow can be seen in Figure 4.5.

- **PAFPN-based neck**. The neck is composed of two branches: a top-down path and a bottom-up path. The top-down path takes as input the deepest feature map and performs upsampling operations whilst receiving the intermediate feature maps. The bottom-up path downsamples the features taking as input the output of the top-down path and concatenating with other feature maps. The key concept is that the top-down part is fed back by the backbone and the bottom-up is fed back by both, the backbone and the top-down path.

- **Decoupled head for detection**. The concept of decoupled heads is still present in YOLOv8. One is taken from the output of the top-down path in the neck, and the other two are

[4] `ultralytics` is a framework that contains ready-to-use versions of YOLOv3, v5 and v8 for detection, segmentation and pose estimation tasks.

taken from the bottom-up path intermediate feature map and its output, respectively. The resolution for each feature map is 80, 40 and 20, enhancing the detection of objects with a smaller area in the image with this multi-scale approach. Moreover, the detection head is anchor-free, so no prior information is needed and it has more adaptability to predict bounding boxes whose aspect relation has not been seen during training.

- **Proto head for segmentation**. The output of P3, an intermediate feature map that can be seen in Figure 4.6, acts as input of the Proto network. Proto is a fully convolutional network that performs transposed convolutions to upsample the image. The output of this block is a tensor whose size is equal to the image size and with a set of masks, one for each channel, along with a mask coefficient.

- **Post-processing stage**. After getting the candidates to detection from the detection head, NMS is applied to extract the final bounding boxes. Also, in this stage, the mask coefficients, which range from -1 to 1, are multiplied by the masks themselves and are matched up with the boxes through cropping.

Once the architecture has been visited in depth, we approach the implementation of the instance segmentation network.

#### 4.2.1.2. Implementation

As previously mentioned, the YOLOv8 family is implemented within the `ultralytics` package. Therefore, the first step is to install `ultralytics` package from the terminal using the following command inside the Docker container.

```
pip install ultralytics
```

Listing 4.1: Installation of `ultralytics`.

A correct approach would involve the adaptation of the network to this dataset via a full training or *fine-tuning*: a training phase with a reduced learning rate and iteration to close the domain gap between the original dataset and a new one. However, it is not possible to perform a training or *fine-tuning* of YOLOv8-seg in VoD dataset due to the absence of image semantic labels. This fact also led us to choose YOLOv8-seg due to their extensive pre-training on the COCO dataset, which will ensure adequate performance, even with this domain shift.

The next stage of the implementation is to extract an image, whose size is equal to the image coming from the camera, in whose channels are going to be encoded the probabilities of a pixel belonging to any of the categories present in the KITTI evaluation protocol: *pedestrians*, *cyclist* and *cars*. To deal with this, we have to perform inference over an image with pre-trained weights, `yolov8x-seg.pt`: the biggest version of YOLO. Then, the output categories of YOLO are restricted to the ones of interest (mapping from COCO to KITTI): *person* $\rightarrow$ *pedestrian*, *vehicle* $\rightarrow$ *car* and *bycicle* $\rightarrow$ *cyclist*. The output of the inference stage will contain the results for both detection and segmentation.

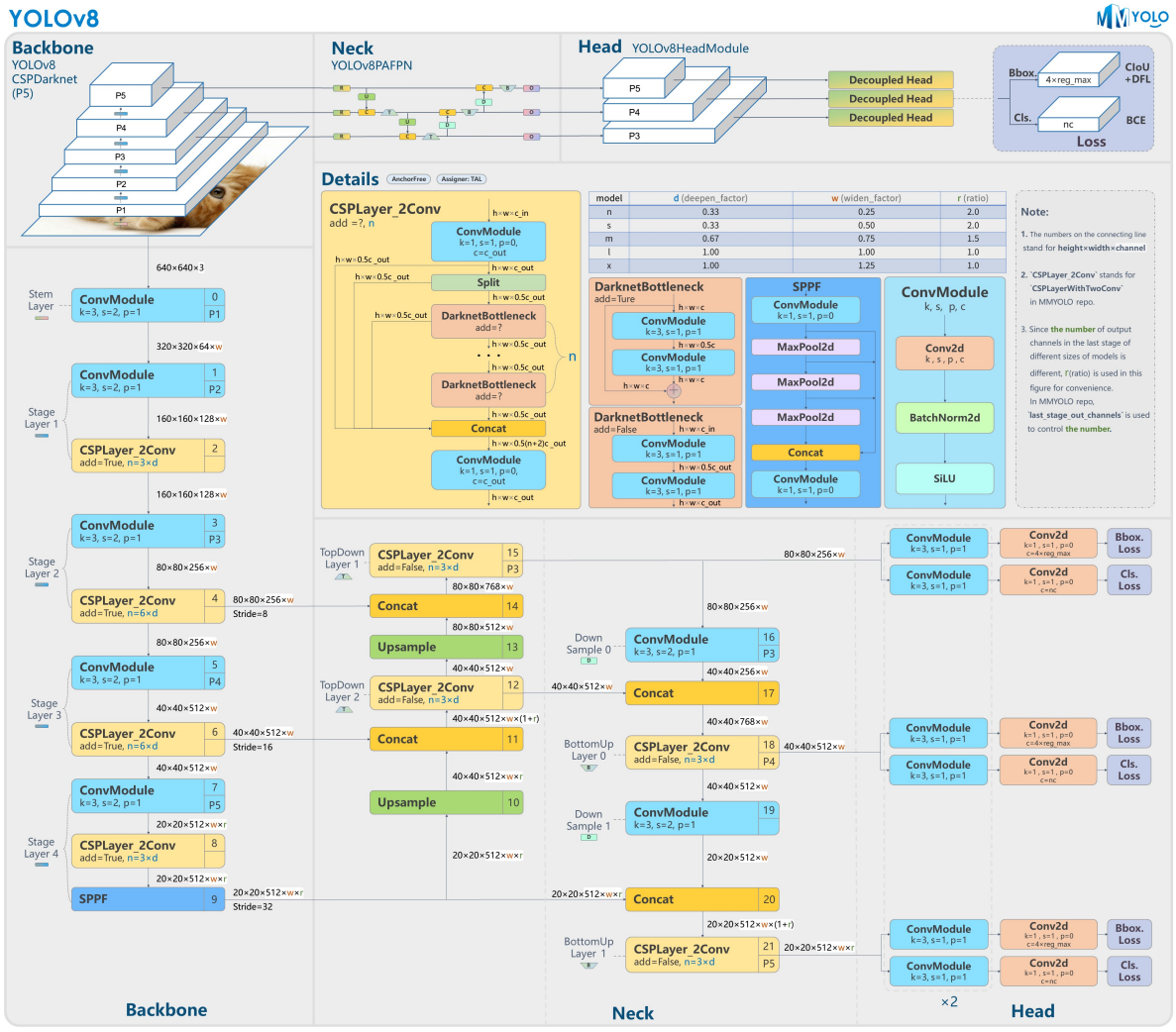



Figure 4.5: YOLOv8 for Object Detection network overview.
Source: `mmyolo`: OpenMMLab YOLO series GitHub repository.

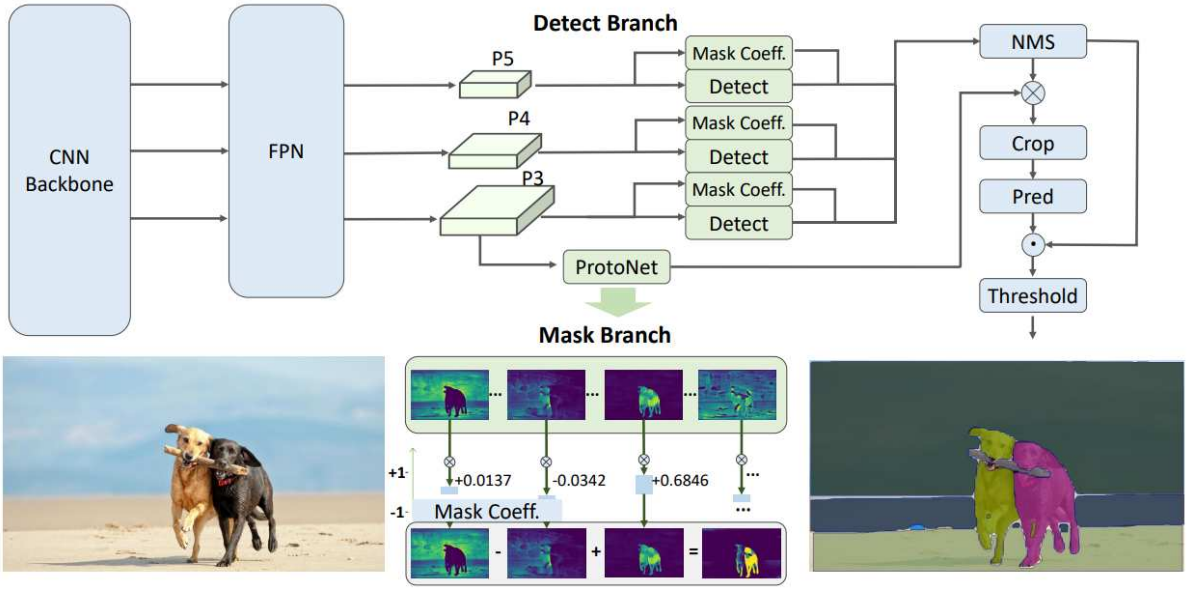



Figure 4.6: YOLOv8-seg for Instance Segmentation network overview.
Source: Fast Segment Anything. [70]

The output masks from the inference stage are binary maps in which positive values represent objectness and negative values represent the absence of an object. Then, the masks are zipped with their corresponding category and confidence of prediction. Each pixel in the mask will add the value of their confidence to their corresponding channel, resulting in an RGB output image. An example of this process can be seen in Figure 4.7.

(a) Camera image for frame `03456`. (b) Segmentation mask for frame `03456`.

Figure 4.7: Image and segmentation mask for frame `03456`.

Once this process is done for all images, we can continue to the fusion step.

### 4.2.2. PointPainting for Geometrical Fusion of Images and Point Clouds

Once the semantic information from the camera image is available, it is necessary to propagate the information from the image plane. For this, we resume the original idea from **PointPainting** [1] paper. During this Section, on the one hand, we will explain the coordinate transform system employed originally by the KITTI dataset and implemented by the VoD dataset, whilst, on the other hand, we will delve into the implementation of the PointPainting algorithm.

#### 4.2.2.1. Sensor Coordinate Transforms

The VoD dataset provides a set of matrices to transform from one sensor coordinate reference frame to another. For this work, we will focus on five reference frames: 2D image plane, 3D rectified camera[5], 3D camera, LiDAR and RADAR. The first refers to the image. The second and third refer to the physical location of the camera, after eliminating distortion components and before, respectively. The fourth and fifth ones refer to the LiDAR and RADAR physical locations, respectively. Four matrices allow us to interchange between these reference frames. By treating reference frames as nodes and transformation matrices as arcs, it is possible to build a directed graph that represents all the possible transformations. Although we have defined this graph as directed, it could be possible to reverse the direction by taking the inverse matrices for each arc. The computation of the inverse is direct for all matrices except for the matrix $P_2$, with

[5]In the VoD dataset, the rectification matrix is an identity matrix, which means that there is no rectification to correct. We include this matrix for the completion of the explanation.

size $3 \times 4$. In this case, the Moore-Penrose pseudo-inverse is applied. For a deeper theoretical explanation, we refer the reader to Chapter 2. The matrices involved are:

- **P2 matrix**.

$$P_2 = \begin{pmatrix} 1495.47 & 0.0 & 961.27 & 0.0 \\ 0.0 & 1495.47 & 624.90 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix} \tag{4.1}$$

- **Homogeneous transformation from LiDAR to camera**.

$$Tr_{velo\,2\,cam} = \begin{pmatrix} -0.008 & -1.000 & 0.015 & 0.151 \\ 0.118 & -0.015 & -0.993 & -0.461 \\ 0.993 & -0.006 & 0.118 & -0.915 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

- **Homogeneous transformation from RADAR to camera**.

$$Tr_{radar\,2\,cam} = \begin{pmatrix} -0.014 & -1.000 & 0.018 & 0.053 \\ 0.109 & -0.019 & -0.993 & 0.981 \\ 0.993 & -0.012 & 0.110 & 1.444 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

With all reference frames and transformation matrices presented and explained, we build a directed graph that represents the relationships among them. This graph, which is also known as **transform tree**, can be seen in Figure 4.8.

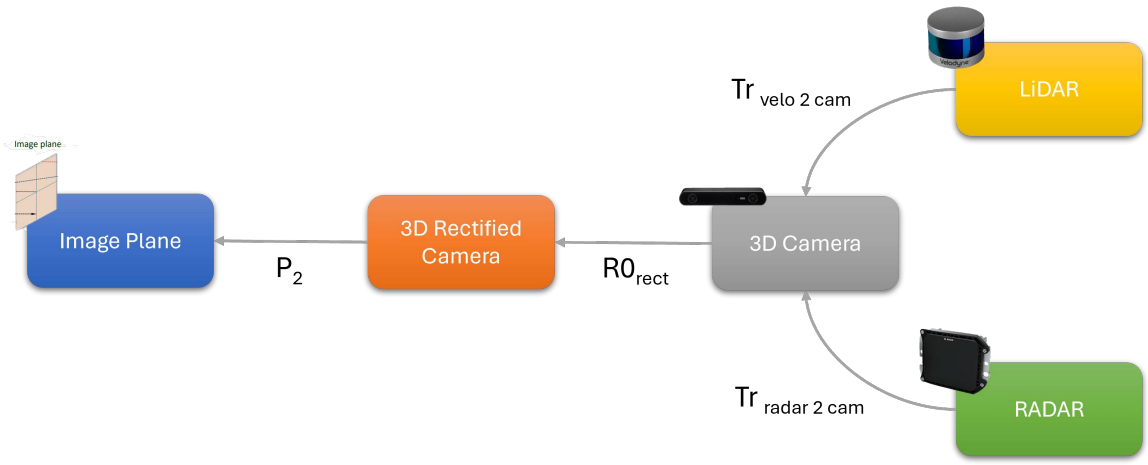



Figure 4.8: Transform Tree: How to navigate between coordinate reference frames.

We can exemplify a path through this graph with the case of transforming a point referenced to the RADAR reference frame to the image plane:

$$P_{UVW} = P2 \,@\, R0_{rect} \,@\, Tr_{radar\,2\,cam} \,@\, P_{XYZ1} \tag{4.4}$$

Where $P_{UVW}$ represents a point in the image plane in non-normalized homogeneous coordinates, $P_{XYZ1}$ is a homogeneous point in the RADAR reference frame and @ denotes the matrix multiplication operator.

#### 4.2.2.2. PointPainting Algorithm

The **PointPainting** algorithm propagates the colour and semantic information obtained from the image to the RADAR point cloud to create a richer representation that **PointPillars** will consume. Once all the transformation matrices have been explained, Equation 4.4 must be applied to all the points in the point cloud to establish a point-pixel correspondence. With this correspondence, the information is concatenated to the points as additional channels. The output of this algorithm is a higher-dimensional point cloud. The whole algorithm is also explained in Algorithm 4.1. A qualitative result of applying this algorithm can be seen in Figure 4.9a.

**Data:** Radar point cloud $R_{PC} \epsilon \mathbb{R}^{N,D}$ with $N$ points, camera image $I \epsilon \mathbb{Z}^{3 \times 1216 \times 1936}$, segmentation scores $I \epsilon \mathbb{R}^{C \times 1216 \times 1936}$ with $C$ classes, homogeneous transformation matrix $T \epsilon \mathbb{R}^{4\times4}$, camera matrix $M \epsilon \mathbb{R}^{3\times4}$.
**Result:** Painted radar point cloud $R_{PC} \epsilon \mathbb{R}^{N,D+C+3}$.
**for** $p \epsilon R_{PC}$ **do**
  $p_{image} = Project\,(M, T, p)$;
  $s = S\left[p_{image}[0], p_{image}[1], :\right]$;
  $c = I\left[p_{image}[0], p_{image}[1], :\right]$;
  $p = VStack\,(p, s, c)$;

**Algorithm 4.1:** Point Painting Algorithm

### 4.2.3. Rule-based Cluster Refinement Algorithm

As claimed in the original PointPainting paper [1], one of the main disadvantages of the method is the *smearing* effect. This effect occurs mainly due to four phenomena: the imperfection in the segmentation quality, the temporal misalignment between the point cloud and the image, the absence of motion compensation methods in these datasets, and the scattering effect in RADAR point clouds.

At a glance, the *smearing* effect produces a trace of coloured points in the angle bin in which the object of interest is projected. For example, when a pedestrian is segmented, some of the points that belong to the pedestrian in the 2D image plane may not exactly fall within the points that belong to the real pedestrian contour in the 3D world. These points may belong to an object or *stuff* that is positioned meters behind the mask (Figures 4.9b and 4.9c).

It is difficult to tune the algorithm in order to obtain an exact segmentation. However, we design a set of heuristic rules to detect the cases with the evident presence of *smearing*

effect through the concept of **radial dispersion**. Given the segmented points, the instance to which they belong and their categories, we analyze the dispersion in range $\Delta r$ of the segmented points. Also, we set prior radial thresholds for each category, which corresponds to double the maximum dimension in the anchors generated by **PointPillars**, i.e. the maximum dimension for *Car* category is their length, 3.9 m, so we double it to obtain a threshold of 7.8 m. For *Pedestrian*s and *Cyclist*s, 1.6 m and 3.52 m are chosen. Further studies could lead to determining the optimal parameters for this threshold.

Once a smeared object is detected, we analyze the distribution of the points within that cluster. Firstly, we determine the presence of moving points thanks to the compensated radial velocity component. Secondly, we filter the points with RCS $\leq 40\, dBsm$, since these points are prone to be noisy reflections. Then, we apply the dynamic object criteria or the static one, depending on the result of the first condition.

- **Dynamic objects**. If an object is dynamic, we cluster the points by means of a DBSCAN clustering algorithm, and we choose the cluster with more points and mean velocity different from zero.
- **Static objects**. If an object is static, we cluster the points by means of DBSCAN in range and we retain as surviving points the closest cluster in range.

In order to clarify the explanation of the algorithm and to remark on its qualitative performance, Figure 4.9 shows an example of a cluster before and after the refinement.

### 4.2.4. PointPillars for 3D Object Detection

The final step of the framework consists of applying a 3D object detection neural network over the painted RADAR point cloud. The modification of the point cloud implies that the detector must have an encoder that is able to admit several features and transform them into a different representation that is feature-agnostic and does not rely on the set properties. For this task, we employ **PointPillars** implementation that is present on `mmdetection3d`[6] framework.

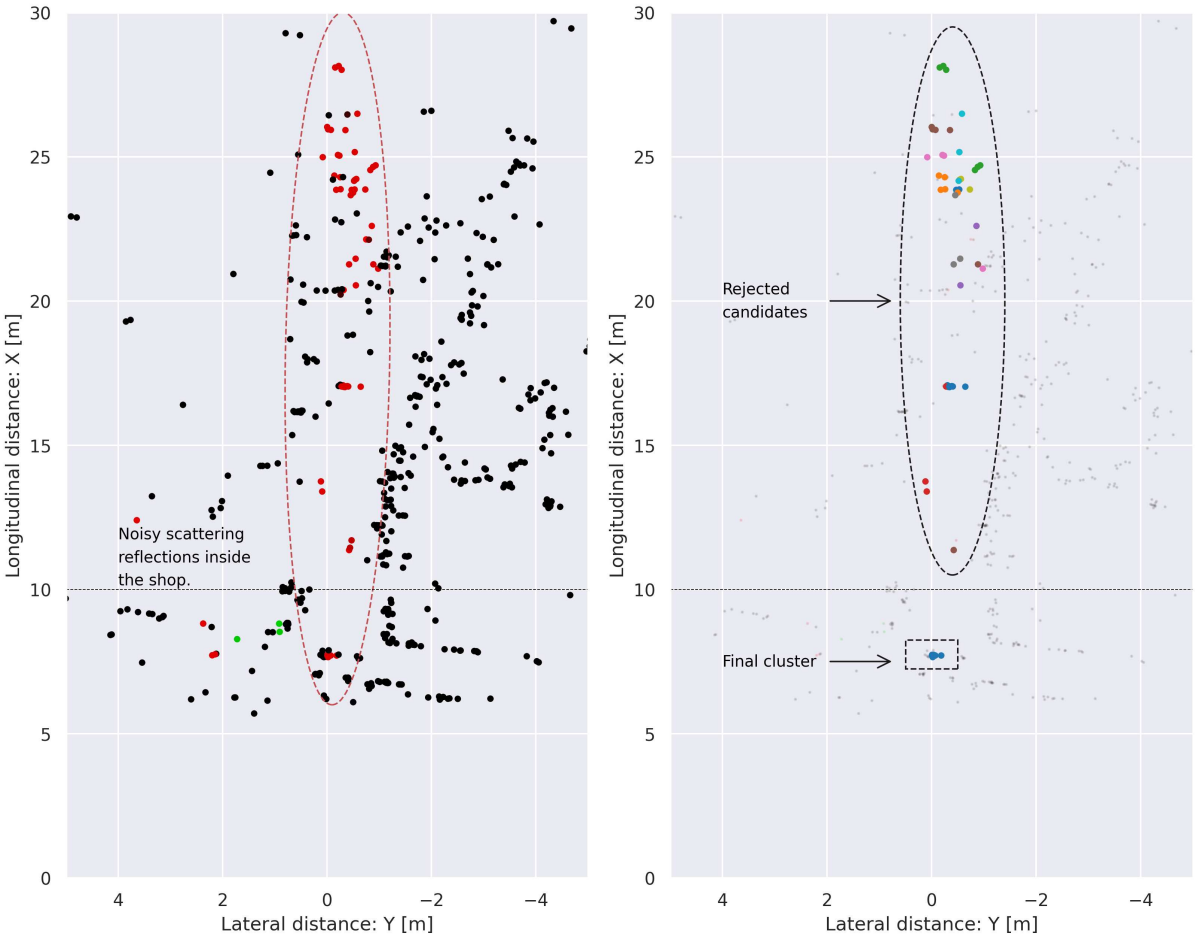
#### 4.2.4.1. Architecture

In Chapter 3, a theoretical explanation of how the 3D Object Detectors evolved until **PointPillars** appeared was made. In this Section, we will delve into the main component of the architecture and their main purpose and contribution to the main architecture following the structure presented in the `mmdetection3d` package and attending to their nomenclature. This will allow us to deeply understand the architecture that is been dealt with in the practice. The explanation will be an interpretation of the base configuration file `pointpillars_hv_secfpn_kitti.py`[7].

[6] `mmdetection3d` is a research-oriented framework that contains several implementations of SOTA 3D object detectors.

[7] Base configuration file of PointPillars implementation for KITTI dataset on `mmdetection3d`.


(a) Camera image for frame `00125`.




(b) Clusters before refinement algorithm.

(c) Cluster of interest after refinement algorithm.

Figure 4.9: Example of cluster refinement algorithm in frame `00125`. A large set of reflections coming from scattering reflections inside the shop are being projected into the image standing at the door, producing a large *smearing*. The cluster refinement algorithm is able to detect the radial dispersion and correct the probabilities assigned to the points in the point cloud.

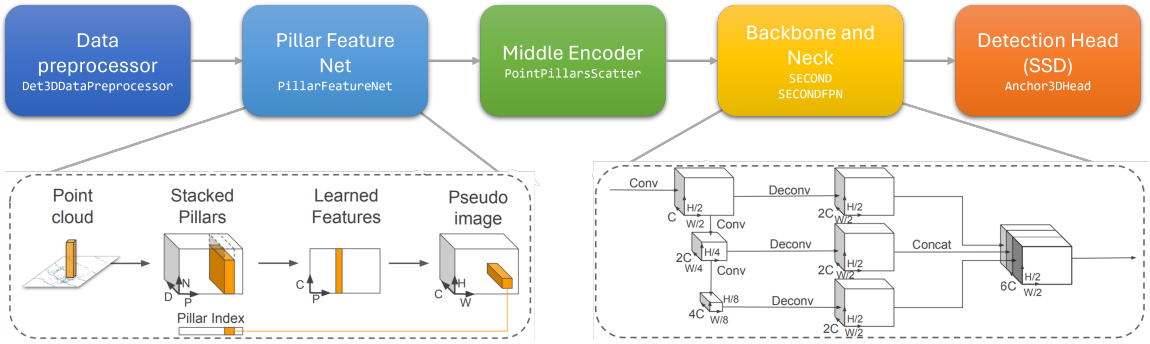



Figure 4.10: PointPillars network overview.
Source: Adapted from PointPillars: Fast Encoders for Object Detection from Point Clouds [52]

The framework divides the network into six main components: the data preprocessor, a voxel/pillar encoder, a middle encoder, a backbone, a neck and a head. This configuration nomenclature follows the naming conventions presented in Chapter 3. Moreover, the architecture overview can be seen in Figure 4.10.

- **Data preprocessor**. The module in charge of preprocessing the data is `Det3DDataPreprocessor`. It receives a boolean flag set to `True` which indicates that the point cloud must be voxelized. The voxelization takes as input the maximum number of points per voxel, the point cloud range, the voxel size and the maximum number of voxels.

  The voxelization process follows a random sampling strategy in which if the number of points that fall within a voxel overpasses the maximum threshold, the candidates are chosen randomly. This strategy acts as a form of data augmentation in the training process, since the network is forced to predict the same instance when the input voxels do not possess the same features. In this case, there is no need to control the height because the pillars voxelize the space in an x-y grid.

- **Pillar Feature Net**. The `PillarFeatureNet` is in charge of extracting features per pillar. In takes as input arguments the number of dimensions of the point clouds and the number of feature channels that will output. The points in each pillar are augmented with the arithmetic mean of all the points in the pillar and the offset of each point to the centre. In LiDAR point clouds, it is claimed that approximately between $6k$ and $9k$ pillars will be empty, so it is important to exploit the sparsity of the clouds by imposing a maximum number of empty pillars. Then, a simplified version of **PointNet** (the original has only one layer, but `mmdetection3d` uses multiple of these) is used to extract a longer feature vector from the original dimensions and augmented geometrical properties.

- **Middle encoder**. The middle encoder converts the learned features from a dense tensor after being processed by `PillarFeatureNets` to a sparse pseudo-image of size $(C, H, W)$, where $C$ is the number of learned features and $(H, W)$ are the height and width of the x-y grid in pixel units, by means of the `PointPillarsScatter`.

- **Backbone**. The backbone is a convolutional architecture that is shared with other architectures such as **SECOND** [51] or **PartA2** [71]. It is a top-down network composed of a series of convolutional blocks (convolutional layers, batch normalization and ReLU activations) that produce features with smaller resolutions for every layer. This block in `mmdetection3d` is called `SECOND`.

- **Neck**. The neck is also shared with the other architectures. It is a series of convolutional blocks whose purpose is to perform upsampling of the deepest features and concatenating them to the earlier feature maps extracted by the backbone. These convolutional blocks use transposed 2D convolutions in order to increase the size of the feature map. The rest of the block is similar: batch normalization and ReLU activations. This block in `mmdetection3d` is called `SECONDFPN`.

- **Head**. The head in the original paper is taken from **SSD** [4] architecture. This component in `mmdetection3d` is named as `Anchor3DHead`. The head is fed with a set of anchors that incorporate prior information about the median sizes of the bounding boxes for every category and a grid of positions in which the boxes can be spawned as a *first initial guess*. The anchor sizes are: [0.8, 0.6, 1.73] m for *pedestrians*, [1.76, 0.6, 1.73] m for *cyclists* and [3.9, 1.6, 1.56] m for *cars*. The anchors are generated with rotations 0 or $\pi/2$ rad. This head treats object detection in the BEV plane as a matching problem using IoU. Once the ground truth and the candidates are matched, height and elevation are additional regression problems, leading to a complete 3D object detection network.

Lastly, the model is trained using a combination of three loss functions: focal loss [65] to address the classification of the bounding boxes in an imbalanced-class scenario, a smooth L1 loss for the regression or bounding box parameters and a cross-entropy loss for the classification of angles (classification losses are used to resolve angle ambiguities). Once the whole architecture has been explained in detail, both the author and reader are fully contextualized to deep dive into the implementation process of **PointPillars** for RADAR point clouds.

#### 4.2.4.2. Implementation

**An introduction to *OpenMMLab* ecosystem.**

The implementation of **PointPillars** used in this project relies on the `mmdetection3d` package. This package is part of *OpenMMLab*'s research framework. *OpenMMLab* is an initiative that open source project for both academic research and industrial applications. Its core is implemented in Python and PyTorch in the packages `mmengine` and `mmcv`. On top of both, task-specific packages cover a wide range of implementations and modules. According to them, the research PyTorch ecosystem produces 50% of new paper implementations in plain PyTorch and the other 50% relies on their framework. In the case of `mmdetection3d`, it is based on `mmcv` and `mmdetection` and incorporates implementations of SOTA methods for 3D Object Detection for point clouds and RGB-D images (uni-modal and multi-modal), and 3D Semantic Segmentation.

All the toolboxes within the *OpenMMLab* environment work similarly. Each toolbox is composed of a model zoo with pre-trained weights of a selection of SOTA method of the specific task toolbox, a set of config files written in Python that allows the customization of the internal modules, the supported dataset files and the tools to train, test and perform inference with all the supported models.[8] Within the configuration files, it is possible to customize the following internal modules.

- The model structure.
- The dataset.
- Data processing pipelines.
- Optimizer and learning rate schemes.
- Runtime configurations.

Then, the internal modules focus on parsing the configuration files and building all the components of the pipeline to train, test and perform inferences with the model. This flexible scheme allows the rapid development of new differential improvements of existing architectures or even completely novel methods and it is the main argument in favour of the adoption by the research community. Currently, works are focused on extending the tasks covered by the framework and creating novel lightweight inference pipelines since depending on big frameworks to deploy a model is not a good software decision.

### Adapting PointPillars for RADAR Point Clouds

The inner workings of *OpenMMLab*'s toolboxes can be explained by following the process needed to adapt PointPillars to admit RADAR Point Clouds as input. Firstly, we will install the package and its required dependencies in the Docker container running the following commands in sequential order.

```
pip install -U openmim
mim install mmengine
mim install 'mmcv>=2.0.0rc4'
mim install 'mmdet>=3.0.0'

git clone https://github.com/open-mmlab/mmdetection3d.git

cd mmdetection3d/
pip install -e .
```

Listing 4.2: Installation of `mmdetection3d`.

This code snippet will install all the required dependencies and trigger the `setup.py` script inside `mmdetection3d`.

[8] A comprehensive tutorial about the inner workings of the *OpenMMLab*'s ecosystem.

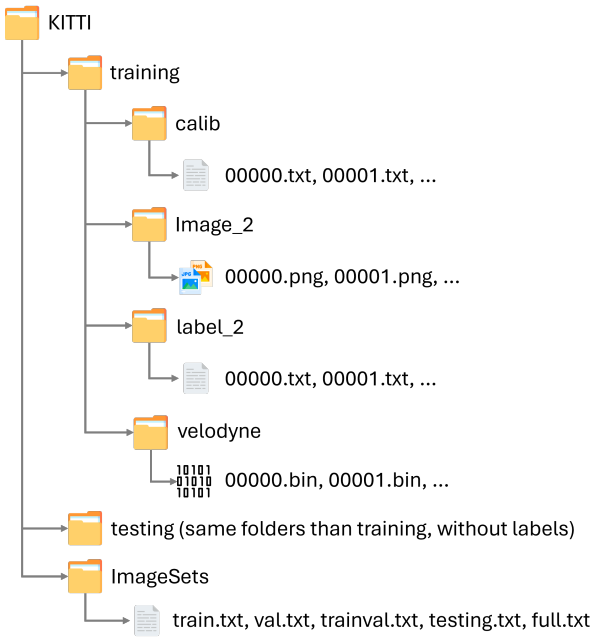



Figure 4.11: KITTI-like dataset folder structure.

Once the package is installed, we need to convert the format of VoD dataset to a pure KITTI format in order to use `mmdetection3d` KITTI internal modules. To do this, we will code a class `Delft2Kitti` that will receive as input the input path of the original VoD dataset and the desired output path. The desired output folder must follow the structure shown in Fig. 4.11. The main changes to the dataset are:

- Colourize the RADAR point clouds using the algorithms previously explained and save all features in `.bin` files.
- Relocate the RADAR point clouds in the folder `velodyne`.
- Change the codification of the `Truncation` field in the labels. The easiest way is to set a zero value for all samples.[9]
- Filter the labels by category. In the KITTI evaluation protocol, only the classes *pedestrian*, *car* and *cyclist* are evaluated. The distribution of the dataset after this filter operation can be seen in Figures 4.2c and 4.2d for training and validation sets, respectively.

The rest of the elements that do not require additional modifications are copied and pasted into the output folder structure. Next, we must create a serialized version of the dataset for fast loading during the training stage. Due to the large amount of data that is handled within these datasets, the KITTI output structure must be encapsulated within `Pickle` files. `Pickle (.pkl)` allows the serialization to disk of Python structures, such as lists, tuples or dictionaries. In this case, *OpenMMLab* ecosystem makes intensive use of dictionaries to

[9]Multiple users claimed in a GitHub issue that pedestrians and cyclist performed poorly. This is due to the codification of moving in the field of truncation. The data processing pipelines of `OpenPCDet` and `mmdetection3d` removed most of the samples of these categories.

propagate the information between their internal modules. To do this, the following command must be executed (at the root directory of `mmdetection3d` folder) and we must indicate the number of dimensions loaded in the respective scripts.

```
python tools/create_data.py kitti
```

Listing 4.3: Convert data from KITTI to Pickle in `mmdetection3d`.

Once the data is prepared for training, it is necessary to adapt the model for training. This will involve the modification of the configuration files related to the dataset, the model and the training stage. To reuse the configurations given by the framework, the modifications will start from the file `pointpillars_hv_secfpn_8xb6-160e_kitti-3d-3class.py` (located in the *config* folder of the repository), which offers a configuration for training in KITTI-like datasets with 3 classes: pedestrian, cyclist and cars. A copy of this file will be renamed `pointpillars_radar.py` and the modifications will be done in this new file. Overall, the main changes that the original PointPillars architecture suffers are:

- **Point cloud range**. As VoD dataset only provides labels annotations inside the area $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}) = (0, 50, -25, 25, -3, 2)\ m$, the point range must be clipped inside the zone $(0, 51.2, -25.6, 25.6, -3, 2)\ m$. The original KITTI range filter covers a wider range $(0, 69.12, -39.68, 39.68, -3, 1)\ m$.

- **Voxel size**. As the point cloud range is taller, the voxel size must increase its height from 4 m to 5 m.

- **Maximum number of points per voxel**. Previously, it was explained that the Pillar Feature Encoder samples the point cloud following a random sampling strategy that adds variability to the training process. This strategy triggers when a number of points coexist in the same voxel/pillar. Due to the sparser nature of RADAR point clouds, this threshold is decreased from 32 points to 10, so the sampling strategy has bigger chances to trigger when operating with RADAR.

- **Elimination of data augmentations**. The original PointPillars proposed to perform data augmentation with three operations: point cloud flipping around the X-axis, point cloud rotation in the range $[-45, 45]$ and point cloud scaling ($\pm 5\%$). The inclusion of features related to radial velocity invalidates the use of these augmentations since they are defined as a vector with a specific modulus and direction. So, the application of augmentations will corrupt their physical meaning.

- **Components related to the loading of point clouds**. Most of the components that deal with loading operations have their dimensions fixed to 4 (if working with KITTI) or 5 (nuScenes). We must change the number of dimensions to 7 for original RADAR point clouds and 13 for coloured point clouds.

- **Pillar/Voxel Feature Encoder component** (PFE). PFE is dependent on the number of input dimensions of the RADAR point cloud, so it must also follow the previous change.

- **Adaptation of the anchor generation ranges**. As **PointPillars** is an architecture based on anchors, they must be generated within the region of the new point cloud.

Additionally, two modifications related to the dataset and data processing pipelines must be done:

- **Removal of filter by points threshold**. In the original KITTI sampler, boxes with less than 15 points (in the case of cars) or 5 points (for pedestrians and cyclists) are filtered. Due to the sparsity of RADAR point clouds, boxes with at least 1 point inside are kept.
- **Batch size incremented in the train dataloader**. From the original batch size of 6, it is augmented to 16 in order to increase the speed of the training process.

Having adapted the architecture, the following step is to execute the command to run the training. At the `mmdetection3d` folder, the user must enter in the terminal the following command:

```
python tools/train.py configs/pointpillars/pointpillars_radar_{version}.py
```

Listing 4.4: Train PointPillars for RADAR in `mmdetection3d`.

This command will trigger the training stage, and the *version* field corresponds to the version of the architecture that will be trained: original or painted, single scan or multi-scan (3 and 5). Detailed information about the training process and its evolution will be discussed in Chapter 5.

# Chapter 5

# Experiments and Results

*No amount of experimentation can definitively prove me right; but a single experiment can prove me wrong.*

Albert Einstein

With the purpose of obtaining an overview of the performance of the implemented method, an evaluation in quantitative and qualitative terms will be made during this Chapter. From the set of metrics explained during Chapter 3, we will evaluate the model by means of the Mean Average Precision (mAP) with a 40-point interpolation and 3D Intersection over Union (IoU) criterion, split over ranges and difficulties. Following the criterion made by the authors of [11], IoU thresholds will be 0.5 for *Cars*, and 0.25 for *Pedestrian* and *Cyclist* categories. The comparison will be performed between **PointPillars** for RADAR, which we will refer to as **PointPillars-R**, and the colourized version, which we will refer to as **Painted PointPillars-R**. In the second part of this Chapter, qualitative results will be presented for the best method.

## 5.1. Training of PointPillars

To perform a fair comparison among models in the research community, there is a set of standard rules to configure the training stage in KITTI.

**PointPillars** models and their variants are trained for 80 epochs in a KITTI-like dataset as a convention[1]. The learning rate is configured using around an initial value of $\eta = 1 \times 10^{-3}$ and is modified every epoch using a learning rate scheduler: **Cosine Annealing** with two stages. The first stage covers from the beginning of the training to epoch 32 and increases from $10^{-2}$ to $10^{-3}$ in a linear trend. The second stage covers from epoch 32 to the end of the training in epoch 80 and decreases $\eta$ from $10^{-3}$ to $10^{-7}$ following Equation 5.1. The optimizer is AdamW, an adaptive momentum optimizer with weight decay.

$$\eta_t = \eta_{min} + \frac{1}{2}\left(\eta_{max} - \eta_{min}\right)\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right) \tag{5.1}$$

[1]Default configuration for **PointPillars** trained on the KITTI dataset applied to LiDAR data.

## 5.2. Quantitative results on View of Delft dataset

Following the aforementioned training procedure, we will train **PointPillars-R** and **Painted PointPillars-R**. The first one will serve as a baseline to compare with the latter, our approach. Moreover, we will experiment with the temporal aggregation of RADAR point clouds among frames and we will train for each of the versions of the VoD dataset: single frame, 3 frames and 5 frames. As a remark, the metrics will be obtained over the validation set of VoD.

| | BEV | | | 3D | | | AOS | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Easy** | **Med** | **Hard** | **Easy** | **Med** | **Hard** | **Easy** | **Med** | **Hard** |
| **AP40 Car (IoU @ 0.5)** | | | | | | | | | |
| **1 frame** | 29.63 | **40.10** | **33.21** | 24.20 | 30.42 | **27.40** | 20.14 | **28.28** | **23.60** |
| **3 frames** | **32.78** | 36.62 | 30.41 | 25.33 | **30.52** | 24.95 | **21.52** | 25.94 | 21.47 |
| **5 frames** | 31.69 | 32.31 | 26.44 | 25.85 | 28.77 | 21.13 | 21.06 | 21.02 | 17.07 |
| **AP40 Pedestrian (IoU @ 0.25)** | | | | | | | | | |
| **1 frame** | 37.46 | 33.36 | **30.61** | 30.45 | 27.37 | 24.18 | 19.22 | 16.88 | 15.25 |
| **3 frames** | **38.20** | **33.62** | 29.85 | **31.76** | **28.26** | 24.88 | **21.27** | **18.45** | **16.55** |
| **5 frames** | 36.09 | 33.13 | 29.41 | 30.44 | 27.81 | **24.97** | 20.04 | 17.85 | 15.91 |
| **AP40 Cyclist (IoU @ 0.25)** | | | | | | | | | |
| **1 frame** | 67.53 | 63.79 | 57.81 | 63.25 | 58.29 | 53.42 | 48.28 | 44.61 | 39.74 |
| **3 frames** | 71.53 | 66.19 | 59.94 | 66.88 | 61.50 | 55.40 | 51.71 | 46.71 | 42.26 |
| **5 frames** | **76.41** | **70.12** | **63.60** | **73.15** | **66.97** | **60.31** | **58.22** | **53.51** | **47.56** |

Table 5.1: Quantitative results of PointPillars-R with IoU thresholds (0.5, 0.25, 0.25) over the validation set of View of Delft dataset. Best results in **bold**.

Table 5.1 shows the results of training the baseline **PointPillars-R** over the validation set of the VoD dataset. The table is split among three different criteria: metric, difficulty and categories. Among metrics, from left to right: mAP in the BEV plane, mAP in the 3D space, and AOS. Among difficulties: easy, medium and hard, depending on the size of the bounding box in the image plane and its occlusion level[2]. Among categories, the three evaluated ones: *Car*, *Pedestrian* and *Cyclist*.

The performance results of the network indicate that RADAR-based methods have difficulties extracting bounding boxes for *Car* and *Pedestrian* categories. This supports the theoretical study made in Chapter 2, in which it was mentioned the poor spatial resolution of RADAR point clouds. Poor spatial resolution leads to scarce reflections in the targets of interest and difficulties in estimating the shape of the objects. Surprisingly, the network shows an excellent performance in the *Cyclist* category, ranging from 63.25 3D AP40 in easy difficulty and single-frame to 73.15 in easy difficulty and 5 accumulated frames. This is an atypical situation in automotive datasets, where VRUs are detected with difficulty. This may be due to the appearance of radial velocity and RCS as additional channels in the RADAR point clouds. After all, these features can be an

[2]Moreover, in KITTI the difficulty includes the level of truncation as an additional criterion, but this attribute is not labelled in the View of Delft dataset.

excellent source of information to distinguish the few reflections that fall within cyclists from the static or different-in-terms-of-reflectivity background. Moreover, comparing single-frame and multi-frame versions, the increment between 1 and 3 frames is more notorious than the one between 3 and 5 scans for 3D AP40 across all categories. However, a rising trend among frame versions is perceived. Then, the network is taking advantage of the temporal aggregation of scans.

Then, we average the individual AP40 values to compute mAP, as it was mentioned in Chapter 3. Table 5.2 reflects the behaviour of the baseline network in terms of BEV and 3D mAP at medium difficulty. The main conclusion extracted from this Table is that the temporal aggregation of frames does not show a clear improvement in the BEV space. However, there is a slight increment in 3D mAP with respect to the accumulated number of scans. Therefore, we can claim that the additional temporal information contributes to the refinement of the height estimation of the objects of interest.

| | **mAP40** | |
|---|---|---|
| | **BEV** | **3D** |
| **1 frame** | **45.75** | 38.69 |
| **3 frames** | 45.47 | 40.09 |
| **5 frames** | 45.18 | **41.18** |

Table 5.2: Mean Average Precision results of PointPillars-R for 3D modality and Medium difficulty over the validation set of View of Delft dataset.

After this, we evaluate the performance of our proposal **Painted PointPillars-R** following the same schema as the baseline. We train the single-frame and multi-frame versions following the procedure in Section 5.1. At a glance, it is observed that there is a clear improvement in both ways: comparing version-to-version with the baseline and comparing between number of scans used for temporal aggregation. For example, in the case of 3D AP40 for *Cars*, the metrics increase with respect to the baseline: 32.45 vs. 30.42 in medium difficulty; and increase with respect to temporal aggregation within the proposed architecture: from 32.45 in single-frame mode to 39.64 in 5-frames aggregation mode.

Both improvement trends are consistent among all categories and difficulties and for all temporal-aggregated versions. The overall improvements are summarised in Table 5.3. In an analogous manner to the baseline, we compute mAP for BEV and 3D spaces for medium difficulty in Table 5.4.

It is observed that for the proposed version, mAP40 for all classes increases in both spaces, BEV and 3D, with respect to the baseline results. This means that the semantic information not only improves the height estimation but also contributes to the estimation of the width and length of the objects of interest, which leads to an overall better shape estimation of the bounding box.

Lastly, as the main objective of this pipeline is to perform 3D Object Detection, we compare the results on 3D mAP (medium difficulty) for **PointPillars-R** (in blue dots) and **Painted PointPillars-R** (in orange dots) in Figure 5.1. In the X-axis, we plot the three different versions

| | BEV | | | 3D | | | AOS | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Easy** | **Med** | **Hard** | **Easy** | **Med** | **Hard** | **Easy** | **Med** | **Hard** |
| **AP40 Car (IoU @ 0.5)** | | | | | | | | | |
| **1 frame** | 45.27 | 41.03 | 33.85 | 32.95 | 32.45 | 26.00 | 24.65 | 25.79 | 20.80 |
| **3 frames** | 51.41 | 39.59 | 32.87 | 43.05 | 33.56 | **32.87** | 33.55 | 26.23 | **27.15** |
| **5 frames** | **62.59** | **51.95** | **43.16** | **47.21** | **39.64** | 32.00 | **33.83** | **30.20** | 24.73 |
| **AP40 Pedestrian (IoU @ 0.25)** | | | | | | | | | |
| **1 frame** | 39.62 | 34.90 | 31.29 | 32.37 | 28.64 | 25.31 | 20.38 | 18.35 | 16.17 |
| **3 frames** | 44.51 | 40.85 | 36.94 | 40.06 | 35.66 | 31.98 | 28.71 | 25.60 | 23.11 |
| **5 frames** | **53.80** | **50.08** | **44.65** | **47.23** | **43.33** | **38.85** | **31.90** | **29.52** | **26.26** |
| **AP40 Cyclist (IoU @ 0.25)** | | | | | | | | | |
| **1 frame** | 75.76 | 71.01 | 64.25 | 68.27 | 62.28 | 55.76 | 46.75 | 41.94 | 36.95 |
| **3 frames** | 79.70 | 75.69 | 69.31 | 77.36 | 73.24 | 66.67 | 65.23 | 60.42 | 54.43 |
| **5 frames** | **81.97** | **77.71** | **70.85** | **79.44** | **75.03** | **67.91** | **64.21** | **59.36** | **53.34** |

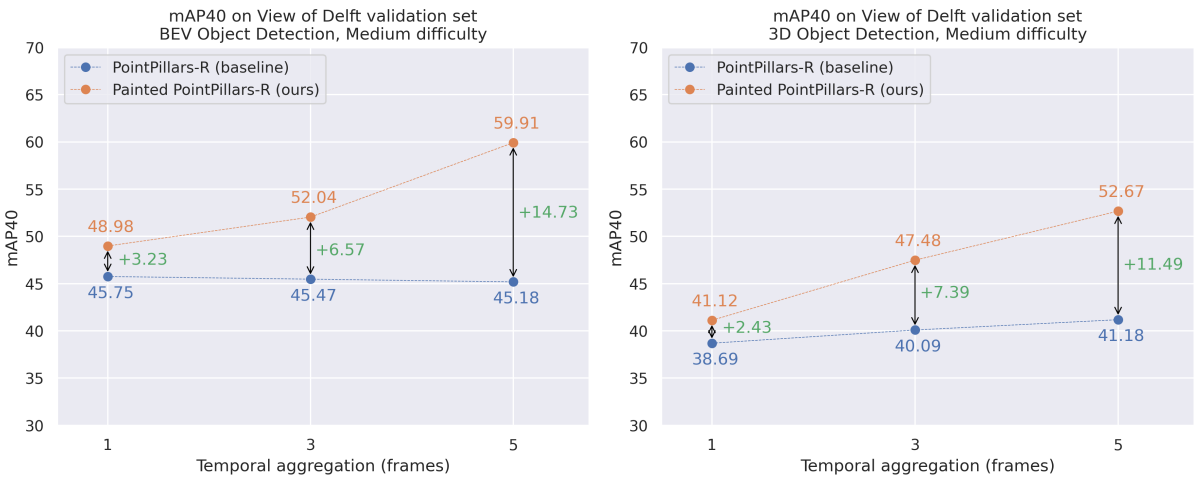
Table 5.3: Quantitative results of Painted PointPillars-R with IoU thresholds (0.5, 0.25, 0.25) over the validation set of View of Delft dataset.

| **mAP40 All Classes** | | |
|---|---|---|
| | **BEV** | **3D** |
| **1 frame** | 48.98 | 41.12 |
| **3 frames** | 52.04 | 47.48 |
| **5 frames** | **59.91** | **52.67** |

Table 5.4: Mean Average Precision results of Painted PointPillars-R for 3D modality and Medium difficulty over the validation set of View of Delft dataset.

depending on the temporal aggregation horizon (number of frames). In the Y-axis, we plot the metrics extracted from Tables 5.2 and 5.4. We remark on the incremental improvement between the baseline and our proposal for all temporal-aggregated versions of the dataset: $\Delta$mAP $= (+3.23, +6.57, +14.73)$ for BEV space (Fig. 5.1a) and $\Delta$mAP $= (+2.43, +7.39, +11.49)$ for 3D space (Fig. 5.1b). Moreover, it is shown that this delta increases with respect to the temporal aggregation of scans.

With these experiments, we can validate the research hypothesis from which this work started. The addition of semantics from a camera image increases the performance of RADAR-based object detectors. These semantics are introduced as high-level feature vectors and they are proven to be effective and contribute to the improvement of network performance without several structural changes. We show the potential of RADAR-camera fusion for 3D Object Detection and there is still room for improvement and research opportunities to reach a LiDAR-like performance at a much lower production cost.




(a) Comparison for BEV Object Detection.

(b) Comparison for 3D Object Detection

Figure 5.1: Comparison on Mean Average Precision (40 points) between PointPillars-R and Painted PointPillars-R over the View of Delft dataset validation set.

## 5.3. Qualitative results on View of Delft dataset

During this Section, qualitative results are presented for the model **Painted PointPillars-R** in the version of 5 frames accumulated of the VoD dataset. Each Figure consists of five Subfigures. Each of them represents a) the camera from the monocular image, b) the segmentation mask from **YOLOv8x-seg**, c) an overlay of the image and the mask, d) the RADAR point cloud projected over the image plane and with semantic and colour information aggregated, e) the result of applying **Painted PointPillars-R** to the enriched and temporal-aggregated RADAR point cloud seen from BEV perspective. In e), the solid boxes represent the ground-truth boxes and the dashed boxes, the predictions. The color code represents: red for *Pedestrian* category, green for *Cyclist*s, and blue for *Car*s. The point cloud points follow another colour code: whilst blue denote points from more recent scans and light green points refer to older points. Due to this, objects with a uniform motion pattern leave a trace behind them that goes from greneer colour to bluer colours.

In Fig. 5.2, a use common use case within the dataset is presented. A narrow urban street with one single direction, vehicles parked at the left hand of the ego-vehicle, pedestrians walking at the right hand of the ego-vehicle and a moving cyclist on the drivable road. In Fig. 5.2e, a trace can be observed behind the cyclist, showing that it is moving frontwards. The algorithm is able to perceive correctly in this case. *Car*s at long range are not perceived correctly at coordinates $\approx (7, 40)$ m, $\approx (8, 47)$ m, and $\approx (20, 47)$ m, since there are no RADAR reflections within their bounding boxes. A single point with the semantics of *Pedestrian* is enough to propose a bounding box at coordinates $\approx (10, -3)$ m. This use case is also represented in Fig. 5.3 in which the network correctly perceives all the four parked vehicles aligned with the road at coordinates $x \approx 3$ m.

Figure 5.4 represents a situation in which the camera has perceived bicycles and that information is propagated to the point cloud (refers to the COCO pre-trained weights issue mentioned in Chapter 4). Moreover, the network does not predict a *Cyclist*, since it does not have information about *Person*s detected in its neighbourhood. The network may have learnt that a *Cyclist* contains information in the *Person* and *Bicycle* channels. Another issue to remark on is that the network still shows difficulties in detecting *Car*s at long range or without aligned orientation, as can be seen in coordinates $\approx (3, 22)$ m and $\approx (38, 8)$ m in Figure 5.4e.

Another common problem can be identified in Fig. 5.5. *Car*s are parked on the left side of the street. As 5.5e shows, there are no RADAR reflections nor semantics propagated in that part of the street. The object is occluded for RADAR since it is positioned in the front bumper of the ego-vehicle. Positioning the sensor on the top roof may solve this issue.

Lastly, Fig. 5.6 justifies the good performance on the *Cyclist* category. One of the main characteristics of the VoD dataset is that this category is well-represented and there are frames in which is the only category present. An atypical use case in automotive datasets that takes place due to the location in which VoD is recorded: The Netherlands.

## 5.4. Inference time analysis

During this Section, an analysis of the inference time of the whole framework will be performed. We will test the inference time of all the components separately, performing cycles of 1000 executions and time measurements for each of them. The experiments are done in a Personal Desktop Computer equipped with an NVIDIA RTX4090 24GB GPU and an Intel Core i9-13900KF processor. The inference of the networks occurs in GPU, then the execution time is measured via `torch.cuda.Events`. The processes that occur in CPU are measured by the `Python` standard library, `time`.

Firstly, we will test the inference time of the segmentation network, **YOLOv8-seg**. For this study, we will compare all versions: from *nano* (n) to *extra-large* (x). Three processes are taken into account: preprocessing (resize operation), inference and postprocessing (NMS). In the final pipeline, version **x** is used. Table 5.5 reflects the results. Processing steps consume a fixed amount of time, while inference time increases from 1.7 ms in the lightest version to 6.8 ms in the heaviest one.

| Model | Preprocessing | Inference | Postprocessing | Total |
|---|---|---|---|---|
| **YOLOv8n-seg** | 0.8 ms | 1.7 ms | 0.5 ms | 3.0 ms |
| **YOLOv8s-seg** | 0.7 ms | 2.1 ms | 0.5 ms | 3.3 ms |
| **YOLOv8m-seg** | 0.7 ms | 3.9 ms | 0.5 ms | 5.1 ms |
| **YOLOv8l-seg** | 0.7 ms | 5.5 ms | 0.5 ms | 6.7 ms |
| **YOLOv8x-seg** | 0.8 ms | 6.8 ms | 0.5 ms | 8.1 ms |

Table 5.5: Inference time for YOLOv8-seg series.

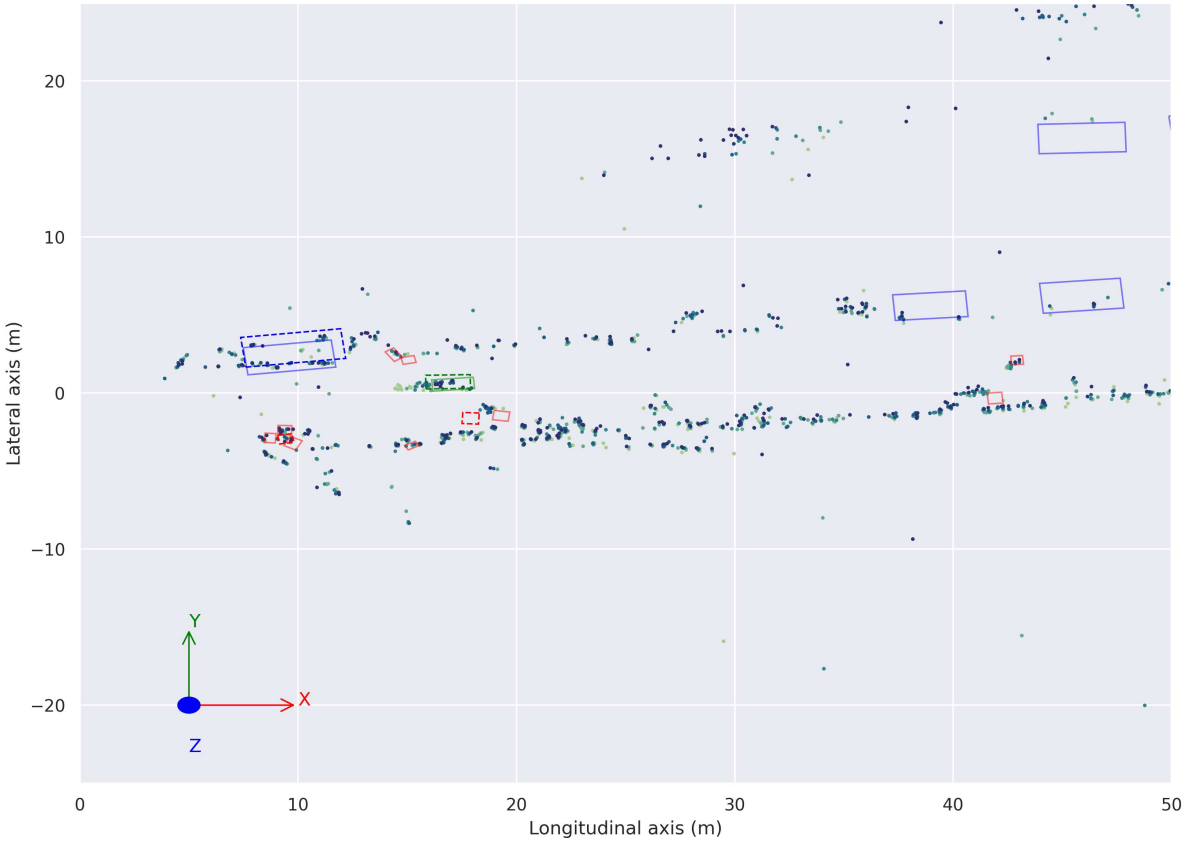
Subsequently, the following steps in the pipeline are the **PointPainting** geometric fusion and the cluster refinement algorithm. The first one contains an *upsampling* operation, so we

(a) Camera image for frame `00170`.

(b) **YOLOv8x-seg** mask for frame `00170`.

(c) Image and mask as overlay for frame `00170`.

(d) After point cloud projection for frame `00170`.




(e) Object detection in BEV space for frame `00170` with **Painted PointPillars-R**. Bounding boxes are: solid for ground-truth and dashed for predictions; red for *Pedestrian*s, green for *Cyclist*s, and blue for *Car*s. Older points are greener and more recent points are bluer.

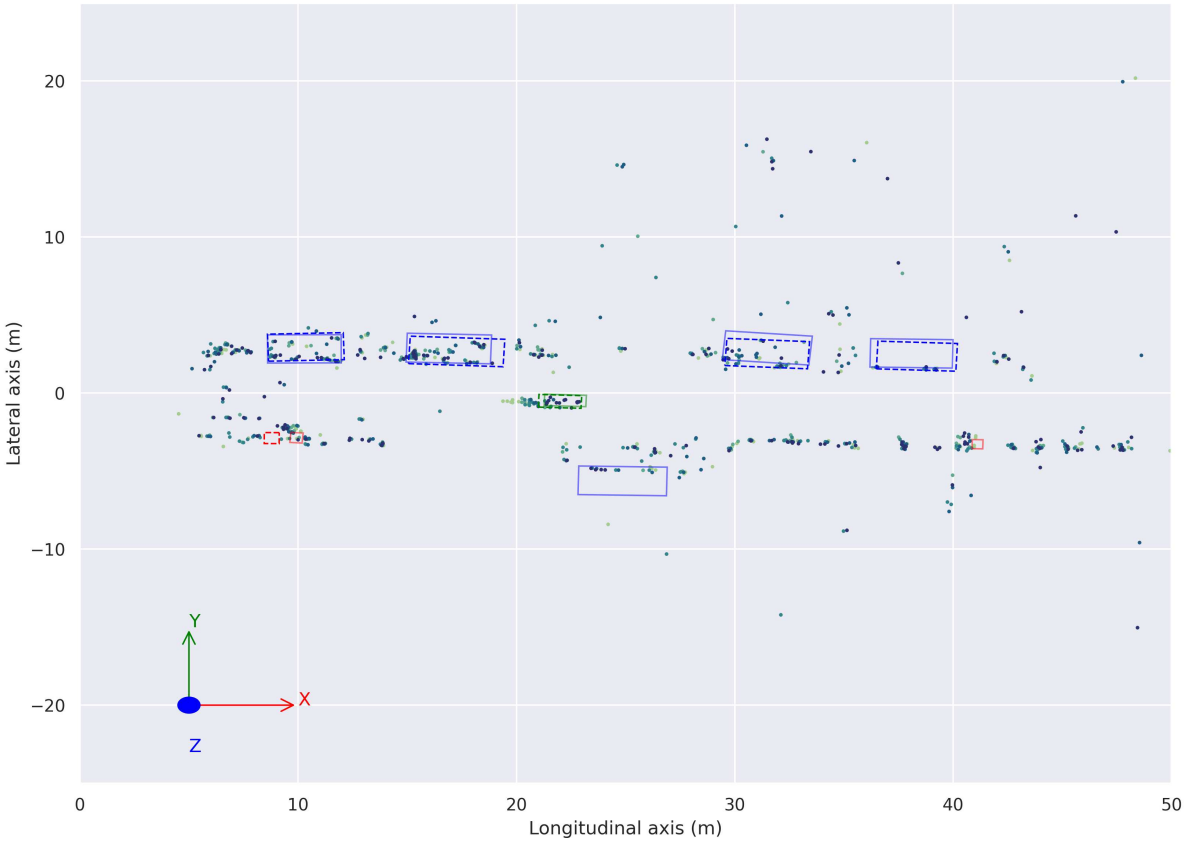
Figure 5.2: Qualititative results for object detection at frame `00170`.

(a) Camera image for frame 00392.

(b) **YOLOv8x-seg** mask for frame 00392.

(c) Image and mask as overlay for frame 00392.

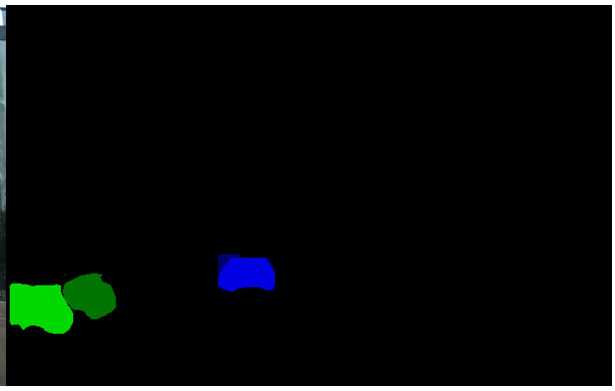
(d) After point cloud projection for frame 00392.




(e) Object detection in BEV space for frame 00392 with **Painted PointPillars-R**. Bounding boxes are: solid for ground-truth and dashed for predictions; red for *Pedestrians*, green for *Cyclists*, and blue for *Cars*. Older points are greener and more recent points are bluer.

Figure 5.3: Qualititative results for object detection at frame 00392.

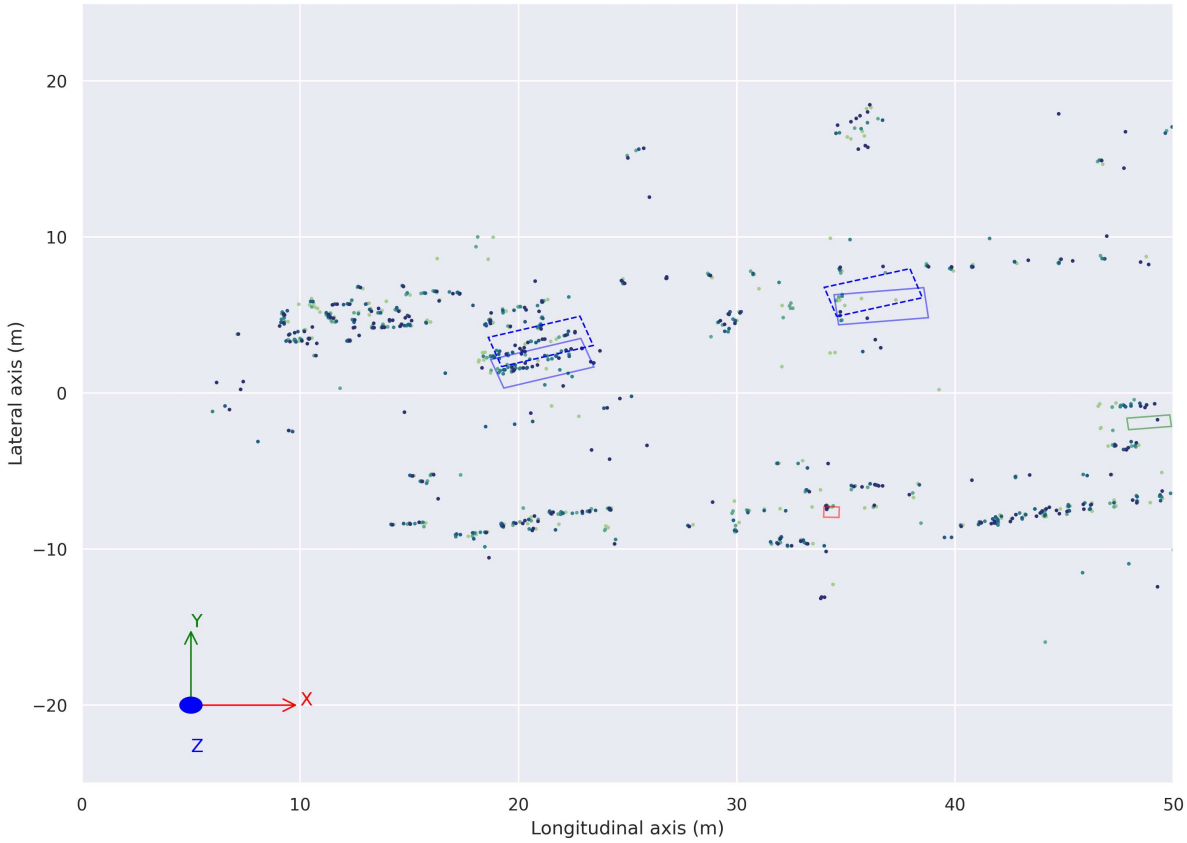
(a) Camera image for frame `01391`.



(b) **YOLOv8x-seg** mask for frame `01391`.

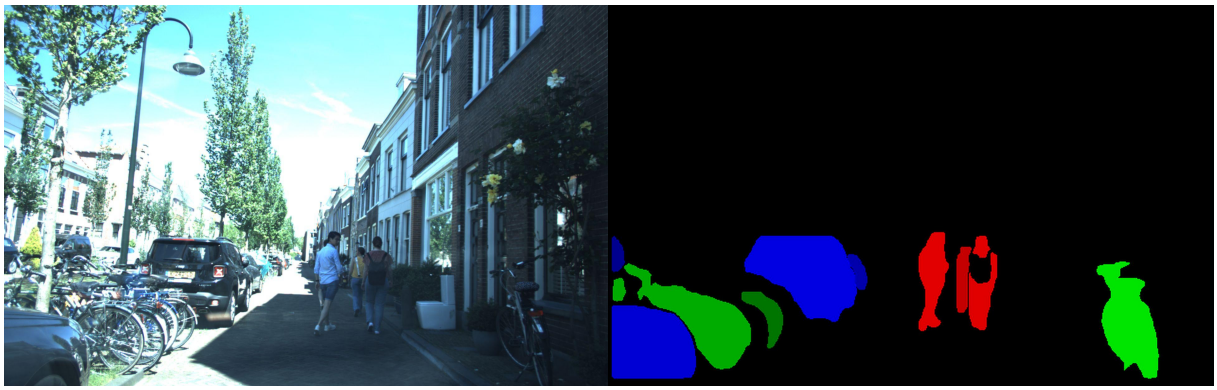
(c) Image and mask as overlay for frame `01391`.

(d) After point cloud projection for frame `01391`.



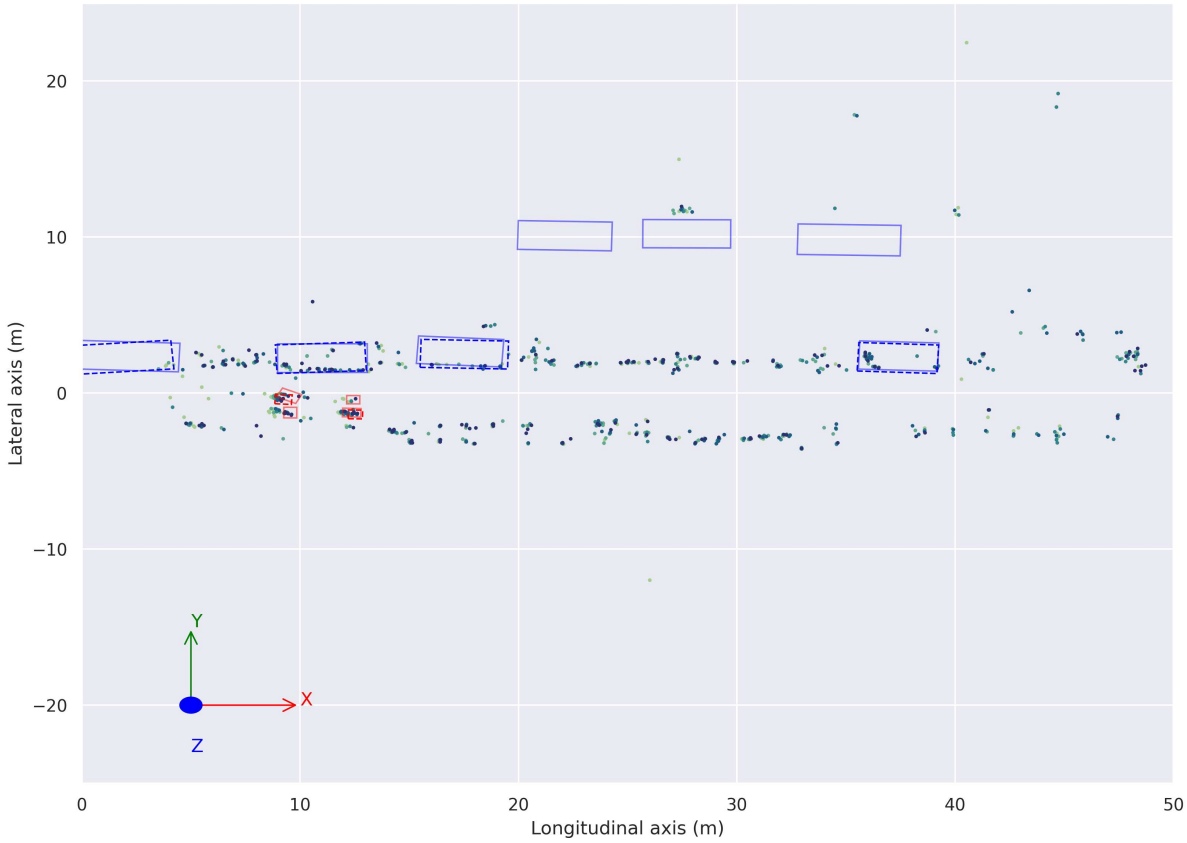

(e) Object detection in BEV space for frame `01391` with **Painted PointPillars-R**. Bounding boxes are: solid for ground-truth and dashed for predictions; red for *Pedestrian*s, green for *Cyclist*s, and blue for *Car*s. Older points are greener and more recent points are bluer.

Figure 5.4: Qualititative results for object detection at frame `01391`.



(a) Camera image for frame 03970. (b) **YOLOv8x-seg** mask for frame 03970.

(c) Image and mask as overlay for frame 03970. (d) After point cloud projection for frame 03970.

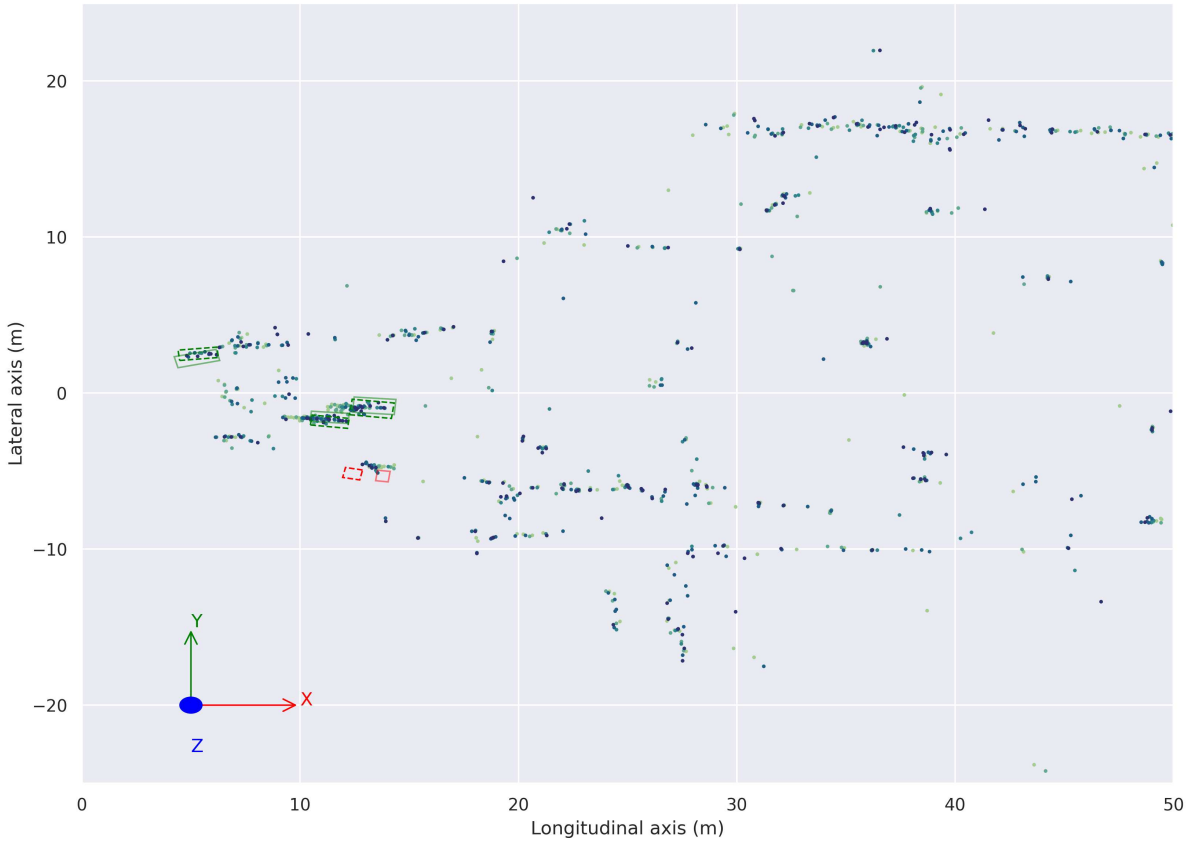



(e) Object detection in BEV space for frame 03970 with **Painted PointPillars-R**. Bounding boxes are: solid for ground-truth and dashed for predictions; red for *Pedestrian*s, green for *Cyclist*s, and blue for *Car*s. Older points are greener and more recent points are bluer.

Figure 5.5: Qualititative results for object detection at frame 03970.

(a) Camera image for frame `04362`. (b) **YOLOv8x-seg** mask for frame `04362`.

(c) Image and mask as overlay for frame `04362`. (d) After point cloud projection for frame `04362`.




(e) Object detection in BEV space for frame `04362` with **Painted PointPillars-R**. Bounding boxes are: solid for ground-truth and dashed for predictions; red for *Pedestrian*s, green for *Cyclist*s, and blue for *Car*s. Older points are greener and more recent points are bluer.

Figure 5.6: Qualititative results for object detection at frame `04362`.

measured a mean time of 7.28 ms, with a standard deviation of 0.25 ms. The latter one contains several runs of DBSCAN clusterings. We measured a mean time of 14.3 ms and a standard deviation of 0.36 ms. Then, we test the inference time of **PointPillars-R** and **Painted PointPillars-R** for the three versions of the VoD dataset. The results are reported in Table 5.6. It is observed that all models infer at an approximate time of 6 ms, and there are no variations among the baseline and our proposal or among dataset versions.

| Model | Dataset version | Mean inference time | Std. deviation inference time |
|---|---|---|---|
| **PointPillars-R** | 1 frame | 6.15 ms | 0.27 ms |
| | 3 frames | 6.15 ms | 0.25 ms |
| | 5 frames | 6.14 ms | 0.28 ms |
| **Painted PointPillars-R** | 1 frame | 6.27 ms | 0.29 ms |
| | 3 frames | 6.29 ms | 0.30 ms |
| | 5 frames | 6.34 ms | 0.31 ms |

Table 5.6: Inference time for PointPillars-R and Painted PointPillars-R models.

Finally, choosing **YOLOv8x-seg** and **Painted PointPillars-R** for 5 frames temporal-aggregated dataset version as components of our proposed framework, the total inference time is: **35.94 ms**, or 27.82 Hz. Therefore, the solution is suitable for real-time systems, even when there is room for further optimization in future works.

# Chapter 6

# Conclusions and Future Works

*A scientist must take the liberty to raise any question, to doubt any assertion, to correct errors.*

J. Robert Oppenheimer

The purpose of this last Chapter is to reflect and draw conclusions from the implemented methods and the theoretical studies that have been carried out during this Master's Thesis project. Moreover, the difficulties and gaps found during the development of the work offer new possibilities and research opportunities in the domain of DL based Object Detection via Camera-RADAR fusion, a research line that has been opened at RobeSafe Research Group[1] with this Master's thesis, either through the continuation of the author himself or of the future students of the group.

## 6.1. Conclusions

To reflect on the author's learning during the development of the full work:

- As for the software tools and resources used, it has been the first research-oriented project in which I have dealt with DL techniques. It has been an enriching experience to learn `PyTorch` and frameworks such as `mmdetection3d` and `ultralytics`, which can be useful for upcoming research projects. Moreover, the whole ML software stack in **Python** has been revisited and applied to a research project: `numpy`, `matplotlib`, `seaborn`, `pandas` and `scikit-learn`.

Conclusions drawn from the practical implementation, the experiment and results of the method proposed within this Thesis:

- SOTA methods across Object Detection for 2D images and 3d point clouds have been revisited during Chapter 3. A comprehensive review of the fundamentals on both research

[1] Official website of RobeSafe Research Group.

lines has served the reader to be contextualized within the problem of perception in the automotive domain.

- The proposed methods have shown that RADAR-camera fusion have a great potential for 3D Object Detection, improving the performance across all classes and categories over the original RADAR-only baseline. Projection-based fusion methods, even being naive geometrical fusion methods, contribute to the performance of detection networks that were not thought for this purpose without deep structural changes.
- The temporal aggregation of information offers additional information to NNs and improves their performance in terms of precision.
- The performance of the network with respect to the `Cyclist` category is surprising. *Pedestrians* and *cyclist*, as main representatives of the vulnerable road users collective, are often underrepresented in automotive datasets. The high presence of this category is biased due to the popularity of cycling in The Netherlands, but this study shows that incrementing the presence of these VRUs increases the performance of networks. This can be also given by the fact that the radial velocity and RCS features give motion and appearance information as additional inputs to the network and cyclists can be seen as dynamic actors, with few reflections, but enough to distinguish them from the static background.
- The elimination of data augmentation is translated into worse convergence during the training stage. It has been claimed that data augmentation is a critical feature for obtaining good performance in KITTI-like datasets. The VoD dataset is smaller than the original KITTI. As aforementioned, actual data augmentation does not take into account the physical issues related to radial velocity. New data augmentation techniques must be proposed in this line, as augmentation is proven to be beneficial for DL in most modalities and domains as a general rule.
- The validation set of VoD presents a great sensitivity. The selection of sequences is focused on urban scenes in a small urban environment. Some of the sequences from which the frames are recorded are in the same physical location and this leads to difficulties in obtaining generalization capabilities with DL-based techniques.

## 6.2. Future Works

During the development of this work, additional research opportunities have been identified and could lead to better-performing architectures based on the fusion of RADAR and camera. The growing interest on RADAR-based DL and RADAR sensors with denser outputs could take advantage of the following:

- We would like to further prove our claims in more datasets that include High-Res 3+1D RADARs, as some additional datasets are expected to be released in the near future. The

validation set of the VoD dataset has presented high sensitivity during the training and evaluation procedures, so we would like to further experiment in other data domains.

- We would like to expand our study and include more 3D Object Detectors present in SOTA and studied in Chapter 3. Revisiting existing detectors could lead to insights or novel conclusions referring to which type of object detector fits better or extracts the most potential from RADAR data.
- We would like to test the potential of multi-frame aggregated networks that are built to deal specifically with temporal aggregation, as the accumulation of scans has been shown to be beneficial in the proposed architecture.
- We would like to explore and propose novel architectures within this domain delving into the way of fusing information inside NNs in a feature-level fusion. These approaches could take advantage of attention mechanisms such as cross-attention modules, in which features from different data modalities can interact between them.
- We would like to perform additional experiments that allow us to compare an Object Detection framework based on LiDAR or LiDAR-camera fusion versus our proposal based on RADAR and camera. We would like to quantify the performance gap between both modalities and analyze the status of both research trends.
- We would like to further optimize our proposal with DL acceleration frameworks and GPU implementations of the actual CPU processes. Then, we will be able to integrate and test our algorithms in the real AD stack located in the electric AV of RobeSafe Research Group.

# Bibliography


[1] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4604–4612.

[2] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 11 784–11 793.

[3] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond", *arXiv preprint arXiv:2304.00501*, 2023.

[4] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector", in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[5] O.-R. A. D. ( Committee, *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems*. SAE International, 2014.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[7] M. Bai and R. Urtasun, "Deep watershed transform for instance segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5221–5229.

[8] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 3354–3361.

[9] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, "Nuscenes: A multimodal dataset for autonomous driving", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[10] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2446–2454.

[11] A. Palffy, E. Pool, S. Baratam, J. F. Kooij, and D. M. Gavrila, "Multi-class road user detection with 3+ 1d radar in the view-of-delft dataset", *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4961–4968, 2022.

[12] L. Zheng, Z. Ma, X. Zhu, *et al.*, "Tj4dradset: A 4d radar dataset for autonomous driving", in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2022, pp. 493–498.

[13] L. N. Smith, "Cyclical learning rates for training neural networks", in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 464–472.

[14] S. Rao, "Introduction to mmwave sensing: Fmcw radars", *Texas Instruments (TI) mmWave Training Series*, pp. 1–11, 2017.

[15] A. Palffy, J. Dong, J. F. Kooij, and D. M. Gavrila, "Cnn based road user detection using the 3d radar cube", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1263–1270, 2020.

[16] Y. Zhou, L. Liu, H. Zhao, M. López-Benıtez, L. Yu, and Y. Yue, "Towards deep radar perception for autonomous driving: Datasets, methods, and challenges", *Sensors*, vol. 22, no. 11, p. 4208, 2022.

[17] B. Barboy, A. Lomes, and E. Perkalski, "Cell-averaging cfar for multiple-target situations", in *IEE Proceedings F (Communications, Radar and Signal Processing)*, IET, vol. 133, 1986, pp. 176–186.

[18] P. Weber and S. Haykin, "Ordered statistic cfar processing for two-parameter distributions with variable skewness", *IEEE Transactions on Aerospace and Electronic Systems*, no. 6, pp. 819–821, 1985.

[19] M. El Mashade, "Detection performance of the trimmed-mean cfar processor with noncoherent integration", *IEE Proceedings-Radar, Sonar and Navigation*, vol. 142, no. 1, pp. 18–24, 1995.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise", in *kdd*, vol. 96, 1996, pp. 226–231.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Advances in neural information processing systems*, vol. 25, 2012.

[22] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey", *Proceedings of the IEEE*, 2023.

[23] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[24] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[25] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models", *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[26] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[27] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition", *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[28] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges", in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 391–405.

[29] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[30] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", *Advances in neural information processing systems*, vol. 28, 2015.

[32] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[36] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

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

[38] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", *arXiv preprint arXiv:2010.11929*, 2020.

[40] Z. Liu, Y. Lin, Y. Cao, *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows", in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[41] D. Shah. "Mean average precision (map) explained: Everything you need to know". Accessed on: 2023-08-28. (Mar. 2022), [Online]. Available: https://www.v7labs.com/blog/mean-average-precision.

[42] M. Simony, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds", in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[43] W. Ali, S. Abdelkarim, M. Zidan, M. Zahran, and A. El Sallab, "Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud", in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[44] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. De La Escalera, "Birdnet: A 3d object detection framework from lidar information", in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 3517–3523.

[45] A. Barrera, C. Guindel, J. Beltrán, and F. Garcıa, "Birdnet+: End-to-end 3d object detection in lidar bird's eye view", in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2020, pp. 1–6.

[46] K. Chen, R. Oldja, N. Smolyanskiy, *et al.*, "Mvlidarnet: Real-time multi-class scene understanding for autonomous driving using multiple views", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 2288–2294.

[47] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection", in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9627–9636.

[48] Z. Tian, X. Chu, X. Wang, X. Wei, and C. Shen, "Fully convolutional one-stage 3d object detection on lidar range images", *Advances in Neural Information Processing Systems*, vol. 35, pp. 34 899–34 911, 2022.

[49] B. Li, "3d fully convolutional network for vehicle detection in point cloud", in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1513–1518.

[50] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[51] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection", *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[52] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[53] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[54] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", *Advances in neural information processing systems*, vol. 30, 2017.

[55] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 770–779.

[56] M. Ulrich, S. Braun, D. Köhler, *et al.*, "Improved orientation estimation and detection with hybrid object detection networks for automotive radar", in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2022, pp. 111–117.

[57] D. Köhler, M. Quach, M. Ulrich, F. Meinl, B. Bischoff, and H. Blume, "Improved multi-scale grid rendering of point clouds for radar object detection networks", *arXiv preprint arXiv:2305.15836*, 2023.

[58] M. Lippke, M. Quach, S. Braun, *et al.*, "Exploiting sparsity in automotive radar object detection networks", *arXiv preprint arXiv:2308.07748*, 2023.

[59] T. Schreier, K. Renz, A. Geiger, and K. Chitta, "On offline evaluation of 3d object detection for autonomous driving", *arXiv preprint arXiv:2308.12779*, 2023.

[60] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator", in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[61] S. Yao, R. Guan, X. Huang, *et al.*, "Radar-camera fusion for object detection and semantic segmentation in autonomous driving: A comprehensive review", *arXiv preprint arXiv:2304.10410*, 2023.

[62] R. Nabati and H. Qi, "Centerfusion: Center-based radar and camera fusion for 3d object detection", in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 1527–1536.

[63] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection", in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6569–6578.

[64] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection", in *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, IEEE, 2019, pp. 1–7.

[65] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[66] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[67] K. Bansal, K. Rungta, and D. Bharadia, "Radsegnet: A reliable approach to radar camera fusion", *arXiv preprint arXiv:2208.03849*, 2022.

[68] M. Cordts, M. Omran, S. Ramos, *et al.*, "The cityscapes dataset for semantic urban scene understanding", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[69] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3d object detection", in *2019 16th european radar conference (EuRAD)*, IEEE, 2019, pp. 129–132.

[70] X. Zhao, W. Ding, Y. An, *et al.*, "Fast segment anything", *arXiv preprint arXiv:2306.12156*, 2023.

[71] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network", *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 8, pp. 2647–2664, 2020.

# Appendix A

# Tools and Resources

This appendix contains an enumeration of all the materials and resources used to carry out the development of this project, in both terms of hardware and software.

## A.1. Hardware Tools

A Desktop Personal Computer with the following components have been used:

- Processor Intel i9 13th Gen 13900k.
- GPU NVIDIA RTX4090 24GB.
- RAM Corsair Vengeance 32GB.
- Hard disk SSD Samsung 980 Pro 1TB.
- Rest of components: power supply, monitors, computer case, communication interfaces...

## A.2. Software Resources

- Ubuntu 22.04 LTS operating system.
- Visual Studio Code IDE for programming.
- Python programming language.
- Python ecosystem for ML: `numpy`, `pandas`, `matplotlib`, `seaborn` and `scikit-learn`.
- DL-oriented frameworks: `ultralytics` and `mmdetection3d`.
- Git control version software.
- Docker containerization and virtualization software.
- Overleaf online programming environment for documentation with LaTeX.

# Appendix B

# Budget

This appendix contains an estimate of the total costs necessary to develop the project in the form of a budget. It is composed of the following items.

- Costs related to the use of hardware resources and software applications involved in the project. Extra costs related to the Spanish tax application (21%) are included.
- Costs related to human resources employed to achieve the objectives of this work.

## B.1. Costs related to material resources: hardware and software

To carry out the development of this AI-focused project, it is necessary to use a Desktop Personal Computer (PC) equipped with a Graphical Processing Unit (GPU) that allows the training and inference procedures of the corresponding algorithms. As for the software tools, the landscape of ML frameworks and utilities is characterised to be open source, so there is no monetary charge for the use of any of them. All PC components are amortised over a period of four years. Only one year is considered as expenses related to this project. However, a premium version of Overleaf is used to write this book. A detailed budget of the material resources used is presented in Table B.1.

## B.2. Costs related to personnel

Considering that the author of this work is a graduate in Engineering, with qualifications for the profession of Industrial Technical Engineer, this point is taken as a reference. The average salary for an Engineer in Spain is 30,000.00€ as of 2023[1], which lead to an approximately gross hour rate of 16.00€/h.

The project started in January 2023 and ends in September 2023. During the first six months, the author is combining his master's studies and work projects with the completion of this thesis,

[1]Average salary of an Engineer in Spain in 2023. Via talent.com.

| | CONCEPT | PRIZE |
|---|---|---|
| Hardware | Processor Intel i9 13th Gen 13900k | 649,00€ × 0.25 |
| | GPU NVIDIA RTX4090 24GB | 2.097,00€ × 0.25 |
| | RAM Corsair Vengeance 32GB | 81,00€ × 0.25 |
| | Hard disk SSD Samsung 980 Pro 1TB | 112,00€ × 0.25 |
| | Rest of components | 1.000,00€ × 0.25 |
| Software | Ubuntu Operating System | 0,00€ |
| | Visual Studio Code | 0,00€ |
| | Git, Docker | 0,00€ |
| | Python and tools: PyTorch... | 0,00€ |
| | Overleaf Premium | 33,00€ |
| **TOTAL PRIZE** | | 1.017,75€ |

Table B.1: Breakdown of costs related to material resources: hardware and software

for which its dedication is counted as part-time, 3 hours per day. The months of July, August and September (half a month) are devoted entirely to the completion of this project, so 8 hours per day are counted. It is assumed that each month has 20 working days. Extrapolating the time dedicated by the author to an engineering project would amount to an hour estimation of 760 working hours. These costs are detailed in Table B.2.

| CONCEPT | PRIZE |
|---|---|
| Gross salary of an engineer (760 h.) | 12.160,00€ |
| Employer's contribution by the hiring company (30%) | 3.648,00€ |
| **TOTAL PRIZE** | 15.808,00€ |

Table B.2: Breakdown of costs related to personnel and human resources

## B.3. Total costs

The total costs of the project will be the sum of material-related and human-related costs reflected in Table B.3.

| CONCEPT | PRIZE |
|---|---|
| Material costs: hardware and software | 1.017,75€ |
| Personnel costs | 15.808,00€ |
| **TOTAL PRIZE** | 16.825,75€ |

Table B.3: Total costs of the project

The project amounts to the sum of SIXTEEN THOUSAND EIGHTY HUNDRED AND TWENTY-FIVE EUROS WITH SEVENTY-FIVE CENTS (16.825,75 EUROS), which includes the total costs of material and personnel in gross terms, including employer's contribution and Spanish taxes (21%), for a total duration of 8.5 months and presented in September 2023.

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá