Universidad
Rey Juan Carlos

DOCTORAL THESIS

End-to-end Vision-based Autonomous Driving using Deep Learning

Author

Sergio Paniego Blanco

Supervisor

José María Cañas Plaza

Doctoral Program in Information and Communications Technologies

International Doctoral School

2024

Technology enables us to be more human

Acknowledgements

This doctoral thesis represents the result of many years of work in which I have learned and grown as a person.

First of all, I would like to thank José María, my thesis supervisor, for his constant support and trust throughout these years, which have been fundamental to this achievement.

My family, friends and partners, who have always shown their unconditional support in my journey and who have always tried to guide me in the best way possible along the way. Thanks to all of you, from the first to the last.

I would also like to thank all the people who have accompanied me along this path at work, such as the teachers with whom I have collaborated, other coworkers, such as colleagues who have gone through Kibotics, and students.

I cannot forget the co-authors of the research projects in which I have participated and my final degree and master's thesis students. His passion for the advancement of human knowledge has been a constant source of inspiration and motivation for me.

My gratitude also extends to all the co-authors of the research works of which I have been a part, because their passion and desire to advance human knowledge are something that unites us and that we have reflected in our work.

Finally, I want to express my deepest gratitude to all those people who have believed and believe in the system that has allowed me to advance to this point in my life. I am especially grateful to all the teachers who have guided and taught me over the years, dedicating their best to drive the progress of society. I also want to recognize those who have dedicated their lives to the advancement of science and technology, and in particular, those who believe in the transformative potential of artificial intelligence. His belief in this discipline has been a constant source of inspiration to me, and has strengthened my conviction that AI has the power to drive human progress and solve some of the most pressing challenges of our time.

To all these people, my most sincere thanks. This work is yours as much as mine.

Español:

Esta tesis doctoral es la representa el resultado de muchos años de trabajo en los que

he aprendido y he crecido como persona.

En primer lugar, me gustaría agradecer a José María, mi supervisor de tesis por su constante apoyo y confianza a lo largo de estos años que han sido fundamentales para este logro.

Mi familia, amigos y parejas, que siempre han mostrado su apoyo incondicional en mi trayectoria y que siempre han intentado guiarme de la mejor forma posible a lo largo del camino. Gracias a todos vosotros, desde el primero al último.

También me gustaría agradecer a todas las personas que laboralmente me han acompañado a lo largo de este camino, como los profesores con los que he colaborado, otros compañeros de trabajo, como los compañeros que han pasado por Kibotics, y alumnos.

No puedo olvidar a los coautores de los trabajos de investigación en los que he participado y mis estudiantes de trabajo de fin de grado y máster. Su pasión por el avance del conocimiento humano ha sido una fuente constante de inspiración y motivación para mí.

Mi agradecimiento se extiende también a todos los coautores de los trabajos de investigación de los que he formado parte, porque su pasión y ganas de avanzar en el conocimiento humano son algo que nos une y que hemos reflejado en nuestros trabajos.

Por último, quiero expresar mi más profundo agradecimiento a todas aquellas personas que han creído y creen en el sistema que me ha permitido avanzar hasta este punto en mi vida. Agradezco especialmente a todos los profesores que me han guiado y enseñado a lo largo de los años, dedicando lo mejor de sí para impulsar el progreso de la sociedad. También quiero reconocer a aquellos que han dedicado sus vidas al avance de la ciencia y la tecnología, y en particular, a aquellos que creen en el potencial transformador de la inteligencia artificial. Su fe en esta disciplina ha sido una fuente constante de inspiración para mí, y ha fortalecido mi convicción de que la IA tiene el poder de impulsar el progreso humano y resolver algunos de los desafíos más apremiantes de nuestra época.

A todas estas personas, mi más sincero agradecimiento. Este trabajo es vuestro tanto como mío.

Abstract

This thesis presents several significant contributions to the domain of AI-driven robotics with computer vision in the application domain of autonomous driving. A key factor in this work is the use of state-of-the-art deep learning techniques, which form the basis of our contributions.

One initial contribution is our solution for traffic monitoring, TrafficSensor, using deep learning for vehicle detection. From this project, we understood the robustness that deep learning provides to perception and the importance of the efficient assessment of candidate solutions for the advancement of the field.

To address this, we developed Detection Metrics, an open-source tool designed for the comprehensive and automated assessment of deep learning visual object detection models. Our experimental validation demonstrates its efficacy in both traffic monitoring and the perception modules within autonomous driving systems.

Building upon this basis, we developed Behavior Metrics, another open source software tailored for the online assessment of autonomous driving systems using simulation. This tool facilitates detailed evaluation of autonomous driving systems for different tasks, generating fine-grained metrics for the quantitative evaluation of the solutions. It supports different autonomous driving tasks that include lane following, driving in traffic and point-to-point navigation. It is focused on the massive and unattended automatic assessment of solutions.

A key aspect of our contributions to the autonomous driving field lies in employing end-to-end vision-based approaches coupled with imitation learning and deep learning. These methodologies are rigorously validated through experimental testing. From this baseline and beyond achieving the basic visual follow lane application using imitation learning, we study the implications of the addition of visual memory and kinematic data to some shallow deep learning models to understand how we can enhance their behavior in a simple follow lane task, yielding new models with enhanced capabilities.

Furthermore, we explored the optimization of deep learning models for driving autonomously to enhance both speed and efficiency without compromising quality. This research aims to produce control models capable of maintaining high performance when piloting the vehicle while being faster and more resource-efficient. For this contribution, we thoroughly investigated various optimization techniques and conducted an in-depth

analysis of how they each contribute to the final driving system.

Finally, we present an innovative vision-based methodology leveraging imitation learning and deep learning to facilitate safe autonomous driving in complex traffic environments. This approach facilitates the creation of adaptable models adept at navigating a spectrum of traffic scenarios while seamlessly extrapolating to situations never seen during training. By employing familiar shallow deep learning models with slight modifications, we substantially broaden their utility and effectiveness in diverse contexts.

Resumen

Esta tesis presenta contribuciones significativas en el dominio de la robótica impulsada por IA con visión por computador en el ámbito de la conducción autónoma. Un factor clave en este trabajo es el uso de técnicas de aprendizaje profundo del estado de la cuestión, que forman la base de nuestras contribuciones.

Una contribución inicial es nuestra solución para monitoreo de tráfico, TrafficSensor, que aprovecha el aprendizaje profundo para la detección de vehículos. A partir de esta contribución, entendemos la robustez que el aprendizaje profundo aporta a la percepción y la importancia de la evaluación eficiente de las posibles soluciones para el avance del campo.

Para abordar esto, desarrollamos Detection Metrics, una herramienta de código abierto diseñada para la evaluación integral y automatizada de modelos de detección de objetos visuales de aprendizaje profundo. Nuestra validación experimental demuestra su eficacia tanto en el seguimiento del tráfico como en los módulos de percepción dentro de los sistemas de conducción autónoma.

Sobre esta base, desarrollamos Behavior Metrics, otro software de código abierto diseñado para la evaluación en línea de sistemas de conducción autónoma mediante simulación. Esta herramienta facilita la evaluación detallada de los sistemas de conducción autónoma para diferentes tareas, generando métricas detalladas para la evaluación cuantitativa de las soluciones. Admite diferentes tareas de conducción autónoma como seguimiento de carril, conducción con tráfico y navegación punto a punto. Una vez más, este software se centra en la evaluación automática, masiva y desatendida de soluciones.

Un aspecto clave en las contribuciones que presentamos en el campo de la conducción autónoma radica en el empleo de enfoques basados en la visión extremo a extremo junto con el aprendizaje por imitación y el aprendizaje profundo. Estas metodologías están rigurosamente validadas mediante pruebas experimentales. A partir de esta idea base y más allá de generar la aplicación visual básica de seguimiento de carril mediante aprendizaje por imitación, estudiamos las implicaciones de incluir memoria visual y datos cinemáticos a algunos modelos de aprendizaje profundo estrechos para comprender cómo podemos mejorar su comportamiento en una tarea simple de seguimiento de carril, produciendo modelos nuevos con capacidades mejoradas.

Además, exploramos la optimización de modelos de aprendizaje profundo para con-

ducir de forma autónoma con idea de mejorar tanto la velocidad como la eficiencia sin comprometer la calidad. Esta contribución tiene como objetivo producir modelos de control capaces de mantener un alto rendimiento y al mismo tiempo ser más rápidos y eficientes en el uso de recursos. Para esta contribución, investigamos exhaustivamente varias técnicas de optimización y realizamos un análisis detallado de cómo cada una contribuye en el sistema de conducción final.

Finalmente, proponemos un enfoque basado en la visión con aprendizaje por imitación y aprendizaje profundo para una conducción autónoma segura en escenarios complejos con tráfico. Este enfoque permite el desarrollo de modelos adaptables capaces de navegar eficazmente en diversas condiciones de tráfico y que generalizan a situaciones nunca vistas en entrenamiento. Para ello, se utilizan modelos de aprendizaje profundo estrechos previamente conocidos con pequeñas modificaciones, ampliando su rango de aplicación significativamente.

Acronyms

Acronyms used in the thesis:

RL Reinforcement Learning

GPU Graphical Processor Unit

GUI Graphical User Interface

MVC Model-View-Controller

LIDAR Laser Imaging Detection and Ranging

COCO Common Objects in Context

Pascal VOC Pascal Visual Object Classes

CNN Convolutional Neural Network

YOLO You Only Look Once

AWS Amazon Web Services

LSTM Long Sort-Term Memory

ConvLSTM Convolutional Long Sort-Term Memory

RNN Recurrent Neural Network

MSE Mean Squared Error

MAE Mean Absolute Error

KLT Kanade-Lucas-Tomasi

ITS intelligent transportation system

SSD Single-Shot Detector

SSP Spatial Pyramid Pooling

PAN Path Aggregation Network

SORT Simple Online and RealTime Tracker

ROS Robot Operating System

AP Average Precision

AR Average Recall

mAP Mean Average Precision

AR Mean Average Recall

IoU Intersection Over Union

MPD Mean position deviation per km

VJ Vehicle longitudinal jerk per km

PCA Principal Component Analysis

PCQAT Sparsity and cluster preserving quantization aware training

RGB Red, Green, Blue

DARPA Defense Advanced Research Projects Agency

ALVINN Autonomous Land Vehicle In a Neural Network

GPS Global Positioning System

CPU Central Processing Unit

IMU Inertial Measurement Unit

RG Research Goal

TP True positive

TN True negative

FP False positive

FN False negative

NHTSA National Highway Traffic Safety Administration

WHO World Health Organization

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The autonomous driving field aims to generate vehicles that drive safely without human intervention. This field resonates profoundly with considerations of safety, accessibility, and societal advancement. It combines, among others, the fields of robotics, artificial intelligence (AI), and computer vision. It has received enormous attention in recent years, both in academia and industry, and is expected to have a broad impact on day-to-day life in the coming years [1]. The field development could potentially generate a series of benefits, from improved traffic safety and security to more optimized mobility for individuals and freights globally. The important advancements generated in the field come from several sides. In the last few years, the artificial intelligence field has experienced an enormous growth. This progress has been possible thanks to the availability of high-intensity computational units, called graphical processor units (GPUs), curated datasets' high availability, and deep learning (DL) algorithms' development. This leverage has affected many fields but some of the most positively affected have been robotics and computer vision.

The main motivation underpinning the adoption of autonomous driving is its potential to mitigate the prevalent issue of road traffic accidents, predominantly attributable to human errors. According to the National Highway Traffic Safety Administration (NHTSA), around 94% of motor vehicle accidents were caused by human driver errors [2] in a study conducted from 2005 to 2007. Another major problem is road traffic injuries. They caused an estimated 1.35 million deaths worldwide in 2016, as reported by the World Health Organization [3] (WHO). By their continuous environmental monitoring and instantaneous response capabilities, autonomous vehicles hold the promise of substantially reducing the incidence of vehicular accidents, thereby safeguarding human lives and improving public health outcomes. Among the reasons for researching on autonomous driving, we can think about the reduction of the driver's stress, improving productivity and mobility due to the fact of releasing humans from the activity of driving. Regarding costs and property, the idea would be to reduce the operative costs and incorporate shared property of the vehicles. Other associated costs such as parking or pollution would also see a reduction. Monitoring the traffic flow is also an idea worth pursuing to reduce traffic accidents.

Some examples from companies developing autonomous driving solutions already suggest this advancement towards an increase in safety conditions. For example, Waymo, which is a subsidiary of Alphabet Inc (the parent company of Google), has already conducted an experiment where they prove that autonomous driving vehicles are safer than vehicles commanded by a human [4]. The results claim that in 3.8 million miles (around 6 million kilometers) driven, the Waymo Driver incurred zero bodily injury claims in comparison with the human driver baseline of 1.11 claims per million miles (cpmm). Similarly, it reduced property damage to 0.78 cpmm in comparison with the human driver baseline of 3.26 cpmm. Cruise is another important company developing autonomous driving vehicles. They have also released a similar report [5] were they claim a 65% reduction in collisions for their autonomous vehicles in San Francisco in comparison to human ridehail driving.

Furthermore, the advent of autonomous driving heralds a paradigm shift in transportation equity, fostering inclusivity and accessibility for individuals until now constrained by mobility limitations or residing in underserved locales.

1.1. Autonomous driving

This section serves to provide context for the thesis. To achieve it, we break it down into several parts. Initially, we provide a general context about the autonomous driving field. After that, we discuss the necessity for precise metrics when evaluating computer vision and autonomous driving solutions. Following this, we outline the two main approaches to developing autonomous driving systems: end-to-end and modular methods. We then detail the various tasks involved in autonomous driving. Finally, we emphasize the importance of creating optimized deep learning solutions, especially in the context of autonomous driving systems that demand high-quality and swift responses.

1.1.1. Introduction to autonomous driving

In this subsection, we provide a general view of the autonomous driving field. We provide a general view of how autonomous vehicles are mobile robots. After that, we describe the different levels of autonomy and provide some historic breakthroughs that have been of importance for the advancement of the field. Following that, we provide details about the range of applications of autonomous vehicles and we finish by looking at the research community's current developments to understand the trends in the field.

We may consider an autonomous vehicle as a mobile robot in terms of hardware (see Fig. 1.1) and software. A modern vehicle is equipped with a series of advanced sensors and actuators, as a robot. For example in terms of sensors, they typically include GPS, IMUs, LIDARs, ultrasonic sensors, cameras, or radars. In terms of actuators, we can consider the throttle, brake, steering wheel, or turn signals as part of them. These com-

ponents are supplemented by computational units such as CPUs and GPUs, essential for processing the vast amount of data collected by the sensors.

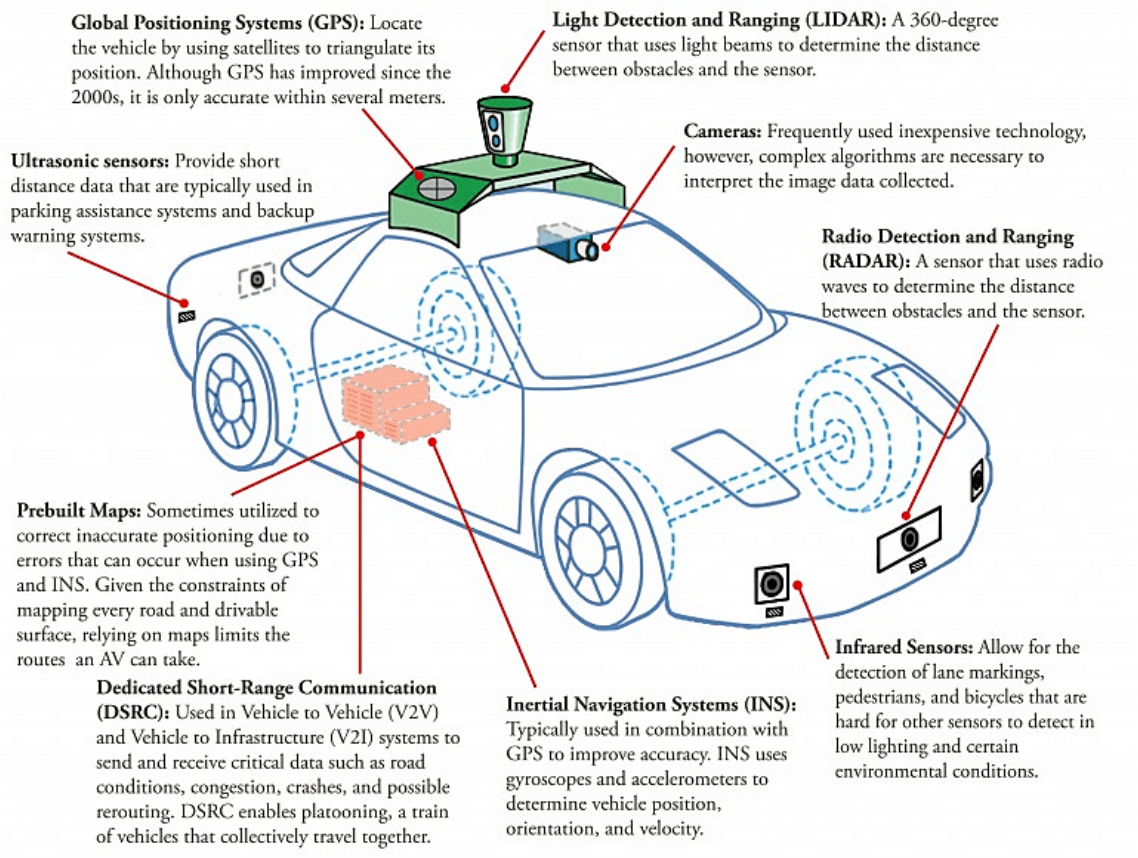


Figure 1.1: Sensor ecosystem in an autonomous driving vehicle. Source [6].

Levels of autonomy

The autonomy of the vehicles is typically categorized into 6 differentiated levels, as described by the SAE J3016 Standard [7], from level 0 where no automation is implemented in the vehicle to level 5 where we can consider the vehicle to be fully autonomous, without the need for human intervention under any circumstances. Presently, some commercial solutions operate at levels 2 to 3, with a few level 4 vehicles already developed by companies (e.g. Waymo, Cruise, Wayve, Tesla, Motional, AutoX, etc.). These solutions encompass various applications, including autonomous taxi services [8], smart automated parking capabilities [9], and driver assistance systems [10] [11]. It is anticipated that advancements in technology will further extend the capabilities and applications of autonomous vehicles in the coming years [12].

Providing further details about the levels of autonomy (see Figure 1.2):

- **[Driver only] Level 0**: No automation is implemented in this level.

- **[Assisted] Level 1**: Almost no automation is implemented. Only one task can be performed automatically at a time, like lane centering or speed control (lateral or longitudinal control).
- **[Partial automation] Level 2**: at this level of autonomy, minimal automation is implemented. The vehicle is capable of performing two tasks autonomously at a time, such as lane centering and adaptive cruise control.
- **[Conditional automation] Level 3**: the automation starts to grow broadly at this level as the vehicle is mostly self-driven and only requires human intervention in extreme environments or in the event of a detected failure.
- **[High automation] Level 4**: no human intervention is needed at this point. The vehicles are completely autonomous but only in certain areas. The most advanced vehicles at this point belong to this category, being fully autonomous but only deployed in certain restricted cities or highly controlled scenarios, like Waymo, Cruise, or Wayve solutions.
- **[Full automation] Level 5**: on the most advanced level, the vehicles can drive autonomously in all conditions, without restrictions on location. However, this level still presents numerous challenges requiring attention from both academia and industry. Given the substantial disparity between levels four and five, further subdivision may be warranted to accurately capture the complexities and advancements within this tier of autonomy.

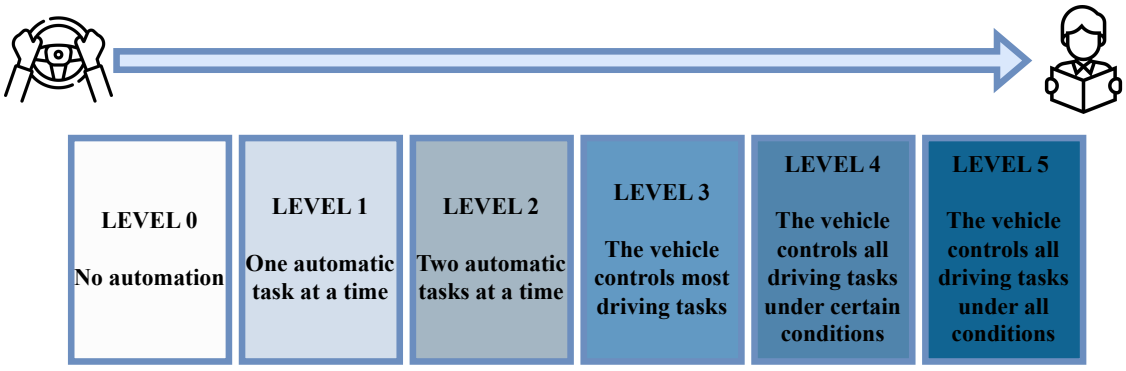


Figure 1.2: Detail of autonomy levels.

History

Although we have already described part of the most recent advances, the history of autonomous driving development began decades ago. The first example of a vehicle controlled by a computer was presented in 1986, called Navlab 1 [1]. Its development started

[1] https://www.youtube.com/watch?v=ntIczNQKfjQ

in 1984 in Carnegie Mellon University. In 1989, ALVINN [13] (Autonomous Land Vehicle In a Neural Network) was released. This project was one of the first that proposed controlling a real vehicle using a shallow neural network. This work was quite limited but still is a grand landmark in the development of autonomous driving solutions.

Another example is the DARPA Grand Challenge [14] and DARPA Urban Challenge, competitions organized by the Defense Advanced Research Projects Agency (DARPA) of the United States. The first edition of this challenge was held in 2004, and its goal was to spur the development of the technologies needed to create the first fully autonomous ground vehicles capable of completing a substantial off-road course within a limited time. In the first edition (DARPA Grand Challenge), none of the robot vehicles finished the route but in the following edition, some of them completed the course with the Stanley [15] vehicle winning the first place (see Fig 1.3 for some example of autonomous driving vehicles). This edition did not incorporate other vehicles into the route, unlike the DARPA Urban Challenge, where other vehicles were included, and they were expected to adhere to human traffic laws and rules.

Figure 1.3: Examples of different autonomous driving vehicles.

In parallel to the development of fully autonomous vehicles, we can also mention the development of advanced driver-assistance systems (ADAS) which are the piece that permits the middle levels of autonomy we mentioned. These systems are common in any modern vehicle and its history goes back to the 1970s. This set of technologies is oriented towards assisting drivers to operate their vehicles safely. As of 2021, the research firm Canalys estimated that around 33% of new vehicles sold in major markets included ADAS features [16]. These robotic technologies include autoparking, lane change, lane keeping, or adaptive cruise control assistance among other examples.

Range of application

In the field of autonomous driving, we can encounter a diverse range of environments of application, each of them requiring particular approaches for success. An idea common to all of them is that safety is a key issue and that the systems must be robust to a vast

range of weather, lighting, and traffic conditions. We can find urban scenarios, roads, or highways but also unstructured scenarios like forests where the needs are different. In the development of the contributions presented in this thesis, we focus on urban, road, and highway scenarios, which are probably the most common approximations that an agent may encounter.

When considering autonomous vehicles, we may think about a typical utilitarian car, but the range of vehicles where these solutions should work is wider. It should also include 4x4, SUVs, motorcycles, trucks, and any other type of vehicle that we may consider to be enhanced by this approach. Although the majority of research efforts are focused on 4-wheeled cars, there are examples of efforts on other types of vehicles, like Waabi with trucks [17] or Navya with logistics solutions and autonomous shuttles [2]. In addition to the type of vehicles, the area of application is also wide. We may consider autonomous taxis to be one of the most relevant examples but the development of autonomous vehicles is applicable in other areas such as logistics, industrial, medical, or even racing applications.

In robotics, there are already solutions for robot navigation that are advanced and reliable like Nav2 [18] that can be used to generate point-to-point navigation in different types of settings and scenarios. This may raise the question of why we need specialized approaches for autonomously driving vehicles instead of using these technologies. While these solutions demonstrate notable capabilities, they may encounter challenges in dynamic environments characterized by rapid movements, such as scenarios involving swiftly moving robotic vehicles that could be driving at 120km/h alongside other dynamic elements like humans and vehicles. In such contexts, the system's ability to respond promptly may be constrained, thereby limiting its applicability for autonomous driving.

Growing research community

Looking at the amount of attention that the field is attracting inside the research community is another way of understanding the incremental advancements that it is experiencing. Many of the principal international conferences on computer vision, robotics and artificial intelligence have included workshops or tutorials to address this topic lately, which indicates a high pace of development and high interest for the problem [3] [4] [5] [6] [7].

The interest in the autonomous driving field, also called self-driving vehicles, is also evidenced in the numerous competitions aimed at advancing it. Notable examples include DuckieTown [19], AWS DeepRacer [20], and F1TENTH [21], among others (see Fig. 1.4.)

[2] https://www.navya.tech/en/
[3] https://waabi.ai/cvpr-2023/
[4] https://opendrivelab.com/challenge2024/
[5] https://cvpr2023.wad.vision/
[6] https://sites.google.com/view/icra2023av/home
[7] https://opendrivelab.com/sr4ad/iclr23

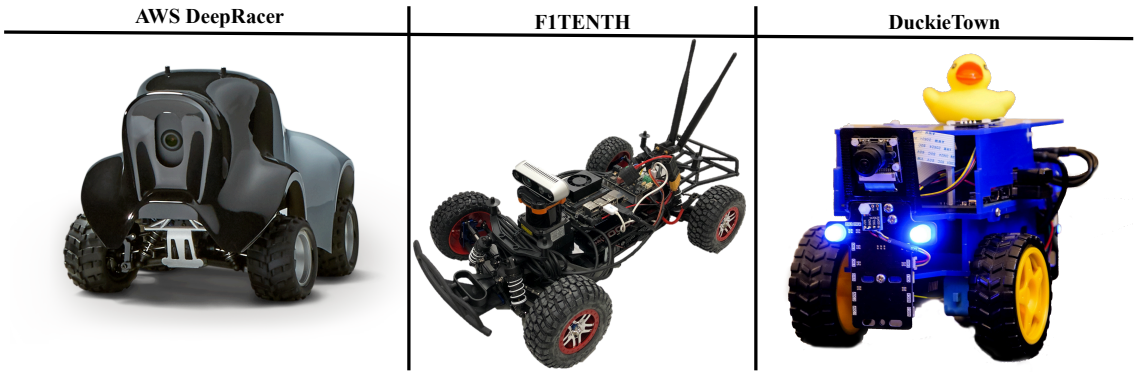


Figure 1.4: Examples of vehicles used in autonomous driving competitions.

Although these ideas are appealing, many challenges remain open, as described in detail in [1]. For example, the advancement and deployment of autonomous driving generates new security and public health issues, like software failures which may lead to catastrophic consequences [22], hacking [23], or the reduction of the use of seat bells due to the false feel of security. Additionally, new stressors are present, like the lack of access to specific places for example due to the lack of maps or complicated weather conditions. As we see, the situations are very heterogeneous with variable dynamics, type of spaces, weather, or light conditions.

Autonomous driving solutions are currently expensive and generate new property forms, like a shared approach. In addition to these examples, the overall cost of autonomous driving technology is not yet proved to be cheaper than the traditional one so more advancements are required to meet that point and make it feasible. Another controversial point is the number of labor forces involved in the actual driving of an autonomous driving vehicle, where sources [24] now point that the amount of people needed to drive a vehicle autonomously is 1.5 per vehicle, where they act as assistants in complicated situations, even increasing the baseline and setting a worse scenario.

1.1.2. Assessment and metrics

The field of deep learning includes a broad spectrum of methodologies aimed at evaluating solutions derived from research. This process is essential to guarantee the robustness, efficacy, and safety of the proposals. For example, each presented dataset usually incorporates a leaderboard mechanism where researchers and practitioners rank their solutions against standardized metrics. This method is used as a catalyst for advancing the field and facilitating the generation of cutting-edge results. Therefore, meticulous attention to the selection and validation of metrics becomes imperative for the advancement of knowledge in this domain.

This idea applies to all the different deep learning areas, such as the ones that are the focus here, computer vision, robotics, and autonomous driving. In the context of object detection and computer vision, where the accurate identification and interpretation

of visual information are fundamental, rigorous evaluation methodologies serve as axles for assessing the performance and generalization capabilities of algorithms across diverse datasets and scenarios. This allows proficiency in tasks such as object recognition, localization, tracking, semantic segmentation, or even autonomous driving.

In the context of this thesis, our major focus revolves around the examination of end-to-end autonomous driving systems, which orchestrate vehicular behaviors based on input data. The rigorous assessment of these behaviors assumes critical significance in elucidating the inherent strengths and vulnerabilities. The input data can include LIDAR, camera, or other sensors. Given the complexity and real-world implications of autonomous driving technologies, rigorous evaluation methodologies are essential to gauge the performance and reliability of these systems across diverse operational scenarios. These evaluation frameworks should be more generalist and also focus on specific driving tasks subjecting these systems to thorough testing under diverse environmental conditions, including variations in weather, lighting, and traffic patterns, researchers can ascertain their robustness and resilience in real-world scenarios. This iterative evaluation process not only fosters continuous improvement in system performance but also instills confidence among stakeholders, including regulators, manufacturers, and end-users, regarding the reliability and safety of autonomous driving technologies.

Inside the evaluation procedures, we distinguish between offline and online evaluation methodologies, with the latter being addressed in depth in this thesis. The offline evaluation (open-loop evaluation) refers to the assessment concerning the common metrics prevalent in deep learning evaluation such as accuracy or mean squared error. The online evaluation (closed-loop evaluation) refers to the assessment of the system in simulation or real scenarios where the system will be deployed. In the online evaluation environment, the system is evaluated within the context of its interaction with the surrounding environment, necessitating consideration of external factors and dynamic changes. Unlike offline evaluation, where system performance is observed in isolation, online evaluation provides insights into how the system's decisions and actions influence and are influenced by the broader environment. This distinction is paramount, as a suboptimal decision made by the system in an online setting can have immediate repercussions, a consequence not readily apparent in offline evaluations. This is addressed in detail in this document.

1.1.3. End-to-end and modular approaches

In the context of developing an autonomous driving solution, there are two common approaches: end-to-end or modular systems (see Fig 1.5 for a diagram of each approach). The majority of the solutions presented in industry and academia implement a modular approach. This approach consists of a series of modules that communicate with each other and that are specialized in certain tasks. These modules are therefore individually developed and integrated into the onboard vehicle [25]. For example, one module can be responsible for generating object detection candidates (bounding boxes) from the input

(camera, LIDAR), another module can be responsible for localizing the vehicle precisely and generating planning paths and a third module can be responsible for the end control of the vehicle. As we can see, the driving task is divided into subtasks that are addressed by each submodule.

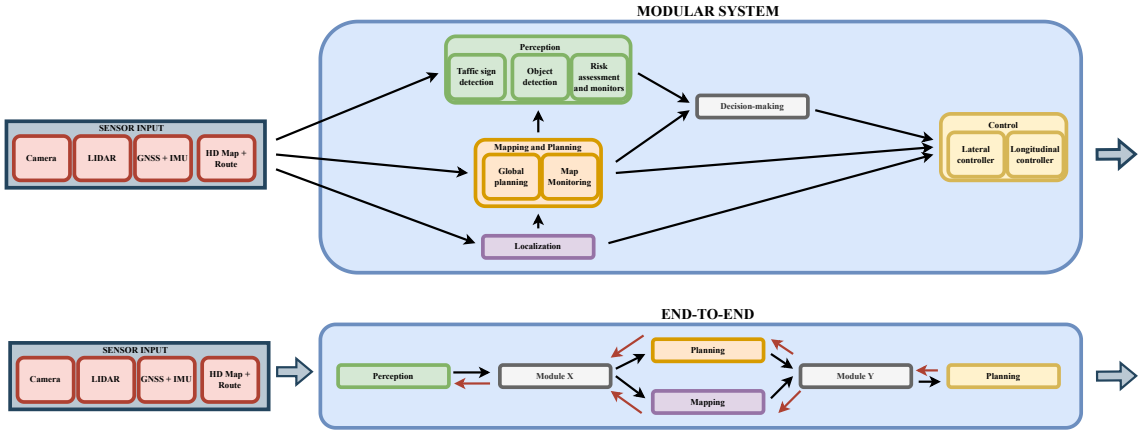


Figure 1.5: Diagram of end-to-end and modular approaches. Adapted from [26] and [25].

On the other hand, there are end-to-end solutions. These solutions directly generate the final control or planning from the raw input data. In this case, the system typically includes a deep learning fully differentiable model that predicts the control commands or planning based on the input data [26]. In this case, the model propagates feature representations across components. The tasks are jointly and globally optimized in this process.

The first approach is typically more prone to the propagation of errors since an error in one module can affect negatively the rest of the subsystems and the final output. These systems are also more complex, leading to a possible sub-optimal use of compute power and they can be redundant. As an advantage, this design presents more interpretability and is easier to debug. In comparison, the end-to-end models are typically more simple and efficient thanks to the combination of tasks into a single model. They are also optimized for a common task, in comparison to the first approach where each sub-system targets one particular problem. They can potentially offer emergent capabilities due to this data-driven optimization paradigm, where scaling the training resources could lead to improvements.

An important consideration is that the end-to-end paradigm does not inherently imply a single black box system solely producing planning or control outputs. Rather, it can adopt a modular structure with intermediate representations and outputs akin to classical approaches. In practice, many cutting-edge systems leverage a modular design while simultaneously optimizing all components collectively, thereby attaining heightened performance levels.

In the context of this thesis, we research end-to-end autonomous driving solutions for different approaches, testing their limitations and presenting innovations while adding pieces to the end-to-end system to increase its range of applicability.

1.1.4. Tasks in an autonomous driving system

In autonomous driving, a multitude of driving subtasks are combined into the final autonomous driving system, encompassing functions such as autonomous parking, lane following, intersection negotiation, and more. Each task presents its own unique set of challenges and addressing all of them with a safe and reliable solution is the final goal of this paradigm. Fundamentally, lane following represents a foundational task, where the vehicle drives while maintaining itself within designated lane markings while adapting to dynamic conditions like turns. From this task, we may move to a more complex task like driving in traffic situations, where the lane following task is complemented by management of traffic flow. Building from this task, we may also include point-to-point navigation or interpretation of traffic signals and signs.

The spectrum of potential tasks and scenarios is vast, with the research community addressing these challenges through platforms like the CARLA Leaderboard [8], where participants address situations like lane merging, lane changing, negotiation at traffic intersections, negotiations at roundabouts, handling of traffic lights and traffic signs, yielding to emergency vehicles or comping with pedestrians, cyclist, and other elements. The complexity inherent in these tasks arises from the need for robust perception, accurate decision-making under uncertainty, and seamless integration of multiple sensory modalities. Moreover, the dynamic and unpredictable nature of real-world driving scenarios further compounds the challenges, necessitating sophisticated algorithms and comprehensive validation frameworks to ensure the reliability and safety of autonomous driving systems. Thus, the multifaceted nature of autonomous driving tasks underscores the complexity and depth of research required to realize fully autonomous vehicles capable of navigating diverse environments with precision and reliability.

In this thesis, we build the end-to-end autonomous driving system from the most fundamental task of lane following and adding more complexity to the environment with other vehicles throughout the document. We explore the tasks in depth to generate a safe solution that is fast and optimized for different environments. We focus on the simplicity of model development to address these tasks.

1.1.5. Optimization in autonomous driving solutions

Inside robotics and particularly in the field of autonomous driving, we need solutions that are fast and reliable. The performance of a robot application not only depends on the quality of the control decision but also their frequency. Ideally, we seek to generate high-quality decisions at a rapid rate. Some autonomous vehicles or robots are equipped with high-performance hardware but there are other systems where this is not true. A possible option in those cases is updating to faster-computing hardware but it is sometimes not feasible. Consequently, solutions for autonomous driving must be optimized while

[8] https://leaderboard.carla.org/

upholding stringent safety standards.

This optimization is also important when thinking about translating a solution from a simulation environment to a real vehicle. This is typically clear when using deep learning models inside autonomous driving systems. Given the disparities in hardware, prior consideration of optimization during the research and development stages of autonomous driving systems can facilitate seamless integration into different hardware configurations. As elucidated in the preceding discussion on online evaluation methodology, the importance of optimized models becomes evident, as suboptimal decisions may precipitate complex scenarios, necessitating the imperative of a swift system.

Within this thesis's scope, we explore optimization techniques tailored specifically for autonomous driving, presenting novel contributions in this realm.

1.2. Traffic monitoring using computer vision

Traffic monitoring plays a pivotal role in urban planning, transportation management, and public safety. It involves the systematic observation and analysis of vehicular movement within traffic scenarios to ensure efficient traffic flow, enhance safety measures, and facilitate informed decision-making.

At its core, traffic monitoring encompasses the classification, identification, and speed measurement of various types of vehicles navigating through diverse traffic conditions. This process involves leveraging advanced technologies such as computer vision, machine learning (deep learning), and sensor networks to accurately detect, classify, and track vehicles in real-time.

One of the primary goals of this field is to provide insight into traffic patterns, congestion areas, and potential safety hazards. This information can be used by transportation authorities to optimize traffic signal timing, plan infrastructure improvements or deploy resources effectively. They are also crucial in enhancing public safety by detecting and responding to traffic violations, accident, and emergencies promptly. Automated surveillance systems equipped with intelligent algorithms can detect anomalies in traffic behavior, such as sudden stops, erratic lane changes, or speeding vehicles, enabling swift intervention by law enforcement agencies or emergency responders.

This field also serves as a gateway to understanding the utility of autonomous driving solutions in enhancing safety or improving transportation in general. It also shows the importance of assessing the solutions broadly with fine-grained metrics as we have described in Section 1.1.2.

1.3. Research Goals

After the presentation of the context of this thesis, we can briefly introduce the research goals and the experimentally validated contributions that they include. The ultimate goal is to progress the field of autonomous driving and with it, in the adjacent fields of computer vision, artificial intelligence, and robotics. We frame these contributions in the field of AI-driven robotics with computer vision, inside the application domain of autonomous driving. The following research goals (RG) have been identified and addressed throughout the thesis, which contribute to the ultimate goal:

- **[RG1] Study the *state of the art* of autonomous driving systems, focused on end-to-end systems and related fields:** this goal entails conducting an in-depth literature review of the autonomous driving field with special interest in the end-to-end systems to gain insights about the current point of development and the questions that still need to be addressed to enhance the field. This objective necessitates a meticulous literature review within the autonomous driving field, with a keen emphasis on end-to-end systems, to gain nuanced insights into the current state of progress, emerging trends, and persisting research questions that warrant further exploration for the field's advancement. Concurrently, we analyze the state-of-the-art for traffic monitoring and object detection in computer vision, elucidating key advancements, challenges, and potential avenues for improvement.
- **[RG2] Validate computer vision in a driving setting generating a *traffic monitoring tool*:** After the review of the autonomous driving and object detection field through a state-of-the-art study, we need to explore the field, combining computer vision and artificial intelligence. We experiment with the safety-critical scenarios that need to be addressed in autonomous driving developing a system for visual monitoring of the traffic using state-of-the-art computer vision techniques. Leveraging state-of-the-art computer vision techniques, this endeavor serves as an initial step toward understanding the evolving needs of the field. This investigation poses a foundational research question, positioning our study as a gateway to further exploration and innovation within the domain.
- **[RG3] Generate an *object detection assessment software* to validate solutions:** It is critical to generate fine-grain metrics and understand them in-depth to produce research that is worth it for the community. In this case, we generate a software application for research experiments that includes functionality for validation of object detection solutions comparing them to a series of compatible datasets that are common in the research community. This development continues the introduction into the computer vision field, which is critical for autonomous driving, and also into the evaluation of computer vision systems.
- **[RG4] Generate an *autonomous driving behaviors assessment software* to conduct the experiments and generate quantitative data:** After studying the im-

plications of the assessment in the evaluation of a deep learning system and understanding the strong safety needs for driving, we need to develop an assessment system for autonomous driving systems. These systems should generate fine-grain metrics that help the research to compare different autonomous driving solutions in different tasks and settings. This evaluation is focused on simulation, thanks to the availability of high-detail simulators for autonomous driving like CARLA. The software should give support for different autonomous driving tasks like following the lane, driving with traffic, or navigating and respecting the traffic lights and signals between different target points.

- **[RG5] Generate end-to-end autonomous driving agents for visual lane following that enhance their behavior based on the addition of visual *memory* and *kinematic* input to visual deep learning imitation learning models:** as we have already described, there are typically two approaches for developing autonomous driving systems, modular and end-to-end. In this case, based on the literature review conducted, we identify the trend towards developing end-to-end systems in research in contrary to generating modular approaches due to its simplicity. Based on that the goal is to generate an end-to-end autonomous driving system that using one single camera as input can generate final control commands that drive the vehicles safely. In addition, the goal includes introducing other data types, the current speed of the vehicle, and understanding whether it helps the system understand further the environment and how to proceed. Also, it includes exploring deep learning architectures for this end-to-end system that include memory capabilities.

- **[RG6] Develop optimized end-to-end autonomous driving control visual lane following models that leverage the latest deep learning *optimization* techniques:** for this goal, we would like to explore the broad range of optimization techniques that compose the state-of-the-art in deep learning and apply them to autonomous driving systems. In this case, the research goal includes leveraging these techniques in the search for findings that can generate advancements in the field by producing systems with the same quality of behavior but in the most optimal way. This study should be extensive to different autonomous driving architectures or tasks while maintaining its simplicity.

- **[RG7] Enhance the control behavior of the autonomous driving vehicle for visual lane following *with traffic*:** in the realm of autonomous driving, there are a lot of possible tasks that a vehicle can address like following a lane, parking autonomously, overtaking a vehicle... for this particular goal, we focus on introducing traffic to a simulated autonomous driving scenario and we study how to enhance the system behavior in this situations. For this goal, the vehicle should be able to drive applying an end-to-end schema following the lane and also considering traffic. This traffic could be different types of vehicles with different shapes, colors... The driving should be safe enough to meet the strong standards needed in research and

industry, keeping a safe distance at all times.

1.4. Contributions

Following the description of several research objectives, we summarize the contributions stemming from the concepts outlined in the preceding section. They will be discussed in detail in Chapter 9, but this enumeration helps to have a global outline of the thesis.

1. **A systematic study of the state of the art in the autonomous driving field:** We reviewed around 200 papers focused on autonomous driving research and related fields. The related fields include traffic monitoring, object detection, autonomous driving systems and imitation learning, simulators, datasets, and assessment of solutions, and optimization in deep learning. This extended study has helped us to understand in-depth the current landscape of the field and to generate the rest of the research goals and contributions studied in this document.

2. **TrafficSensor, a tool for monitoring traffic in highway scenarios using computer vision:** We have developed a system for monitoring traffic in highway scenarios based on state-of-the-art computer vision techniques. In this study, we have studied the safety enhancement that computer vision can give in traffic management. This study represents and initial step towards the introduction of the autonomous driving ecosystem, comprehending the safety implications of the field and how it can be improved using these tools.

3. **Detection Metrics, an open source software for the assessment and comparison of object detection models and datasets that eases the research process:** in the route towards developing an end-to-end system for autonomous driving based on computer vision, we have developed Detection Metrics, a software tool for assessment of object detection models and datasets. In this case, we understand the implications of a proper evaluation in advancing deep learning research and autonomous driving research. Our study demonstrates its research value for improving the work of a researcher in comparing different object detection models fast and unattended, also serving as an assessment tool for the previous TrafficSensor contribution.

4. **Behavior Metrics, an assessment and comparison tool for different autonomous driving tasks that generate fine-grained quantitative data useful for research:** as part of the contributions of this thesis, we developed Behavior Metrics, a software tool for the online assessment of autonomous driving solutions through fine-grain metrics. After studying the monitoring of traffic and developing an assessment tool for object detection models, we understand the need for similar software for comparison of autonomous driving solutions for different tasks. These tasks include following the lane, driving with traffic, and navigating a simulated environment. Each task comes with its evaluation metrics that are specifically designed for them,

in addition to the common metrics that are general to all the tasks. Based on this premise, we have developed this tool that is a contribution to the thesis. It generated quantitative experimental data that a researcher can use for comparing different autonomous driving systems objectively. This tool is designed to ease the work of a researcher, providing tools for unattended experimental batch validation or for quantitative evaluation through a simulation view. Our study demonstrates its validity and research value with the following studies described in the following points.

5. **Empirical study of the improvement to the basic visual lane following with the addition of visual memory and kinematic input:** as part of the contributions of the thesis, we have explored the addition of current speed and visual memory to end-to-end autonomous driving system based on visual input. In this case, the target task is lane following in an urban scenario meeting all the safety standards needed. We explore whether or not introducing the current speed to the system as input and adding visual memory capabilities improves the final behavior of the system. We conduct extensive ablation studies and use the system in extreme scenarios to explore the actual advantages that this addition can introduce to the final system. Our study demonstrates that adding visual memory and current speed data is helpful in certain scenarios for an end-to-end control system even in a simple task of following the lane in a simulated urban scenario.

6. **Empirical study of model optimization for the controllers of the autonomous driving vehicles, generating several optimized models that improve the autonomous driving system on various fronts:** another contribution of this thesis is the study of deep learning optimization techniques for improving the autonomous driving deep learning models keeping the quality of their decisions. As part of the contribution, we have generated a series of optimized models for end-to-end lane following in a simulated urban scenario that are faster in inference time and smaller in size while maintaining their decision quality. Our study demonstrates the importance of optimizing deep learning models for autonomous driving to reduce inference time and model size while meeting high safety standards. This particular contribution is critical for edge devices where the hardware capabilities are more limited.

7. **Generation of an end-to-end shallow model capable of driving *in traffic* scenarios based on vision input:** another contribution of this thesis is the empirical study of including traffic in an urban scenario where the autonomous vehicle has to drive and including traffic information inside the model. We study and prove that simple end-to-end models can drive autonomously in a traffic situation maintaining a safe distance from other vehicles while also driving at a normal pace following the lane. For this task, we also study the evaluation metrics that can be helpful to understand and compare the system and include them in Behavior Metrics.

1.5. Structure of the document

Each contribution is thoroughly developed in its dedicated chapter. As they constitute distinct self-contained projects and papers aimed at the final goal of advancing the field of autonomous driving, certain sections may be duplicated or slightly overlapped across separate chapters.

The structure of the remainder of this thesis is outlined as follows:

1. **Chapter 2:** a detailed state-of-the-art description is conducted. This literature review is divided into different subfields relevant to the research.
2. **Chapter 3:** presents TrafficSensor, a system for monitoring traffic in highway scenarios using computer vision.
3. **Chapter 4:** provides a detailed description of Detection Metrics, a software tool for the assessment of object detection models and datasets.
4. **Chapter 5:** presents BehaviorMetrics, a software tool for the online assessment of autonomous driving solutions using fine-grained metrics.
5. **Chapter 6:** presents the study conducted to understand the enhancement of autonomous driving behavior using kinematic input and memory-based architectures.
6. **Chapter 7:** provides an in-depth description of the optimization of the end-to-end autonomous driving solutions for control.
7. **Chapter 8:** describes the study conducted for enhancing end-to-end control in situations where traffic is involved.
8. **Chapter 9:** presents the conclusions outlined from all the previous contributions and proposes some promising and solid future lines of work.

Chapter 2

State of the art

The autonomous driving field has suffered an impressive advancement in recent years, both in industry and academia. This PhD thesis builds upon the existing body of knowledge in autonomous driving research and also in the rest of the fields that affect this one, leveraging insights from related disciplines such as computer vision, artificial intelligence, and robotics (see Fig. 2.1).

Our review begins with an examination of literature concerning traffic monitoring and object detection, laying the groundwork for further exploration in autonomous driving. Subsequently, we investigate the intricacies of the autonomous driving domain, exploring topics such as simulators and solution assessment. Additionally, we explore optimization techniques for deep learning and other optimization methodologies, which form a crucial component of our contributions. To provide a comprehensive overview, this chapter is structured into six main sections, each focusing on a distinct aspect of the state-of-the-art in autonomous driving research and traffic monitoring.

2.1. Traffic monitoring

One classic problem within computer vision research is traffic monitoring [27] [28] [29] [30] [31]. This entails the tracking and classification of vehicles, for example on highways, to monitor them. Numerous studies have been dedicated to vehicle classification, employing vision-based techniques [32] [33] [34] [35] [36] [37]. Before the advent of deep learning, traffic surveillance using video cameras was significantly limited, primarily focusing on rudimentary tasks such as passive monitoring or basic automatic processing [38] [39] [40]. However, the emergence of deep learning has led to significant advancements in this field, notably in handling occlusions [41] [42] [43].

The classification of vehicles is intricately linked to monitoring traffic [44] [45] [46] [47] [48] [49] [50]. Traditionally, background subtraction has been a commonly employed technique for vehicle detection [51] [52] [53] [54] [55] [56] [57][58] [59]. Additionally, monitoring involves tracking vehicles, with many solutions relying on feature-based ap-

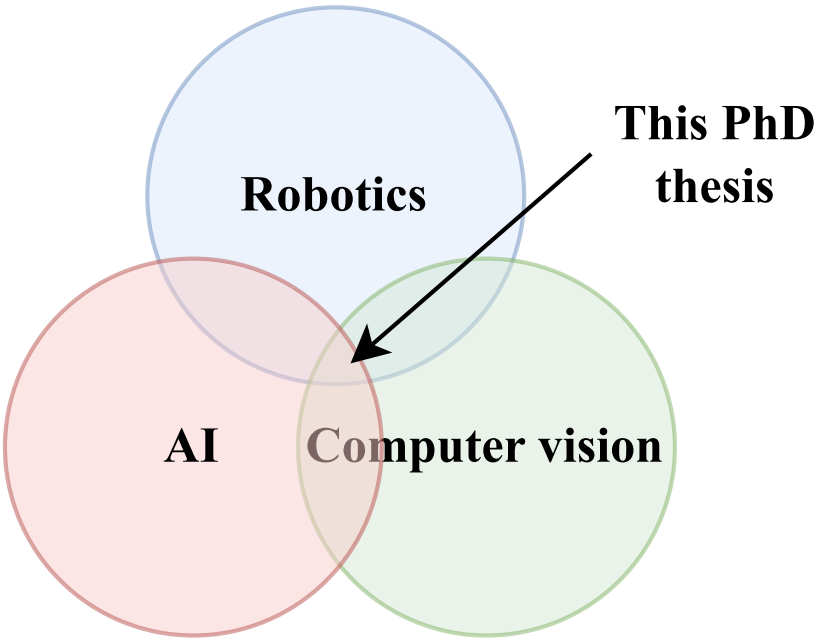


Figure 2.1: Venn diagram of the fields that are the core of this thesis.

proaches [60] [61] [62] [63] [64] [65]. The Scale Invariant Feature Transform (SIFT) [66] technique is often utilized to extract and utilize these features in numerous works dedicated to traffic monitoring [60] [67]. Alternatively, some approaches utilize 2D/3D models for this purposes [68] [69] [70]. Moreover, Kalman filters are employed in certain studies within the literature for vehicle tracking [71] [72]. Oriented FAST and Rotated BRIEF (ORB) [73] is an efficient and viable alternative to SIFT and SURF [74].

As our objective encompasses both classification and localization of objects, the utilization of object detection techniques becomes imperative [75] [76]. Numerous studies within the realm of traffic monitoring have centered around object detection, predominantly leveraging deep learning methodologies [77] [78] [79] [80] [81] [82] [83] [84] [85] [86].

2.2. Deep learning object detection, datasets and assessment

The field of object detection has experienced significant popularity in recent years, owing to notable advancements evident in the proliferation of high-quality literature reviews on the subject [87] [88] [89] [90] [91] [92] [93] [94]. The extensive adoption of object detection in real-world applications underscores its significance. Its utilization spans various domains, including autonomous driving systems [95] requiring a comprehensive understanding of surrounding objects, as well as applications such as in-camera filters and effects, particularly those reliant on recognizing specific objects like faces [96]. Moreover, object detection finds practical utility in scenarios within the retail sector, facilitating tasks such as inventory management and customer analytics [97].

Before the emergence of deep learning techniques, object detection relied on traditional approaches, primarily utilizing image processing methods and specific object features to identify objects within images [98]. In this conventional approach, the object detection pipeline typically involved several stages. Initially, features such as edges and textures were extracted from the image. Subsequently, a classifier, often based on algorithms like Support Vector Machines or Random Forests, was trained using these extracted features. Following training, a sliding window technique was commonly employed, where the classifier would analyze different portions of the image to predict the presence of objects. Finally, post-processing techniques were applied to refine and consolidate the detected objects, generating the final set of detections.

Analogous to the advancements witnessed in autonomous driving research, the progress in object detection owes much to the availability of high-quality open datasets. Notable examples include COCO [99], ImageNet [100], Pascal VOC [101], Princeton [102], Spinello [103] and Open Images Dataset [104]. In addition to these widely used datasets, there exists a plethora of specialized datasets tailored to specific subfields, such as small object detection [105] [106], human detection in crowded scenarios [107], or traffic sign detection [108] [109]. These datasets are typically curated using annotation software [110] during the construction process. Within the realm of object detection, few-shot learning has emerged as a topic of significant interest among researchers, demonstrating notable advancements [111] [112].

In addition to object detection, computer vision encompasses other key areas such as image classification [113] [114] and image segmentation [115] [116] [117] [118]. In autonomous driving, object detection and image segmentation are used for the perception part of the system, while object detection locates and classifies objects in the image, and image segmentation classifies each pixel (see Fig. 2.2 for an example).

Each of these tasks includes a specific set of evaluation metrics used to measure the effectiveness of a model and for a fair comparison of several approaches. These include Precision, Recall, Accuracy, Intersection over Union (IoU), F1-score, mean average precision (mAP), or mean average recall (mAR), among others. Precision, Recall, and Accuracy are based on the number of true positives (TP), true negatives(TN), false positives (FP), and false negatives (FN). A review of their equations is provided below:

Precision:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.1}$$

Recall:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.2}$$

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

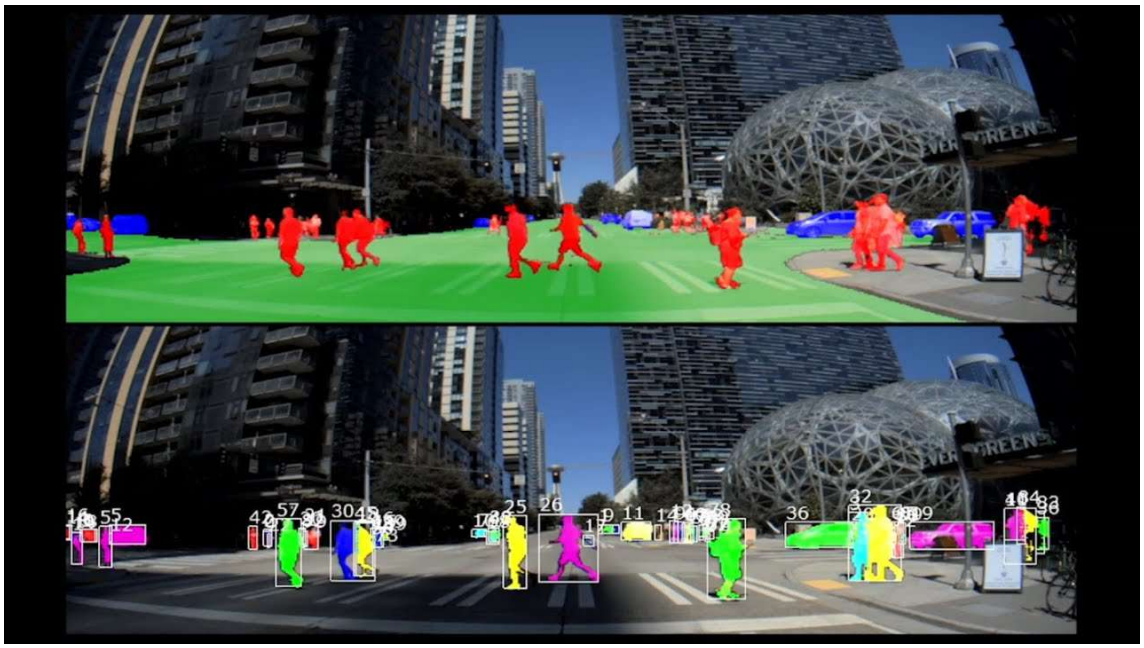


Figure 2.2: Object detection and image segmentation. Source: https://blogs.nvidia.com/blog/drive-labs-panoptic-segmentation/.

IoU (Intersection over Union):

$$\text{IoU} = \frac{\text{Intersection area}}{\text{Union area}} \tag{2.4}$$

F1-Score:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

Mean Average Precision (mAP):

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{2.6}$$

Mean Average Recall (mAR):

$$\text{mAR} = \frac{1}{N} \sum_{i=1}^{N} \text{AR}_i \tag{2.7}$$

Mean Absolute Error(MAE):

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.8}$$

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.9}$$

All of the current state-of-the-art solutions inside computer vision heavily rely on deep learning methodologies. The development process of a deep learning model involves several steps, with one crucial step being the selection of a deep learning framework [119]. In recent years, popular frameworks such as TensorFlow [120], Caffe [121], Darknet [122], Keras [123] or PyTorch [124] have been commonly employed for this purpose. The use of deep learning has been indispensable in this PhD thesis using specially PyTorch and Tensorflow frameworks in the majority of the research contributions.

Focusing on object detection, another critical aspect involves selecting the appropriate architectural technique. These techniques are usually divided into two groups, two-stage or one-stage. The first group includes multiple stages from the input to the generation of the final bounding box candidates while the second group directly generates the detection and location in a single forward pass. Noteworthy advancements in recent years have introduced architectures such as Faster Regional-CNN [125] (two-stage), Single Shot MultiBox Detector [126] [127] [128] (one-stage), and You Only Look Once (YOLO) [129] (one-stage). This last one, YOLO, has experimented with a huge development and several new versions have been released (YOLO9000 [130], YOLOv3 [131], YOLOv4 [132], YOLOv5 [133], YOLOv9 [134]...). More recent developments have shown interesting results using Tranformers-based architectures, showing a promising direction with DETR [135], Swin Transformer [136] or DINO [137], among others [138], and indicating a potential direction for future research. All the ideas detailed here are useful in the perception of autonomous driving agents and traffic monitoring, as we describe in the contributions.

2.3. Autonomous driving and imitation learning

The field of autonomous driving combines artificial intelligence, robotics, and computer vision, as we have already mentioned in Chapter 1 (see Fig. 2.1). In that chapter, we have already introduced the main reasons for the latest advancements. We have also discussed the main components of an autonomous vehicle, including the actuators, sensors, and processors [139]. We have presented some of the tasks that autonomous driving addresses, like obstacle avoidance [140], following the lane, or driving in traffic [141]. The solutions for generating these systems typically fall into two groups, end-to-end or modular approaches, as we have described in the previous chapter (see Fig. 1.5 for a detail).

The modular approach [142] [143] [25] combines several specialized modules, each specific to particular activities of the driving procedure. They communicate among them and their combination and that combined and communicated between them drives the vehicle. The driving tasks are divided into several submodules: perception, planning, mapping, control... [142] [143] [144] so the modular approach can implement a solution for each of them that communicates and final are aggregated into the final solution. This is the most widespread approach for solving autonomous driving applications, in part due to its flexibility. One of the most used criticisms of the modular approach is that the errors

in some modules are difficult to track and that they can lead to cascading errors in the rest of the system. Conversely, the end-to-end approach receives criticism for its lack of interpretability.

One of the most relevant open-source examples of a modular approach for autonomous driving in active development is Autoware [145]. It is built using ROS and includes modules for each autonomous driving activity like perception, planning, control... This project supports more than 30 different types of vehicles. We may find other examples like Stanford's Junior [143] or Boss [142], which won the DARPA Urban Challenge with a GPS, lasers, radars, and cameras as sensors.

In contrary, the end-to-end approaches [146] [147] [148] [149] [150] [151] directly translate the input raw data provided by the sensors in the vehicle to final control commands that drive the vehicle. Many developments of this idea have been published. For example, [146] proposes an end-to-end model based on a CNN architecture, [152] proposes this approach orienting its development towards planning, [147] an architecture to explain the decisions of an end-to-end architecture, or [153] in which they predict the HD map to improve the driving capabilities.

Vision-based end-to-end autonomous driving has gained significant attraction in recent literature but still presents some limitations [154]. Despite this, recent strides indicate significant advancements. For example [26] proposes a combination with large language models for driving. This vision-based approach is an integral segment of the accumulation of exteroceptive information about the surrounding environment of the vehicle [155] [156]. It has been explored with Bayesian approaches [157], used for explaining the decision [147], or used in real vehicles [158].

PilotNet [159] is a shallow deep learning model for vision-based end-to-end control proposed by researchers at Nvidia. This solution employs a convolutional neural network (CNN) [160] to extract features and generate control commands from input images. This solution was further analyzed by augmenting fully connected layers to a CNN [161] to reach the point of performance comparable to a human driver. Simulated images have also been used [162] to study the significance of road-related features in the images.

Building upon this shallow model, numerous advancements have been released, enhancing both complexity and functionality. For instance, TCP [163] integrates a single camera as input data, deviating from other methods that rely on a combination of sensor data but still achieving great results in the CARLA Leaderboard. Another example is ReasonNet [164], which combines temporal and global reasoning to facilitate informed decision-making. Other recent examples of the end-to-end approach are NEAT [165], or Transfuser [166], among others [167].

2.3.1. Imitation learning and reinforcement learning for driving autonomously

Imitation learning [168] consists of generating a driving policy learning from expert agent-gathered data (see Fig. 2.3). One relevant technique inside imitation learning, utilized in this thesis, is called behavior cloning [169] [170]. In this technique, applying it to autonomous driving, one or more expert agents drive while the data produced by their actions are collected. This means that the input data from the sensors are collected and the actions (controls) are also saved. With these data, a deep learning model is trained to learn from the expert agents' behavior and generate a model that can imitate their behavior. Some successful uses of this technique have already been generated for autonomous driving, like in [171] where they combine a successful behavior cloning policy with hard constraint handling for those scenarios that are not found in the collected data that lead to catastrophic safety-critical failures.

In a different example [172], the researchers include high-level input control commands in the system because the baseline architecture does not allow control at testing time. Introducing these commands, they generate a chauffeur-like control at test time. Another study [165] presents NEAT, where they generate a representation of the semantic, spatial, and temporal structure of the diving scene using imitation learning. In [166], the researchers use this technique combining it with Transformers and generating a mechanism to integrate image and LIDAR representations into a global context that leads to a safe driving behavior. In a separate contribution [167], they proposed a system to train driving policies not only from a single expert agent but from the rest of the agents that it observes. The previously presented examples of TCP [163] and ReasonNet [164] also use imitation learning.

This technique is also applicable to other domains outside autonomous driving [173], for example for gaming agents where the model is trained from data from the expert agent in the same way. Generating a successful model necessitates a diverse range of data points. However, the majority of cases recorded during normal driving experiences may lack the relevance required, leading to an imbalanced dataset. Since the majority of cases that we will be recording are of normal situations, the performance of the vehicle in complicated situations or never-seen situations would deteriorate rapidly (see Fig. 2.4 for an example). To mitigate this issue, researchers have proposed solutions like using the learned policy for querying the expert agent in difficult situations and aggregating these extreme cases to the dataset, a strategy known as DAgger [174] or exposing the learned policy to perturbations to generate valuable data [175]. Other possible approaches include oversampling of extreme cases for balancing the dataset and using data augmentation techniques, which are common in machine learning problems and are supported by the deep learning frameworks, with dedicated packages like Albumentations [176].

Other techniques for driving autonomously use reinforcement learning (RL), which is also capable of managing complex environments. In this area, the vehicle interacts with the environment through exploration-exploitation and uses a reward function to learn

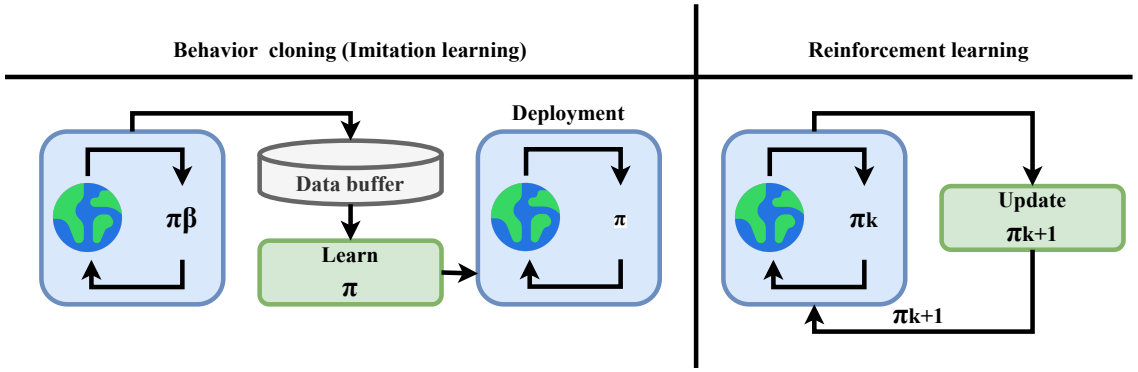


Figure 2.3: Diagram of behavior cloning (imitation learning) and reinforcement learning. Adapted from [26].

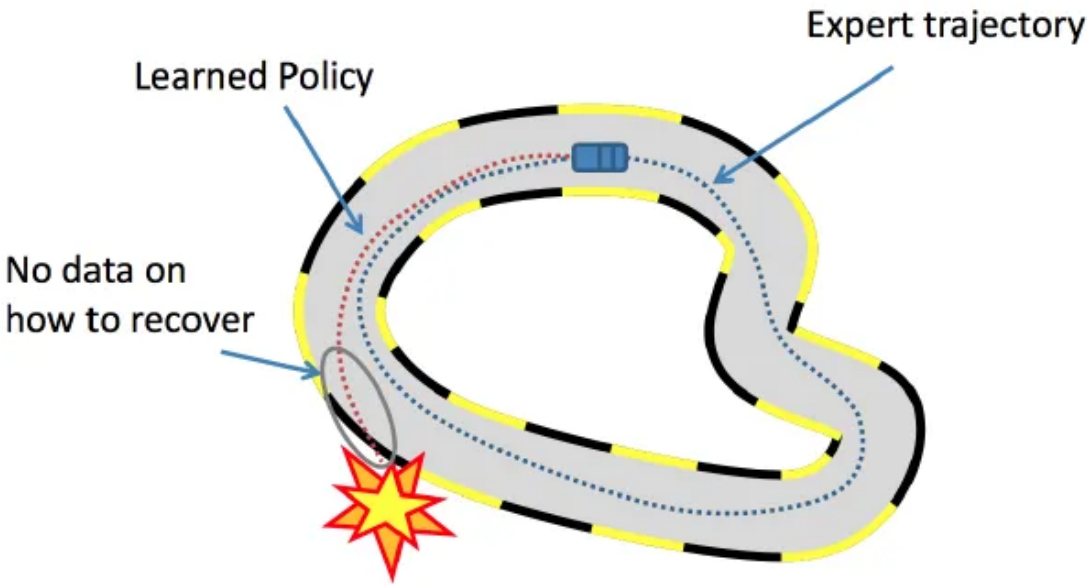


Figure 2.4: Learned policy has issues when it encounters a new situation.

a policy [177]. These techniques include Q-learning [178], are also end-to-end [148], have interesting results in racing [149], and can be deployed in real vehicles [179]. The combination of reinforcement learning and deep learning, deep reinforcement learning (DRL), has also been explored [180] and even the combination of imitation learning and reinforcement learning shows a promising direction [181].

2.4. Simulation in autonomous driving and assessment. Datasets

In the development and research of robotics systems, simulators are commonly used to generate solutions rather than directly developing in real robots or vehicles. The simulators facilitate easier and more cost-effective iterative development, testing, and debugging of solutions, validating the research hypothesis. They give access to ground truth data from sensors and actuators, easing the validation of solutions. Using simulators, re-

Figure 2.5: CARLA and Gazebo simulators.

searchers can generate synthetic datasets that include ground truth data at a cost-effective rate. They also allow the assessment of solutions.

Regarding autonomous driving, a diverse array of simulators are available, playing a pivotal role in this research field [164] [163]. SUMO [182] is an open-source simulator specialized in traffic that pursues simulating efficiency and management strategies. TORCS [183] is a racing simulator. Gazebo [184] is an open-source general-purpose robotics simulator that has also been employed for autonomous driving research [185]. CARLA [186] is an open-source simulator, specialized in realistic simulation of urban driving and is widely used. It is run using Unreal, a real powerful game engine. It is easily customizable, offering a wide selection of common day-to-day vehicles, sensors, and maps for validating autonomous driving solutions. The system features a rule-based expert agent designed for autonomous vehicle navigation, which can be leveraged to generate synthetic datasets. An alternative to this expert agent, named Roach, has been developed using reinforcement learning techniques [187].

DeepDrive [188], Baidu Apollo [189], Autoware [145], AirSim [190] or Udacity's Self-Driving Car Simulator [191] are other possible options, with their strengths and limitations. Even video games have been used as simulators for the development of autonomous driving solutions [192] [193].

Certain simulators, including Gazebo and CARLA (see Fig. 2.5), are compatible with ROS [194] [195], which serves as the global standard for robotics middleware. ROS, being open-source streamlines the development of applications and facilitates a seamless transition from simulation to real-world robots. This integration enables researchers and developers to leverage the robust capabilities of ROS for controlling and interfacing with simulated environments, thereby accelerating the development and testing of robotics solutions.

2.4.1. Datasets

Besides simulators, datasets play a crucial role in the development of autonomous driving systems. In this field, there are a diverse range of publicly available datasets that support the training of deep learning models for various tasks, as previously mentioned, significantly contributing to the field's recent advancements. For visual perception, we have nuScenes [196], BDD100K [197], KITTI [198] [199] or Cityscapes [200], for planning nuPlan [201] or for lane following (lane keeping) commaAI [158] or Udacity datasets [202]. These datasets, along with others [203] [204], have become benchmarks for researchers and developers in the field.

2.4.2. Assessment

Given the wide range of traffic scenarios and driving tasks that are part of autonomous driving and the security standards needed [205], robust evaluation metrics are imperative. While common deep learning metrics like mean squared error (MSE) or accuracy are suitable for validating static supervised data, they may prove inadequate for assessing the performance of a fully autonomous driving system or task-specific systems. These offline metrics fail to capture the dynamic nature of driving tasks over time intervals, such as lane following, obstacle avoidance, intersection traversal, and automated parking.

Hence, there is a need for a complementary evaluation framework that considers the temporal aspects of driving tasks. For instance, a failure in a single control iteration of the autonomous vehicle could potentially lead to a collision later, despite only manifesting in one frame. Therefore, evaluating system performance requires metrics that account for the system's behavior over time and its ability to handle various dynamic scenarios effectively.

This question has already been addressed in the literature and a vast range of metrics and evaluation strategies can be found for different parts of the autonomous driving systems and different situations [206] [207] [208] [209].

CARLA simulator already generates metrics that can be used for validating solutions (Driving Score, Infraction penalty...). CARLA Autonomous Driving Leaderboard [9] is an assessment framework and challenge built on top of the simulator for evaluating and ranking solutions for autonomous driving in urban scenarios and routes. This framework is designed for broadly testing and validating fully autonomous driving solutions in a variety of traffic scenarios simultaneously, so it can be overly challenging or even unsuitable when considering developing solutions for specific driving tasks or researchers starting in autonomous driving. Furthermore, it presents limitations for assessing the autonomous driving tasks supported by Behavior Metrics (described in Chapter 5) and lacks the level of detail in the metrics. Therefore, alternative approaches for assessing autonomous driving solutions may be necessary [210].

[9] https://leaderboard.carla.org/

2.5. Memory-based approaches in end-to-end visual autonomous driving

We have outlined the primary approaches for developing autonomous driving systems, encompassing typical inputs and outputs. An intriguing avenue of research within the end-to-end approach involves integrating memory capabilities. This memory can manifest within the deep learning architecture through dedicated memory modules or directly within the input data. For instance, [211] proposed the addition of temporal analysis using memory-based deep learning models, examining the importance of LSTMs and convolutional layers with LSTMs respectively. Memory-based solutions have been explored in previous research publications as well. For example in [212], researchers used a combination of CNN and LSTM layers, and in [213] an [214], they used similar approaches combining ConvLSTM modules for extracting the temporal information of the data input for generating the control commands of the vehicle. Long Short-Term Memory (LSTMs) networks [215] are a special type of recurrent neural networks (RNN), that are specialized in learning temporal dependencies that are present in the dataset. A variation of the LSTMs is Convolutional LSTMs (ConvLSTMs) [216], which incorporate convolutions to learn temporal dependencies from sequences of images. Additionally, Conv3D convolutions are commonly employed architectural layers when learning temporal dependencies is crucial, like in video classification.

Regarding enhancements to incorporate memory into the system through input modifications, notable techniques include juxtaposing images [13] or amalgamating multiple images from different time steps into a composite image with increased channels.

2.6. Optimization of deep learning models for autonomous driving

The vision-based solutions are usually generated using deep learning models, which are high-demanding computational solutions. An important component in this scenario is the available computing hardware as the performance of robot applications depends not only on the quality of the model decisions but also on their frequency. Some autonomous vehicles or robots are equipped with high-performance hardware while others are not. An option in this case is to update to faster-computing hardware, but this is not always feasible. A possible solution is to optimize the deep learning model with different techniques [217].

There are an extensive number of optimization techniques, including quantization of the computations [218] [219], pruning of some parts of the model [220], fine-tuning (retraining) with optimization aware techniques, or clustering some components (see Fig. 2.6 for a detailed diagram). These techniques are usually included in the most common deep learning development frameworks such as PyTorch or TensorFlow. For instance [120] [221] [124] used them for building their models and some of them have been also further optimized for certain hardware devices, such TensorRT for Nvidia GPUs [222]. These techniques are usually combined in the development of an optimized efficient deep

learning model, always considering the trade-off between the optimization percentage and quality of the model outputs.

Some studies have addressed the optimization of autonomous driving systems [223], but they have not considered deep learning models as potential controllers. In this thesis, however, we focus on utilizing deep learning models as controllers and we explore this optimization, filling a gap in the existing research literature.

This comprehensive review of the state-of-the-art across various fields central to the thesis serves as an introduction, providing essential context for the ideas discussed in the subsequent chapters. In the following chapters, we will examine the contributions made for the thesis, commencing with the examination of traffic monitoring using computer vision.

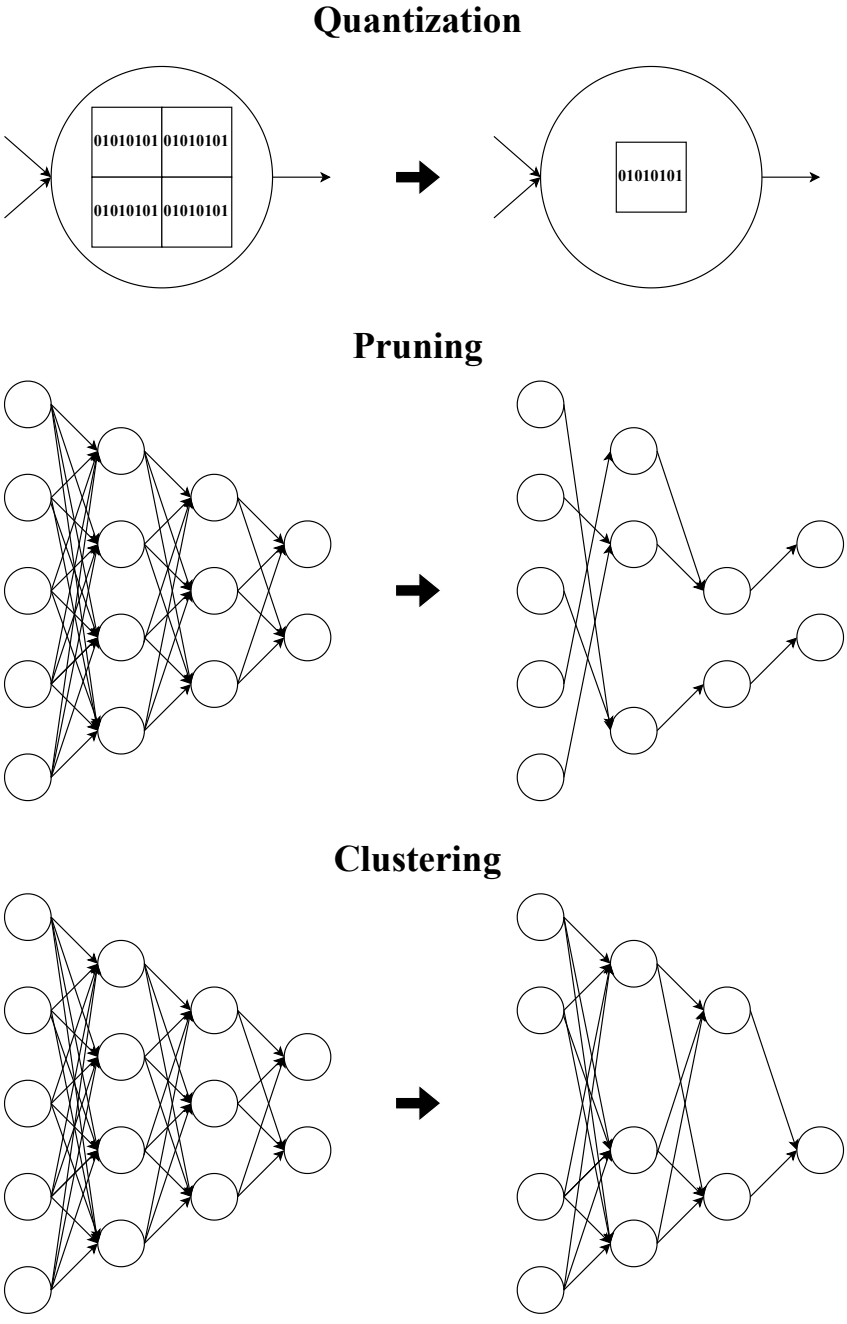


Figure 2.6: Optimization techniques diagrams.

Chapter 3

Monitoring and assessing traffic with deep learning

Monitoring of real-time traffic on highways, roads, and streets may provide useful data both for infrastructure planning and for traffic management in general. This area of research has been already addressed in computer vision, as we present in Chapter 2. This area has experienced a rise in recent years, particularly thanks to the advancements in deep learning for object detection and classification. In this chapter, we present TraficSensor, which is one of the contributions of this thesis. It is an open-source system that employs deep learning techniques for automatic vehicle tracking and classification on highways using a calibrated and fixed camera. For this purpose, a new traffic image dataset was created to train the models, which includes real traffic images in poor lighting or weather conditions and low-resolution images. The proposed system consists mainly of two modules, one responsible for vehicle detection and classification and the other for vehicle tracking. After a test and comparison phase of different object detection approaches trained on the new traffic dataset, YOLOv3 and YOLOv4 were selected. The second module combines a simple spatial association algorithm with a more sophisticated KLT (Kanade-Lucas-Tomasi) tracker to follow the vehicles on the road. Several experiments have been conducted using challenging traffic videos to validate the system with real data. The experimental results demonstrate that the proposed system successfully detects, tracks, and classifies vehicles in real-time on a highway. This paper has been published in a journal [224] and in this chapter we describe it. The open-source software described in available online for replicability and extension [225].

3.1. Introduction

The rise of vehicles in circulation raises several challenges of environmental, economic, and infrastructure management types. This situation creates the need for reliable monitoring techniques. The intelligent transportation systems (ITSs) goal is to monitor the

different vehicle transport networks smartly, using dedicated sensors and advanced video cameras. This data may be used for coordination of the traffic networks, minimizing congestion, and enhancing mobility. Video cameras are usually used on ITSs systems due to their simple installation and maintenance combined with their rich nature of information. This makes them one of the best solutions when it comes to surveillance and monitoring. Depending on the conditions of the ITS system, it will be necessary to use moving cameras instead of fixed ones. Additionally, this system made use of other sensor types like radars for speed enforcement or inductive loops and laser and infrared sensors for vehicle classification, as discussed in Chapter 2. The systems that use such technology try to classify the vehicles by extracting certain information such as the vehicle's length and distance between axles. They have some drawbacks, like an intrusive installation and the fact that not all of them provide the possibility of multilane monitoring. Additionally, the cost of installation is usually high and the data extracted from these systems is basic and cannot be used to extract high-level traffic data such as vehicle orientation, position...

Previously, the utilization of video cameras in traffic surveillance was primarily confined to passive monitoring tasks or rudimentary automated processing. However, significant strides have been made in this domain, chiefly attributed to advancements in deep learning techniques. Presently, sophisticated systems capable of detecting vehicles in both routine and challenging scenarios have become feasible, even in situations characterized by high vehicle density. This advancement serves as the foundation for executing high-level tasks such as automated traffic management, automatic incident detection, law enforcement, monitoring of weather conditions, and handling various other incidents. Building upon these advancements, we have developed a vision-based traffic monitoring system named TrafficSensor. It incorporates a robust vehicle detection and classification algorithm, along with a novel technique for addressing occlusions [41] [42] [43]. This system marks the evolution from a previous iteration [98] [226], achieving higher reliability and performance even in challenging lighting or weather conditions and with poor camera resolutions, all while maintaining real-time operation. By employing a fixed camera, the system effectively detects and monitors vehicles. This work, a significant contribution to the thesis, signifies our entry into the field of autonomous driving. In the subsequent sections, we detail the system's creation, conduct experimental validation, and discuss the results.

3.2. TrafficSensor: a deep learning-based traffic monitoring tool

TrafficSensor is a software tool designed for real-time traffic monitoring, capable of classifying vehicles into seven categories: motorcycles, cars, vans, buses, trucks, small trucks, and tank trucks. It comprises three main blocks: vehicle detection, vehicle classification, and vehicle tracking, as depicted in Fig. 3.1. These blocks are implemented as two separate modules, as detections and their classification are performed jointly due to the utilization of deep learning techniques. The tracking mechanism prioritizes spatial prox-

imity, resorting to the KLT algorithm if spatial proximity tracking fails. All detected blobs are continuously tracked over time.

In TrafficSensor, detection, classification, and tracking are conducted within a designated image area referred to as the evaluation area. This region, indicated by the user in the image, specifies the area on the road where detections should be focused, as illustrated in Fig. 3.2. While TrafficSensor is primarily designed to monitor outgoing traffic flow, its functionality can be extrapolated to handle incoming traffic flow as well.

3.2.1. Deep learning-based detection and classification

The system receives input images captured from the monitored video stream. These images are then fed into the deep learning model, which detects and classifies various vehicles present. Information is continuously stored at each moment, allowing for effective tracking based on data recorded from the previous instance. TrafficSensor is compatible with deep learning models trained using various frameworks such as TensorFlow, Darknet, and Keras, enabling it to accurately detect and classify different vehicles appearing in the images.

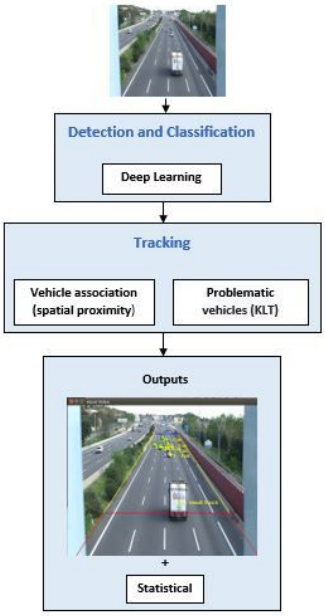


Figure 3.1: Block diagram of *TrafficSensor* system.

In the detection and classification block, the system implements the following criteria:

- Within the evaluation area, two zones are defined (Fig. 3.3). Zone 1 corresponds to the half of the evaluation area where vehicles enter. Detection and classification are generally more straightforward in this zone due to better visibility of vehicles. Zone 2 refers to the half through which vehicles exit the evaluation area. This zone poses greater complexity as vehicles tend to be smaller compared to those in Zone 1.

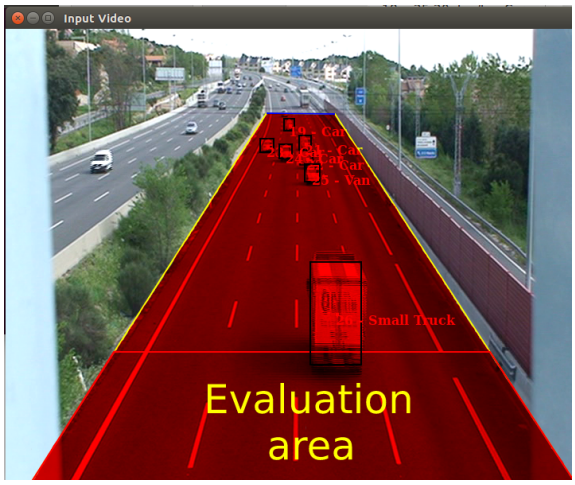


Figure 3.2: Evaluation area.

- Vehicles always enter the evaluation area through Zone 1. They cannot suddenly appear in the middle of the road. Thus, no new vehicle can be detected in Zone 2.
- If a vehicle is not detected in Zone 1 during a five-frame sequence, it will be considered a false positive and subsequently discarded.
- Any vehicle within Zone 2 is considered valid. If a vehicle is not detected using deep learning, the KLT algorithm will be employed for localization.

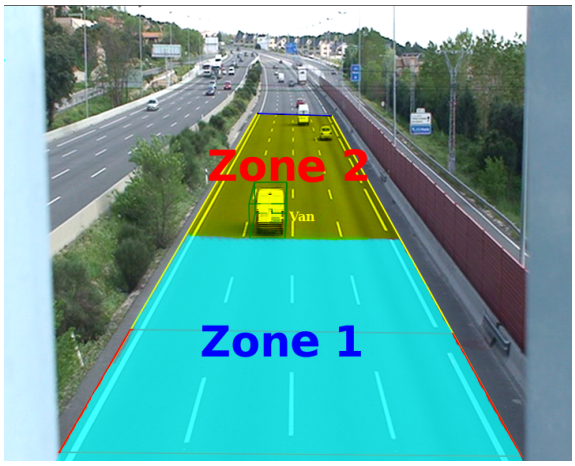


Figure 3.3: Evaluation zones.

Three different frameworks (TensorFlow, Keras, and Darknet) and four deep learning models have been tested to evaluate the optimzal configuration for the final TrafficSensor application. Specifically, the SSD MobileNetV2 network with TensorFlow, the SSD VGG-16 network with Keras, and YOLOv3 and YOLOV4 with Darknet.

- **SSD MobileNetV2 network:** the SSD MobileNetV2 network (Fig. 3.4) utilized in this study was trained using the COCO dataset. To use this network, we used the

configuration file *ssd_mobilenet_v2_coco.config*. This network architecture combines the Single Shot Multibox Detector (SSD) with MobileNetV2. MobileNetV2 extracts feature maps that are then utilized for classification and detection tasks in subsequent layers. The SSD aspect employs a convolutional feed-forward network to generate a set of bounding boxes at fixed sizes and scores the presence of object class instances within these bounding boxes. Finally, non-maximum suppression is applied to produce the final detections.

- **SSD VGG-16 network:** another SSD network has been used with VGG-16 as its base network, pretrained using ImageNet. Fig. 3.5 shows this network model. VGG-16 consists of 16 layers, of which 13 are convolutional, 2 are fully connected, and a softmax layer that is used to classify. Fig. 3.6 illustrates the architecture of the VGG-16 network.

- **YOLOv3:** You Only Look Once (YOLO) imposes strong spatial constraints on bounding box predictions. Each cell in the grid only predicts N bounding Boxes (N being a fixed parameter) and can only have one class. This spatial limitation restricts our model's ability to predict nearby objects effectively. YOLOv3 [131] (see Fig. 3.7 comprises a total of 107 layers, organized into two groups responsible for feature extraction and object detection

  - **Feature extraction (from layers 1 to 75):** it is the Darknet-53 network trained using ImageNet, comprising 53 convolutional layers (Fig. 3.8). This network has 416x416x3 images as input shape and features as output 3D 13x13x1024 and incorporates 23 residual layers. When a neural network increases in depth its precision tends to degrade in terms of propagating its characteristics, leading the greater losses in training. The residual layers mitigate this problem.
  - **Objects detection (from layers 76 to 107):** the model takes the 3D features (13x13x1024) as input and conducts object detection. A distinguishing feature of YOLOv3 is its ability to detect objects on three different scales, enhancing its effectiveness. It extracts features at three scales (13x13x39, 26x26x30, and 52x52x39), which are then processed by the final YOLO layer for object classification and bounding box regression.

- **YOLOv4:** this is the fourth iteration of the YOLO architecture that continues to enhance the previous versions with the latest advances introduced in the literature. It comprises three main components: backbone, neck, and head (Fig. 3.9). For the backbone, it uses CSPDarknet53 [227], for the neck SSP [228], and PAN [229] and YOLOv3 for the head. This network allows real-time object detection on a conventional GPU, thanks to its improvements in speed compared to other approaches and even its previous versions.

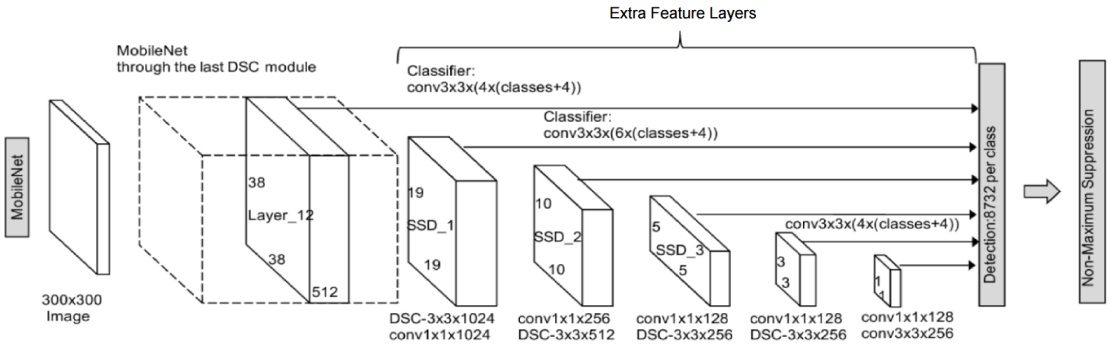


Figure 3.4: SSD MobilenetV2 network.

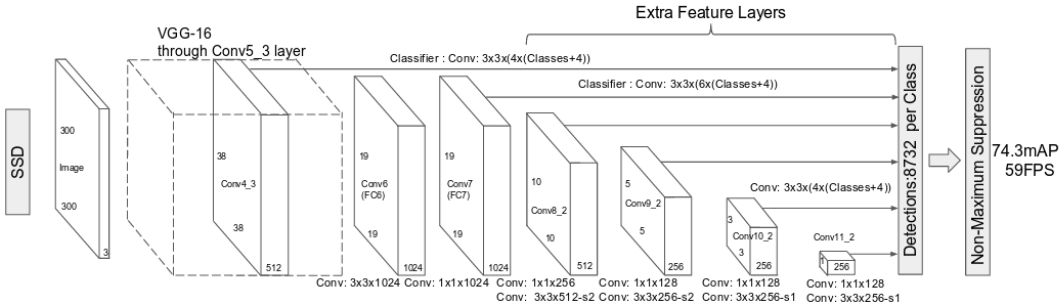


Figure 3.5: SSD network model.

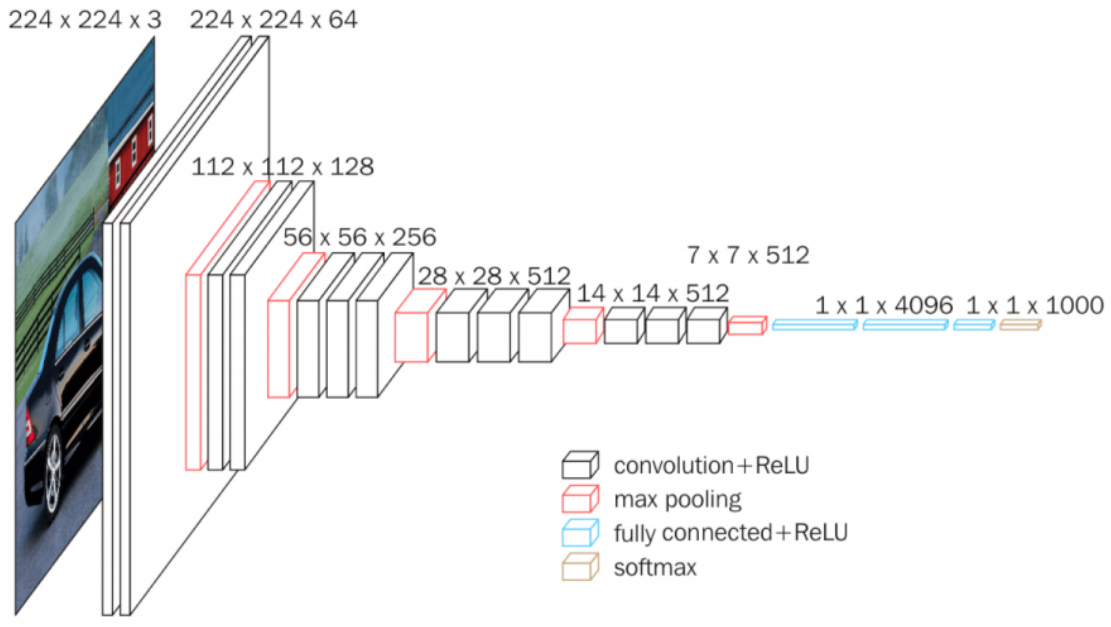


Figure 3.6: VGG-16 model.

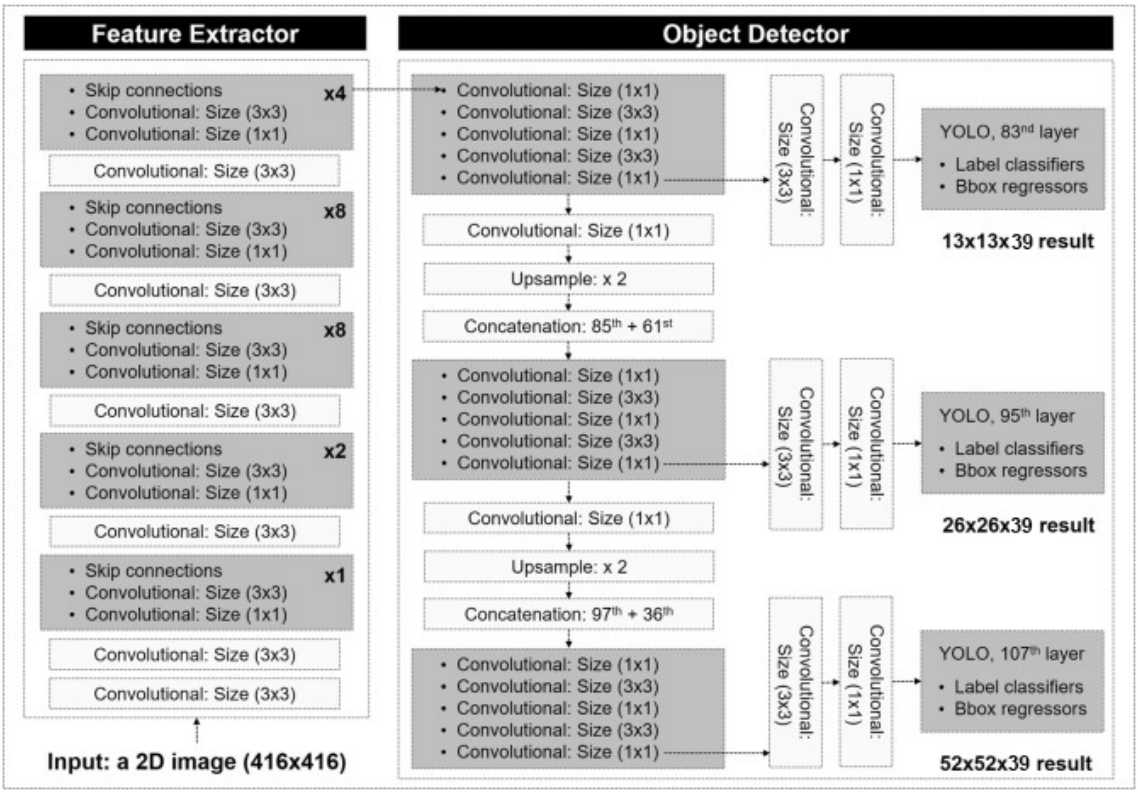


Figure 3.7: Yolov3 model.

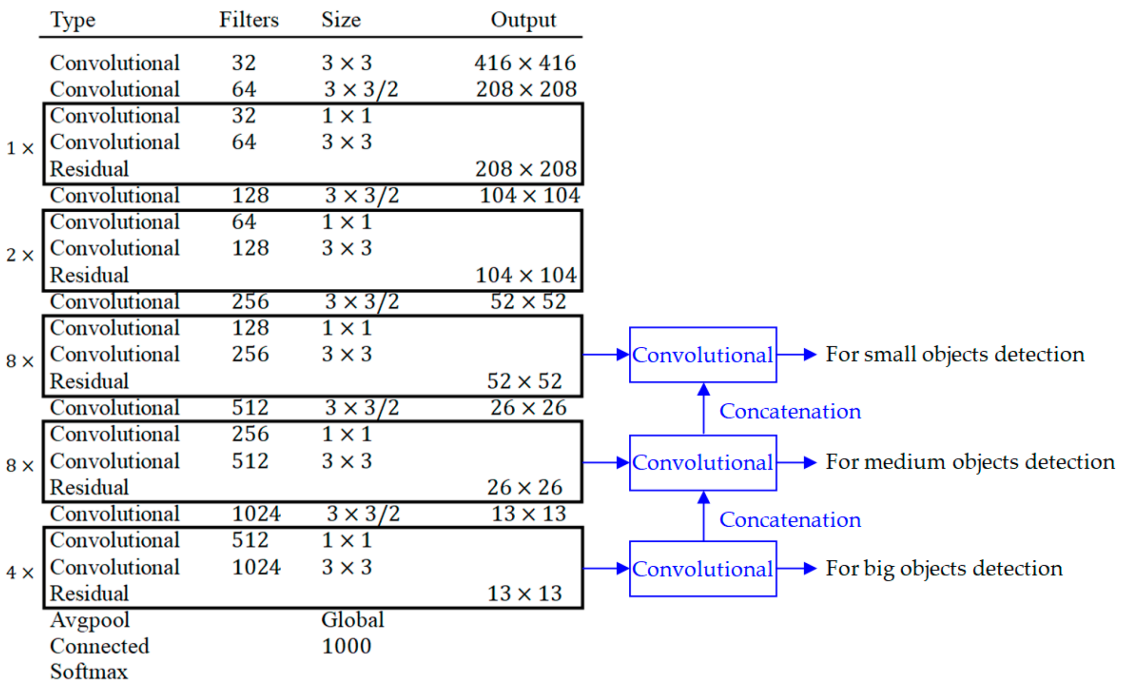


Figure 3.8: Darknet-53 model.

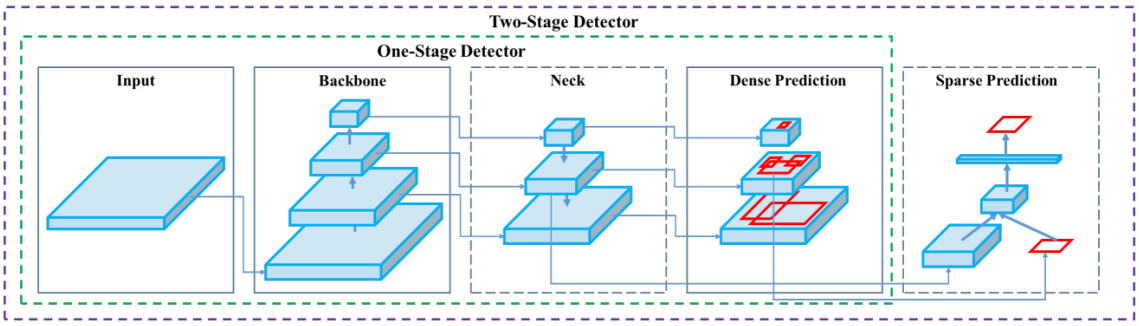


Figure 3.9: Yolov4 object detector.

3.2.2. Vehicle tracking

The execution flow of the tracking module is shown in Fig. 3.10, which shows the steps conducted when a new blob is detected inside the image, and in Fig. 3.11, which illustrates the procedure that is applied on already to previously registered vehicles. The tracking process emphasizes associating current detections with vehicles stored from the previous instant. Various considerations are taken into account:

- If a vehicle reaches the end of the evaluation area, it will be removed from tracking.
- Vehicles stored at instant (t-1) are analyzed to match them with vehicles detected at instant (t). This matching process pairs vehicles from (t) and (t-1) based on the minimum Euclidean distance between their centers.
- If the vehicle at instant (t) associated with the vehicle from instant (t-1) is not within the circular or elliptical area around the center of the vehicle from instant (t-1), it will not be matched.
- If spatial proximity fails to pair a vehicle from instant (t-1), the KLT algorithm will be employed.

Spatial proximity and the KLT algorithm are employed for vehicle tracking. While spatial proximity effectively distinguishes between vehicles, real-world videos often present challenges such as occlusions and detection difficulties for small or distant vehicles. In such cases, the feature-based tracking algorithm KLT is utilized, complementing the robustness of deep learning methods.

Spatial proximity tracking

Typically, the difference in pixels between a vehicle's position in instant (t-1) and instant (t) is minimal. Consequently, a vehicle at instant (t) will likely be located in a vicinity very close to its position at instant (t-1). When searching for a vehicle at instant (t), it is expected to be found within a small circular radius around its position at instant (t-1). Spatial proximity tracking in TrafficSensor is based on the method described in [98].

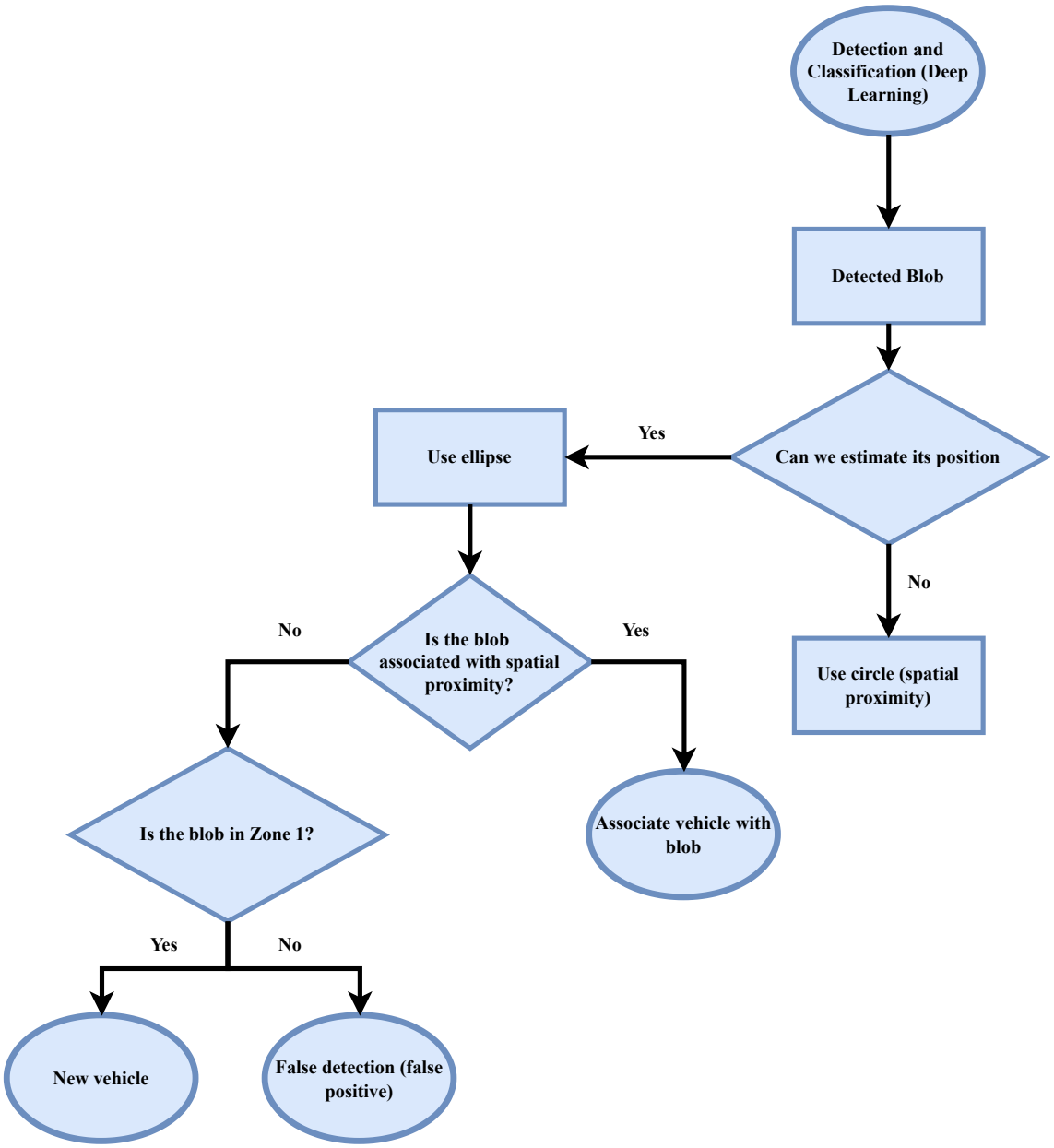


Figure 3.10: Execution flow chart of detected blobs.

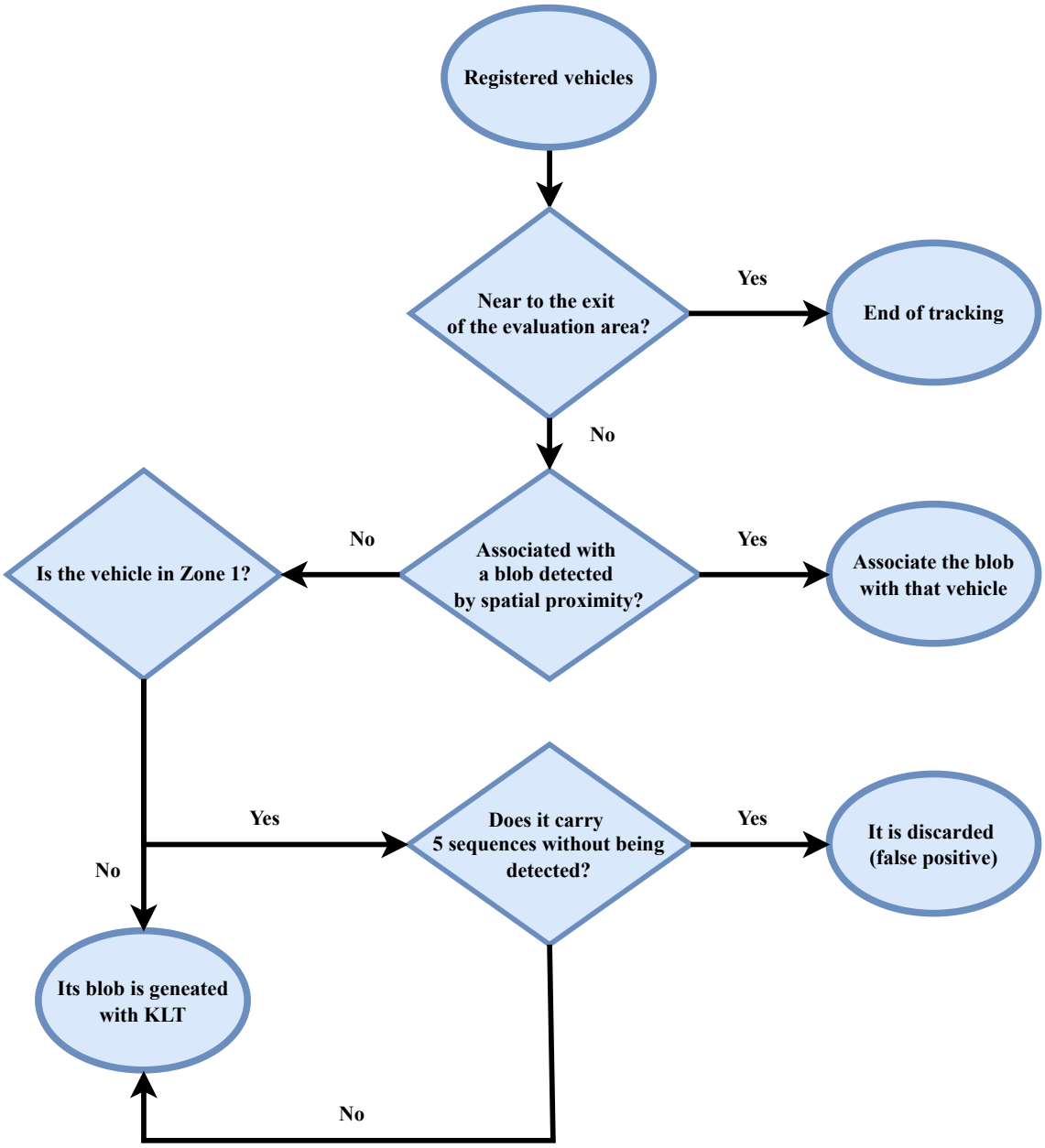


Figure 3.11: Flow chart of registered vehicles.

This method estimates the area where a vehicle should be located based on its position at instant (t-1). As vehicles progress along the road, this estimated area is continuously updated.

At first, the area is taken as a circle because the system does not have enough data about its orientation. But as the vehicle advances, the system has enough information to know its orientation, and so, it takes the areas as an ellipse whose center corresponds to the center of the vehicle in (t-1). It is considered that we have enough information to estimate its orientation when we have the position of the vehicle in 6 frames. Linear regression is used to calculate the orientation of the vehicle based on the position the vehicle will take as it progresses. Once we have information about the orientation, we will define the search area as an ellipse whose center is the same as the vehicle in t-q and whose direction is calculated with the following Equation 3.1:

$$\rho(r_i) = 2(\sqrt{1 + \frac{r^2}{2}} - 1) \tag{3.1}$$

The pairings between the vehicles detected at time (t) and the vehicles stored from time (t-1) are limited to vehicles that fall within the area of the circle of the ellipse that is obtained based on the position of the vehicle at time (t-1). The ellipses are defined as $C_{xc,yc,\omega}$, where ω is the orientation and (x_c, y_c) is the center of the vehicle. These parameters are illustrated in the Fig. 3.12.

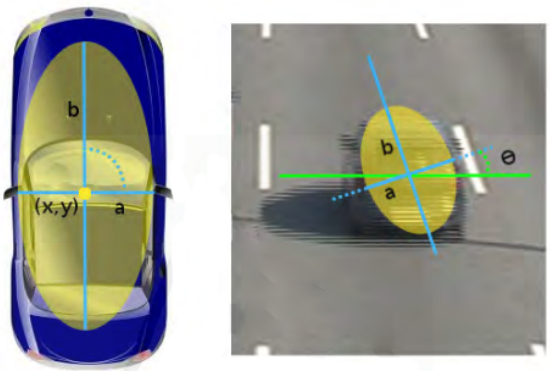


Figure 3.12: Vehicle associated 2D ellipse.

The 2D vehicle, whose center is $B(x, y)$, will be inside the ellipse $C_{xc,yc,\omega}$, if it accomplish the Equations 3.2 and 3.3:

$$C_\omega = \arctan(\frac{a_x}{a_y}) \tag{3.2}$$

$$\left(\frac{cos(C_\omega)(B_x - C_{x_c}) + sin(C_\omega)(B_y - C_{y_c})}{b}\right)^2 + \left(\frac{cos(C_\omega)(B_y - C_{y_c}) - sin(C_\omega)(B_x - C_{x_c})}{a}\right)^2 \leq 1 \tag{3.3}$$

where a_x and a_y are the components of the orientation vector. Fig. 3.13 shows the tracking between two consecutive vehicles.

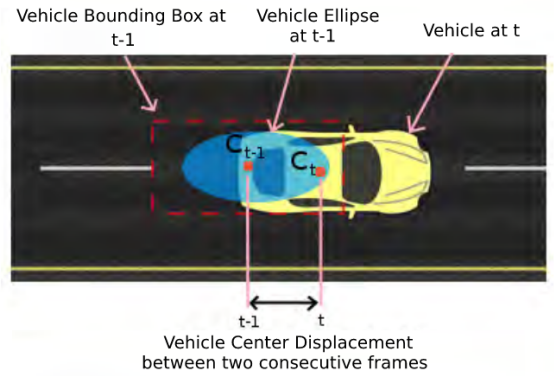


Figure 3.13: Proximity tracking ellipse.

Fig. 3.14 shows an example of *TrafficSensor* where the tracking of two vehicles by space proximity is shown. The vehicle identified as 2 in the image of the instant (t-1) is associated with the closest vehicle to its position in the current image (t).

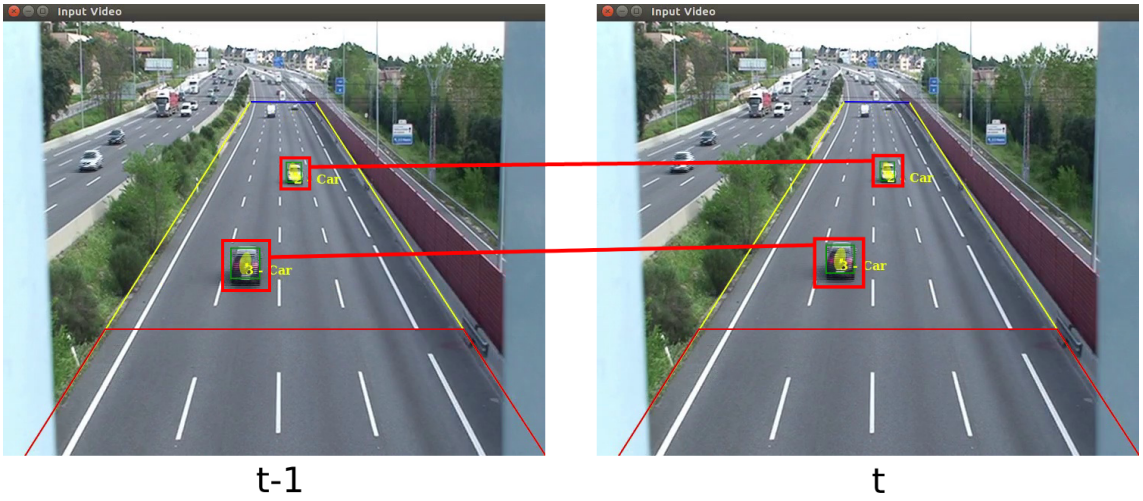


Figure 3.14: Tracking with spatial proximity *TrafficSensor*.

To be identified as the same vehicle, a detection in instant (t) must fall within a certain area around the blob detected in instant (t-1). In cases where two vehicles may fall within this area, the Euclidean distance between the center of the blob at instant (t-1) and the center of the blob at instant (t) is taken into account. The blob at instant (t) that is closest to the blob at instant (t-1) and is within the specified area is considered the same vehicle as in instant (t-1). In other words, if the distance between the blobs at (t-1) and (t) is minimal and falls within the predefined area, they correspond to the same vehicle in consecutive moments.

KLT tracking

The follow-up tracking primarily relies on spatial proximity, with the KLT algorithm serving as a fallback in challenging scenarios, thereby enhancing the robustness of our system. KLT is applied in all sequences to continually update feature points. In cases where a vehicle is not detected due to occlusion or distance, KLT proves effective, even over a small number of consecutive frames.

For KLT to be effective, knowledge of the vehicles' center of mass and their visual features is required. Based on the feature points of the vehicle at instant (t-1), KLT calculates matches for each feature point, generating a new set of feature points corresponding to the vehicle in question. Achieving accurate matches relies on a voting mechanism involving the associated feature points of an object. An example is illustrated in Fig. 3.15.

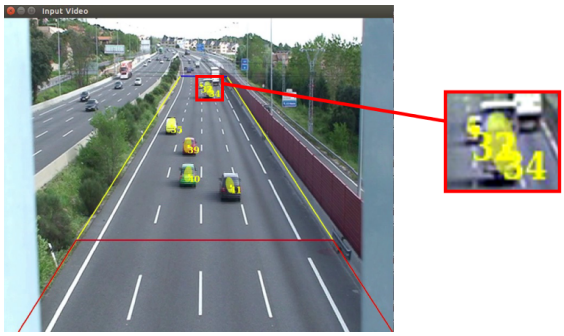


Figure 3.15: Tracking with KLT in *TrafficSensor*.

KLT is a feature-tracking algorithm known for its use of differential and local methods to analyze the neighborhood of each pixel. The algorithm operates under the assumption that the optical flow remains constant within a given neighborhood. By employing the method of least squares, the equation of optical flow is solved for all pixels within this neighborhood. The calculation of velocity vectors is achieved through the following equation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_{xi}^2 & \sum_i I_{xi} I_{yi} \\ \sum_i I_{xi} I_{yi} & \sum_i I_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_{xi} I_{ti} \\ -\sum_i I_{yi} I_{ti} \end{bmatrix} \tag{3.4}$$

The vector (u, v) is the displacement vector of the optical flow. Ix is the mean of gradient in x between two consecutive images, that is, if $I(t)$ is the image of the instant current and $I(t+1)$ is the image at the next instant, the Ix of these frames is:

$$I_x = \frac{I_x(t) + I_x(t+1)}{2} \tag{3.5}$$

where $Ix(t)$ is the gradient in the x axis of the image $I(t)$ and $Ix(t+1)$ is the gradient in x of the image $I(t+1)$. Iy is the mean of the gradients in y of the image $I(t)$ and $I(t+1)$:

$$I_y = \frac{I_y(t) + I_y(t+1)}{2} \tag{3.6}$$

It is the difference between $I(t)$ smoothed and $I(t+1)$ smoothed:

$$I_t = I'(t+1) - I'(t) \tag{3.7}$$

KLT is applied in the form of *kernels* of size ω x ω throughout from the image. The size of the *kernels* must be defined according to the amount of movement that the image has. A small *kernel* would be ideal for evaluating small displacements of a point. Using a large *kernel* increases the risk of getting an error, but there are cases where the displacement of a point is very big and this is necessary.

TrafficSensor uses the pyramidal implementation introduced in [230]. On it, the KLT algorithm is applied recursively over an image pyramid, as illustrated in Fig. 3.16.

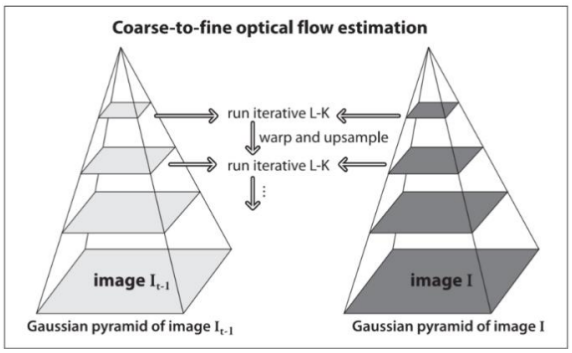


Figure 3.16: Pyramidal KLT.

3.3. Experimental validation

The proposed system is validated with a dataset of real traffic images, which has been divided into training and test subsets. Additionally, the four studied deep learning architectures for the detection and classification module of TrafficSensor have been quantitatively compared using an open-source tool, named Detection Metrics, so the one could be selected for the final system. This assessment software is also part of the contributions of this thesis and is presented comprehensively in Chapter 4. While utilized specifically for this work, it is applicable to various object detection applications. Furthermore, the final system undergoes testing and validation under varying lighting conditions, including poor visibility and adverse weather conditions commonly encountered in real-world deployments.

3.3.1. Dataset

To train and evaluate the networks, a new dataset was curated. This dataset includes images in good weather conditions, images in bad weather conditions (with fog and rain), and poor-quality images. This dataset consists of the following:

- The dataset compiled in [231], comprising 3460 good-quality images.
- The GRAM Road-Traffic Monitoring (GRAM-RTM) database, as introduced in [71]. This database features images extracted from three videos: *M-30* (7520 frames) captured on a sunny day, *M-30 HD* (9390 frames) recorded under cloudy conditions in a similar location, and *Urban1* (23435 frames) taken at a bustling intersection. From this extensive dataset, 3646 images from the *M-30 HD* video and 2348 images from the *M-30* video were utilized.
- Images sourced from publicly available online cameras, comprising 615 images depicting rainy conditions and 705 images of poor quality.

In total, the dataset used consists of 9774 images. All of them were manually annotated with LabelImg tool[10], using seven categories: car, motorcycle, ban, bus, truck, small, truck, and tank truck. In these 9774 images, we have a total of 48914 samples distributed as shown in Table 3.1.

Table 3.1: Dataset samples.

Class	**Sample**
Car	38976
Motorcycle	1886
Van	5631
Bus	401
Truck	963
Small-Truck	938
Tank-Truck	119

Table 3.2 shows the number of images that exist for each type of image (good conditions, bad weather, and poor quality) and Fig. 3.17 shows some illustrative images of our database.

Of these 9774 images, one part was used in training and another in the test. Table 3.3 shows the distribution of images according to training and test.

The training dataset for the neural networks was partitioned into train and validation subsets. Out of the 9246 images, 7401 were allocated for training and 1845 for validation.

[10] https://github.com/HumanSignal/labelImg

Table 3.2: Dataset images.

Type	Number of images
Good conditions	8406
Bad weather	663
Poor quality	705

Figure 3.17: TrafficSensor dataset samples.

Table 3.3: Dataset distribution.

Type	Training images	Test images
Good conditions	6717	389
Bad weather	1892	71
Poor quality	637	68
Total	9246	528

Table 3.4 provides a breakdown of the number of images used for training based on their quality and weather conditions (good quality, poor quality, and bad weather).

Table 3.4: Training dataset.

Type	Train Images	Validation Images	Total
Good conditions	5323	1394	6717
Bad weather	1568	324	1892
Poor quality	510	127	637
Total	7401	1845	9246

3.3.2. Comparison of deep learning models

The performance of the four trained deep learning models was assessed using images captured under good conditions. Additionally, the pre-trained versions of YOLOv3 and YOLOv4 were tested without fine-tuning for our dataset. These experiments were conducted using a Nvidia GPU GeForce RTX 3070. Table 3.5 provides the main features of the GPU used.

Table 3.5: GEFORCE RTX 3070 specifications.

GPU engine specifications	
Nvidia CUDA cores	5888
Base clock (GHz)	1.5
Boost clock (GHz)	1.73
Memory specifications	
Memory speed	14 Gbps
Standard memory configuration	8 GB GDDR6
Memory interface width	256 bit
Memory bandwidth (GB/sec)	448

The quantitative results obtained from the experiment are presented in Table 3.6. It

is evident that the YOLO networks outperform SSD MobileNetV2 and SSD VGG-16. Notably, all trained networks exhibit similar detection speeds. However, the use of pre-trained weights leads to longer detection times, likely due to these weights being trained with a broader range of classes.

Furthermore, the quality results, measured by mAP and mAR, are inferior when using pre-trained weights compared to trained weights, as anticipated. This underscores the importance of re-training the network using a suitable database tailored to the specific detection requirements.

Table 3.6: Results of trained networks

Neural network	**Framework**	**Performance**		**Mean inference time (ms)**
		mAP	**mAR**	
ssd300adam.h5	Keras	0.7478	0.7831	13
ssd_mobilenet.pb	TensorFlow	0.5484	0.61361	10
yolov3voc.weights	Darknet	0.8926	0.9009	15
yolov3voc_pre_trained.weights	Darknet	0.4577	0.5843	34
yolov4.weights	Darknet	0.9056	0.9670	13
yolov4_pre_trained.weights	Darknet	0.4799	0.5879	24

Indeed, YOLOv4 demonstrates notable enhancements in both detection quality and speed compared to YOLOv3. One key aspect contributing to this improvement is YOLOv4's utilization of data augmentation, allowing it to interpret the same information from various perspectives. This augmentation involves pixel-level modifications in the training images, such as changes in color, texture, black or white patches, cuts, and other alterations. These modifications enable the algorithm to enhance its precision and flexibility without compromising its speed performance.

In our tests, YOLOv4 exhibited a 13% increase in speed compared to YOLOv3, a result closely aligned with the findings reported by the authors of YOLOv4 [132], who reported a speed increase of 12%.

3.3.3. Experimental validation in good lightning conditions

For the final implementation of TrafficSensor, YOLOv3 and YOLOv4 were selected as they yielded the best results. To validate these final networks, the quality of the whole system was measured with *Detection Metrics* and the created testing dataset. Furthermore, for comparison purposes, the quality of the initial baseline system, TrafficMonitor [98], which lacks deep learning layers, was evaluated using the same dataset and measurement tool. In addition, *TrafficSensor* was compared to *Deep SORT (Simple Online and Real-time Tracking with a Deep Association Metric)* [232], which is an algorithm commonly employed in object tracking. It is an extension to *SORT (Simple Online and RealTime*

Tracker) [233] that incorporates appearance information through a pre-trained association metric. All systems were evaluated with the same good-condition videos and images.

Table 3.7: Results of good conditions video

System	mAP	mAR
TrafficSensor YOLOv3	0.8926	0.9009
TrafficSensor YOLOv4	0.9056	0.9670
TrafficMonitor	0.4374	0.5940
Deep SORT	0.8164	0.8689

The results obtained are shown in the Table 3.7. YOLOv4 and YOLOv3 have similar results although YOLOv4 is slightly better. This result was expected as the authors of YOLOv4 [132] indicated that the quality of the detections was superior to that of YOLOv3.

The results of *TrafficSensor* outperform those of *TrafficMonitor*. In the successive tests with *TrafficMonitor*, it became evident that it struggles with distant vehicles, often misclassifying cars as motorcycles and facing difficulty in distinguishing between cars and vans. Particularly, small vans are frequently misclassified as cars. This issue arises from the classification process, which relies on 3D models. Consequently, a small 3D van model may closely resemble the 3D model of a car, leading to misclassifications

In *Deep SORT*, the YOLOv3 Darknet network trained with our dataset was used and the results obtained by *TrafficSensor* and *Deep SORT* are very similar. *TrafficSensor* performs slightly better because it predicts the position of vehicles when they are not detected. *Deep SORT* uses Kalman Filter to predict and track, but predictions are used to improve detections, not to predict if there is no detection.

3.3.4. Experimental validation in poor conditions

The final *TrafficSensor* system was also evaluated with bad weather conditions and poor quality videos, as shown in Fig. 3.18. The Table 3.8 and Table 3.9 show the obtained experimentals results.

Table 3.8: Results of bad weather video

System	mAP	mAR
TrafficSensor YOLOv3	0.9899	0.9926
TrafficSensor YOLOv4	0.9904	0.9949
TrafficMonitor	0.2407	0.3162
Deep SORT	0.9801	0.9824

Figure 3.18: *TrafficSensor* with poor resolution (left) and bad weather (right) videos.

Despite being in rainy conditions the system can work successfully and with very good results. In this test, it can be seen that *TrafficMonitor* is not so robust, because it is not able to function correctly with rain. In the case of *Deep SORT*, again the results are similar to *TrafficSensor*.

Table 3.9: Results of poor quality video

System	**mAP**	**mAR**
TrafficSensor YOLOv3	0.9439	0.9444
TrafficSensor YOLOv4	0.9902	0.9911
TrafficMonitor	0.4479	0.6303
Deep SORT	0.8852	0.8910

With all the experimental results gathered, it can be concluded that *TrafficSensor* exhibits robustness against poor-quality images and bad weather conditions. In addition, it can continue tracking vehicles when they are far away from the camera. It works better with nearby vehicles, as they are easier to detect, but it is still able to detect and track distant ones with great quality.

Comparing the experimental results in the videos, the performance with poor quality videos and unfavorable weather conditions is slightly better than for good quality videos. This can be explained since the minimum requirements we set for good-quality images are higher than those for bad weather conditions and poor-quality videos. We do not expect the system to be able to detect distant vehicles in bad weather conditions and poor-quality videos. It is not even easy for humans to classify such vehicles. The images in the dataset have been labeled following this approach.

When evaluating the results obtained by *Deep SORT*, they are similar to those of *TrafficSensor*. *TrafficSensor* has greater precision since in cases where the neural network is not capable of detecting, it predicts such *detection* using the tracking algorithm.

3.3.5. Processing times

In the *TrafficSensor* system, three main processes can be identified: image processing (obtaining images, displaying images, obtaining data from the delimited road), detection, and tracking. Their computing time performance, both with both YOLOv3 and YOLOv4, has been monitored and evaluated. Table 3.10 illustrates the obtained results.

Table 3.10: Processing time

Function	**with YOLOv3 (ms/call)**	**with YOLOv4 (ms/call)**
Image processing	10	10
Detection algorithm	15	13
Tracking algorithm	18	18

3.4. Conclusion

TrafficSensor is a solution for vehicle surveillance using deep learning. It distinguishes between seven possible classes. Four different state-of-the-art deep learning models are studied and tested experimentally. A new dataset is created and used to train and evaluate the system, which includes a variety of images with poor quality or adverse weather conditions, more challenging for the system. TrafficSensor proves to be robust to bad weather conditions, and blurred or low-resolution traffic images. This improvement is achieved by training with the new specialized dataset and the combination of spatial correspondence tracking and KLT tracking on deep learning-based detections.

Both YOLOv3 and YOLOv4 deep learning architectures are selected for TrafficSensor since they are the ones that perform better. Comparing the two options, YOLOv4 outperforms YOLOv3.

In this work, we explore a problem in computer vision and begin to understand the possibilities of the field of autonomous driving. We developed a system for traffic monitoring and we can see the need for an automation system to increase security of improve the mobility of people. In the following chapter, we describe in detail Detection Metrics, which is the tool used in this chapter for assessment.

3.4.1. State-of-the-art enhancements

Since the inception and publication of this contribution, notable advancements have emerged within the state-of-the-art. While these recent contributions are not explicitly incorporated within this work, we acknowledge their significance. For instance, within the field of object detection, the YOLO architecture has undergone continuous evolution, with new

iterations being introduced (YOLOv5, YOLOv9...). Concurrently, there have been developments utilizing Transformer-based architectures (DETR, Swin Transformer, DINO...). These concepts are further elucidated in Chapter 2.

Chapter 4

Assessing object detection deep learning architectures with quantitative metrics

In the previous chapter, we explored the monitoring of traffic using state-of-the-art computer vision techniques. For that contribution, we developed a comparison tool that was used during the research process for the objective assessment of the object detection approaches that we were exploring. This comparison tool is Detection Metrics, which is a contribution of this thesis and is presented in this chapter. It has been published in a journal [234]. The most relevant international object detection datasets are supported along with the most widely used deep learning frameworks. Different network models from different frameworks can be compared fairly. This process is useful when developing deep learning applications or research. A set of tools is provided to manage and work with different datasets and models, including visualizations and conversion between several common formats. Detection Metrics automates the process of experimental validation launching the processes as batches, saving the researchers time. Using it, new domain-specific datasets can also be created from videos or webcam inputs. The tool is open source [235], and can be audited, extended, and adapted to particular requirements. In the previous chapter, we show how it has been validated experimentally for a research project, but it has also been used to compare the performance of some of the most relevant state-of-the-art deep learning models for object detection.

4.1. Introduction

In the field of computer vision, object detection stands as a fundamental task, involving the precise localization and classification of objects within images. As detailed in Chapter 2, recent advancements in computer vision owe much to various factors, including the availability of extensive high-quality datasets dedicated to object detection, the evolution of deep learning architectures, particularly convolutional neural networks, and the accessibility of powerful GPU hardware.

The development process for a deep learning-based object detector typically entails iterative experimentation, involving adjustments to model hyperparameters and fine-tuning to optimize performance. To facilitate this process, Detection Metrics has been developed as a contribution within this thesis. This tool provides objective performance metrics, allowing researchers to systematically evaluate different deep learning models on large datasets and determine the most effective approach for specific use cases.

Given its broad applicability, Detection Metrics can be employed across various object detection tasks. For example, it can be used for assessment of traffic monitoring deep learning solutions or the perception module of an autonomous driving system. It comprises a suite of tools, each offering unique features to facilitate the objective comparison of different deep learning models for object detection

4.2. Detection Metrics tool kit

Detection Metrics is a multi-platform command-line and graphical software application that provides several tools for comparison of object detection architectures objectively. Its GUI is based on the Qt framework (see Fig. 4.1) and written in C++. The application is natively built for Ubuntu and provided for the most common operative systems as a Docker image. Thanks to this technology, the functionality is the same independently of the platform used. It supports TensorFlow 2, Keras 2, PyTorch 1, Caffe2, and Darkent. It also supports commonly used object detection datasets like COCO, ImageNet, Pascal VOC, Princeton RGB, Spinello, and Open Images.

The application supports three use cases: headless mode, live detection, and use as ROS Node inside a distributed application. When used as a ROS Node, it acts as an executable node integrated into the distributed robotics application. This node can perform live detections, share them with other ROS Nodes, capture datasets, and store metrics.

4.2.1. Global architecture and workflows

The simplified architecture of the application can be illustrated as a black box (see Fig. 4.2). It usually receives a combination of datasets and a group of deep learning models and it generates the objective metrics for the experiment generation predictions using the deep learning models over the datasets provided. We call this workflow headless. Inside the black box, the application uses several tools that can also be used independently, especially when using the graphical part of the application. This use case allows the researchers to run several trained models over a batch of datasets easily at the same time, comparing their experimental performance and obtaining an idea of what is the best model for the problem at hand.

Going into the detail of the architecture, six differentiated building blocks integrate the toolset, as displayed in Fig. 4.3. They are the Viewer, Detection, Evaluation, Deployer,

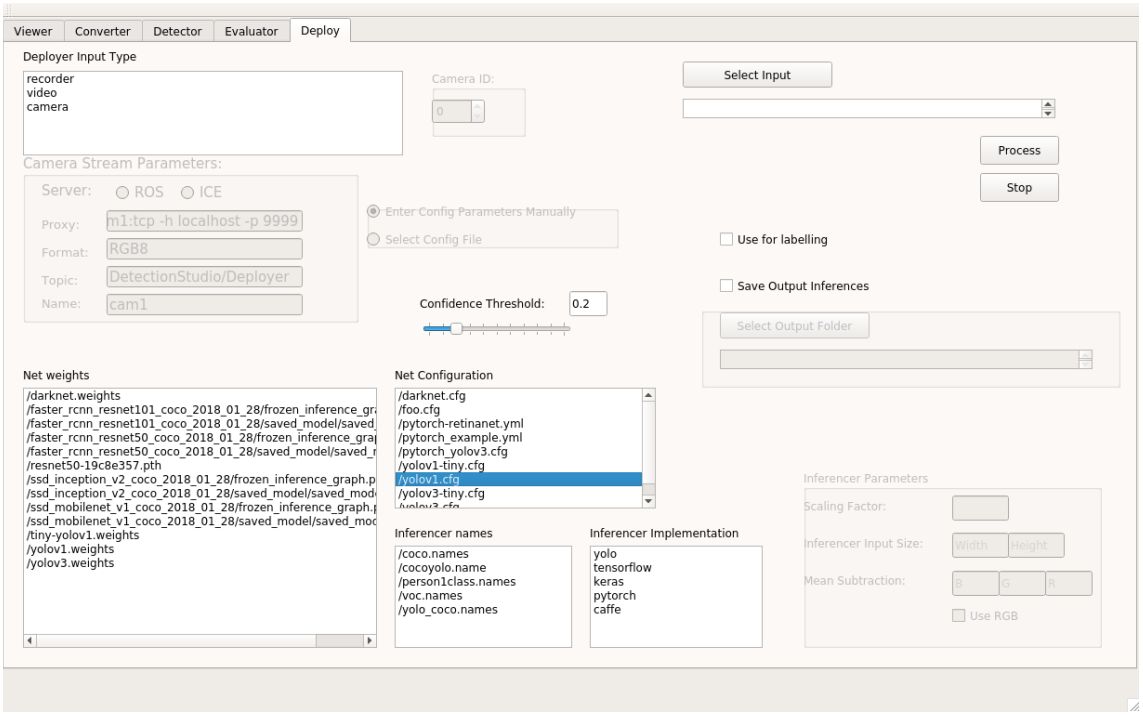


Figure 4.1: Detection Metrics GUI. The user can select the tool to use from the tool kit and enter the parameters directly using the graphical interface. In addition to the GUI, the headless mode is also available using the command line and a configuration file to access the functionality.

Labeling and Converter. These six parts can be combined in three divided sections. Two of them are the main workflows or modules.

The second use case is live detection visualization. The main difference with the headless evaluation is that the sources for the live detection can be videos or live streams (e.g. cameras). They generate the predictions online and these predictions can be saved or even modified with the Labeling functionality. Finally, the Converted remains disconnected from the main pipelines. It is used to convert datasets to other formats.

Viewer is a tool used to display annotated datasets that are not part of the main workflows. When evaluating using the GUI, the researcher can view the detection that the model is generating and compare them with the ground truth since both are displayed. It supports several dataset types: COCO, ImageNet, Pascal VOC, Princeton RGB, Spinello, and Open Images dataset. It also supports displaying and labeling depth images (for the datasets that give support to this feature) by converting them into a human-readable depth map.

The images are displayed along with their corresponding detected object with each bounding box and label. The bounding box and label have different colors for different classes. When used separately from the evaluation, it provides slightly different functionality. Given a set of images and annotations, it displays them one by one. Additionally, the final annotated images that Viwer displays can be further filtered based on some specific

Figure 4.2: Detection Metrics illustrated as a black box diagram. Detection Metrics receives a batch of datasets and deep learning models as input, calculates all the metrics from combining the datasets and deep learning models and finally outputs the metrics results.

classes (i.e., only particular classes will be labeled and only images containing those specific classes will be displayed). This option can be interesting when looking for images that contain objects belonging to specific classes.

4.2.2. Headless evaluation

The headless evaluation is one of the main use cases of the application. This mode is accessible directly via the command line. A researcher can determine a set of experiments that will run independently and unattended (fully autonomous), retrieving the final experiment report with the objective metrics that will help detect flaws and advantages for each model in each scenario.

This mode receives a batch of datasets and deep learning models, generates the predictions for each combination of models and datasets, and outputs objective metrics. Inside this process, three Detection Metrics tools are involved: Viewer, Detector, and Evaluator. When working detached as headless, these three tools work together as one but they are additionally available separately when using the GUI application.

4.2.3. Detection generation

Detector tool is responsible for generating a new annotated dataset with the predicted labels obtained from a deep learning model. The generated dataset contains the images with the predicted object detections, their position in the image, and probabilities for the predictions. Different inference frameworks are supported: TensorFlow, Keras, Darknet, Caffe, and PyTorch. When this tool is run, it also communicates with the Viewer to show the detections with the ground truth, giving an intuition of the performance visually.

To provide the different frameworks support, Detection has interfaces for each of them, connecting the actual framework to the tool in an agnostic way that prevents the user from facing any complexity. Thanks to the modularity of the Detector tool and the fact that the project is open-source, new deep learning frameworks may be added seam-

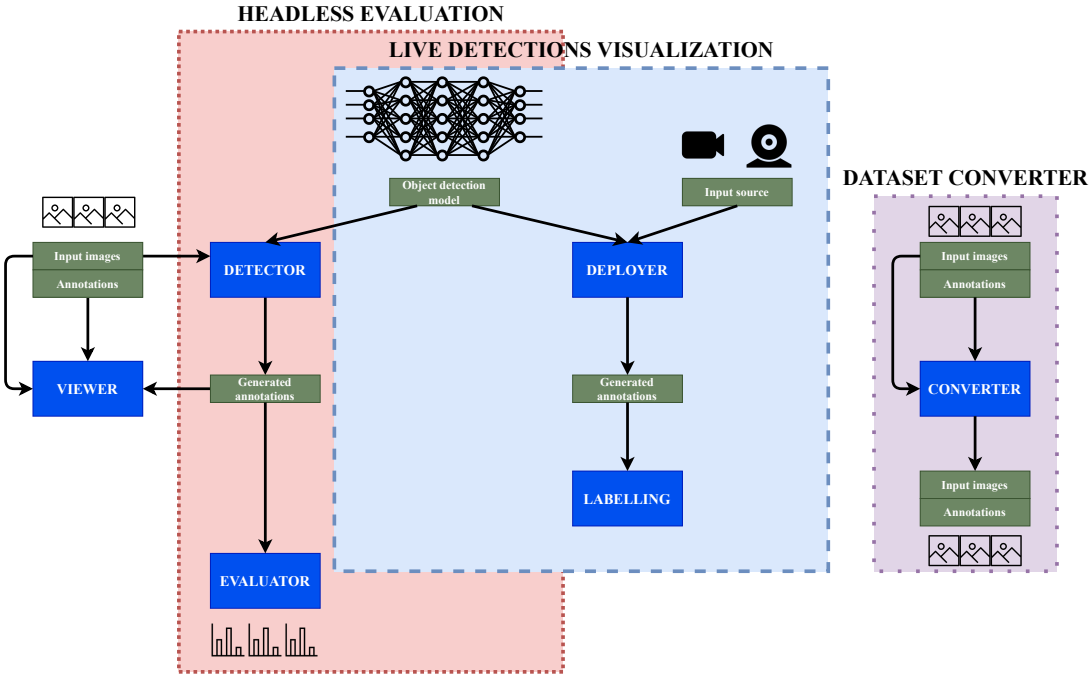


Figure 4.3: General Detection Metrics architecture. The software provides three main use cases: headless evaluation, live detection visualization, and dataset converter. Each of them has a set of tools (in blue), that can be used individually or combined.

lessly. In addition, the scientist saves time since they create the experimental description structure and it runs autonomously without explicitly considering the differences of the underlying frameworks or dataset structures and only focuses on the experimental results.

4.2.4. Evaluation of detections with objective metrics

Evaluator can evaluate two annotated datasets with the same dataset format on a fully autonomous basis considering one as the ground truth and the other as the generated detections dataset. Evaluator support mAP and mAR metrics. It outputs these metrics for each class and a range of IoU thresholds.

Every object detection in an image will be evaluated, comparing the detection in both datasets. Since the evaluation procedure in the application is written in C++, it provides faster performance than the original COCO toolbox written in Python. This procedure is done for every image in the dataset, loading, comparing, and then releasing the resource, making a fair comparison.

When running in headless mode, the set of experiments is evaluated after the two previous steps, and then creates a report in *csv* format with the experimental information.

Using the GUI, Evaluator can also be used independently, providing additional features. The valuation can be further filtered by a specific object class from the detected dataset, so only the classes selected will be considered during the evaluation. There are

two types of IoU available in Evaluator: bounding boxes and masks. Additionally, the different person classes available in some of the dataset class names can be merged into just one person class that contains all the different ones.

4.2.5. Live detection visualization

The second main use case is live detection visualization. One of the main differences between this use and the headless is the input source. For the Deployer, the main tool used for this use case, the input can be a video file or even a stream of images coming from a video camera.

Once the configuration is decided, the tool displays a video player that plays the input while displaying the objects detected with their class names in real time. If the input is a video file, two video players are displayed, one of them with the raw video and the other one with the video and detected objects, similar to Detector. This video player offers typical play/pause functionality and goes backward or forward in the playback frame by frame. Another feature provided by Deployer is the confidence threshold (minimum value to consider the detection) that can be adjusted to different values to show the differences in the inferences in real time. This will affect the real-time detection in the video since then, if the threshold is set to a high value the number of objects that will be found in a frame will probably be lower and the other way around.

The predicted labels can be saved to an output file if needed, setting an output folder, which may be used to create a new dataset with annotations from a video record or web-cam output.

Labels correction on demand

Deployer comes with some labeling tools. This functionality is provided in the video player created when using Deployer:

- The first feature is the possibility of adjusting the bounding boxes generated. The user can adjust the size and position of a certain detection bounding box stopping the video when the error is found and adjusting the distribution of the box to the object.
- The second feature is changing the class name for every detected object. This means that a user can select a detected bounding box in the video image and change the class name in real time to one of the class names provided or to a completely different one, also having the chance of adjusting the probability of the selection.
- The third feature is related to the previous ones and is adding new detections. The user can draw a new bounding box in a stopped frame and then give this a class name and probability.

This workflow can be interesting for generating new datasets, creating proposals using the assistance of a deep learning model, and adjusting them by hand.

4.2.6. Dataset converter

The dataset format is usually specific for a certain implementation, so the purpose of this tool is to convert a dataset format into another format. This tool receives as input a dataset with the object's class names that are supported by it and the type of dataset format it implements. It needs the type of dataset as input to create a reader, a tool that understands the format for a specific dataset. The format implementation of the wanted dataset to be converted is also needed, so Detection Metrics creates a writer, another tool that knows how to create a specific dataset format. Converter allows filtering by classes and mapping between corresponding classes. This means that in the case that the object class names in the input and output datasets are different, the application tries to map from the input class names to the output ones, considering the common class name connection between the common datasets and also considering synonyms.

The converted dataset can be split into test and train parts. To do so, a training ratio is provided to the tool and it divides the dataset into two separate parts. This option can be useful to create divisions of the converted dataset.

After the conversion is completed, Viewer functionality can be used to display the converted dataset and make sure the process is completed successfully or it also can be used with the different tools provided by Detection Metrics.

4.3. Experimental results and discussion

In this section, we present an experiment conducted using Detection Metrics. We have also used this tool for the contribution presented in Chapter 3 that has been published [224] [98]. In the experiment, we compare the performance of the most well-known state-of-the-art detection networks and validate the published results from the original network authors.

4.3.1. Comparison of state-of-the-art detection networks

In this experiment, four different pre-trained object detection networks are evaluated using Detection Metrics. The goal is to compare the results obtained by the toolkit with those published by the original authors. The selected networks include several popular object detection methods [89]: SSD, Faster RCNN, and YOLOv3. In the process, the *Headless evaluation* mode of Detection Metrics (Fig. 4.4) was used. The measured performance metrics are compared among them and also with those published by the authors.

The evaluation dataset is COCO minival, a small subset of COCO's validation set.

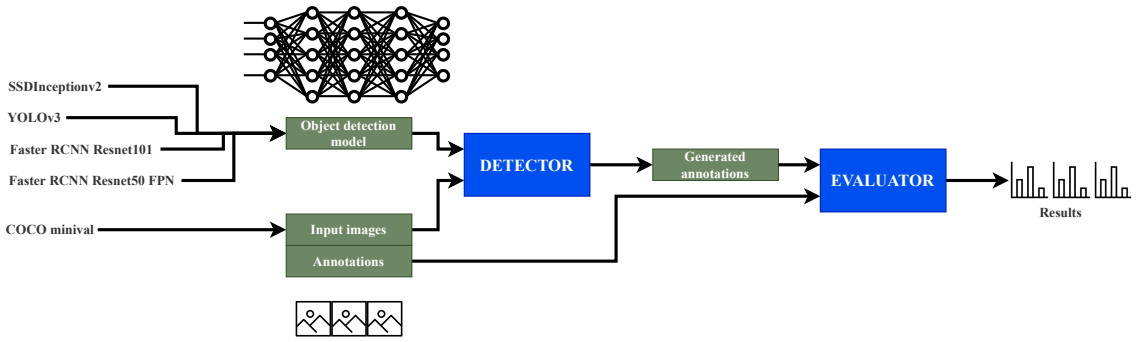


Figure 4.4: Experiment pipeline using headless evaluation. Detection Metrics receives a set of deep learning models and a dataset and generates annotations with Detector that are the input to Evaluator for obtaining the experimental results.

Table 4.1: Comparison of official network results with results generated using Detection Metrics. Our software is used to replicate the official results of common network architectures programmed in different deep learning frameworks, probing the software capabilities for working with different frameworks and providing common metrics that match the official results. ✘: official results do not give that information.

Network	Framework	Published mAP	mAP using Detection Metrics	Published mAR	mAR using Detection Metrics	Published Mean inference time	Mean inference time using Detection Metrics
SSD Inceptionv2	TensorFlow-Keras	0.24	0.27	✘	0.31	42	44
YOLOv3	Darknet	0.55 (IoU = 0.5)	0.47 (IoU = 0.5)	✘	0.5 (IoU = 0.5)	29	31
Faster RCNN Resnet101	TensorFlow-Keras	0.32	0.37	✘	0.43	106	122
Faster RCNN Resnet50 FPN	PyTorch	0.35	0.37	✘	0.46	59	102

Since the dataset is part of the validation set, some networks could be biased towards having greater performance than the real one (with a test dataset which they have not ever seen) because they were trained on the COCO dataset. The experiments were run on an Nvidia GeForce GTX 1080 GPU.

The selected networks are an implementation of SSD Inception v2, a Faster RCNN Resnet 101, YOLOv3, and a Faster RCNN Resnet 50 FPN. The first and second are downloaded from the TensorFlow detection model zoo [236]. It offers a broad variety of pre-trained networks with metrics. For YOLOv3 the configuration and weights were downloaded from the official documentation and the fourth is included in PyTorch vision model zoo [237]. With this set of different networks, the wide variety of frameworks supported is shown in a real experiment, involving in this experiment TensorFlow, PyTorch, and the YOLO-OpenCV module.

In Table 4.1, the results obtained are shown. For SSD Inception v2, YOLOv3, and Faster RCNN Resnet101 networks the mean inference times are close to the ones provided by the original researchers, slightly higher for the experiments conducted with Detection Metrics. This is probably due to the different GPU used and computational load at the

time of the experiment on the computer. TensorFlow's pre-trained networks and YOLOv3 official results were obtained using an Nvidia GeForce GTX TITAN X card. Regarding the Faster RCNN Resnet50 FPN network the difference is significant, maybe because this PyTorch's pre-trained network was tested by its authors using 8 V100 GPUs and in the experiment with Detection Metrics a single GPU was used.

Detection Metrics considers both AP and AR in the evaluation, providing these metrics from an IoU of 0.5 to 0.95 and the mean of each metric for that range. The mAP measured values are also approximately equal to those published by the original researchers, with slightly better numbers when using Detection Metrics, in general. The numbers confirm the results provided by the authors of each network. Regarding the network comparison, YOLOv3 is the best-performing network in mAP, as expected.

With this experiment, the use of Detection Metrics for the validation of the results of widely used detection network models and their cross-framework comparison has been illustrated.

4.4. Conclusion

Detection Metrics is an open-source software for the automatic assessment of deep learning object detection models. It is a contribution of this paper and has been published in a journal [234]. We have described its two main workflows for working with object detection networks and large datasets. They are the headless evaluation, which evaluates automatically several models independently for a set of image datasets returning objective metrics, and the live detection visualization, which allows real-time visualization of predictions. The workflows have been proved experimentally in Chapter 3 and Section 4.3. The software is open-source, its code can be audited, modified, or extended for the particular needs of the project. The source code is accessible via [235].

In the following chapter, we present another contribution of the thesis, Behavior Metrics, where we understand the importance of the assessment software for benchmarking solutions for autonomous driving, which is the core of the thesis. After we have understood the importance of the assessment of solutions in order to compare them objectively here, we move the area of application to autonomous driving, studying the assessment of solutions in that field.

4.4.1. State-of-the-art enhancements

Since the development of this contribution and its publication, new contributions have appeared that are relevant in the state-of-the-art. We do not include them in this development but we acknowledge them. For example, concerning the object detection field, YOLO architecture is continuously evolving and new versions have been released (YOLOv5, YOLOV9...). In parallel, some developments that use Transformer-based architectures

have also appeared (DETR, Swin Transformer, DINO...). This ideas are described in Chapter 2.

Chapter 5

Assessing autonomous driving behaviors fine-grained metrics

In this chapter, we describe one of the contributions of the thesis, Behavior Metrics [238], which is published as a journal publication [239]. Behavior Metrics [240] is a software tool that we have developed to help research in the autonomous driving field. The development and validation of autonomous driving solutions require testing broadly in simulation. Addressing this requirement, we present Behavior Metrics for the quantitative and qualitative assessment and comparison of solutions for the main autonomous driving tasks. This software provides two evaluation pipelines, one with a graphical user interface used for qualitative assessment and the other headless for massive and unattended tests and benchmarks. It generates a series of quantitative metrics complementary to the simulator's, including fine-grained metrics for each particular driving task (lane following, driving in traffic, route navigation, etc.). It provides a deeper and broader understanding of the solutions' performance and allows their comparison and improvement. It uses and supports state-of-the-art open software such as the reference CARLA simulator, the ROS robotics middleware, PyTorch, and TensorFlow deep learning frameworks. BehaviorMetrics is available open-source for the community.

5.1. Introduction

As we have already discussed in chapter 1 and chapter 2, the autonomous driving field has gained incremental popularity in recent years. The adoption of this robotics and AI technology will significantly impact the future, emphasizing the critical need for reliable and secure solutions.

We present here Behavior Metrics, a multi-platform open-source software for the assessment of autonomous driving solutions in simulation for different driving tasks (currently lane following, driving in traffic, and navigation between points). It assists both everyday users and researchers in developing and validating autonomous driving solu-

tions by augmenting simulator-generated metrics with enhanced evaluation metrics.

It supports both CARLA and Gazebo simulators using ROS as communication middleware. It can be used with different sensory inputs for the vehicle like a camera, LIDAR, or any other type of sensory data input supported by the simulators, like the bird-eye-view. They all are managed and added to the simulation using the configuration file. The tool conducts comprehensive online evaluation across driving tasks, yielding objective, fine-grained metrics superior to those provided by simulators. It offers both GUI-based interaction and headless batch processing for large-scale testing.

Designed for the research and advancement of autonomous driving, it establishes a unified framework for evaluation and facilitates the creation and automated execution of extensive benchmarks across various vehicle types, dynamics, lighting conditions, and scenarios. This enables fair comparison among different approaches, including deep learning, reinforcement learning (RL), or explicit programming, providing valuable insights for enhancing each method.

5.2. Software description

Behavior Metrics' software architecture (Fig. 5.1) is based on a Model-View-Controller (MVC) design pattern implemented in Python. The evaluation configuration is described in a dynamic YAML configuration file, including the scenario, vehicle, driving task, sensors, and vehicle robot controller. Using this configuration, Behavior Metrics conducts the experimental evaluation, initiating the simulator with the ego vehicle, utilizing the vehicle's robot controller for driving, and ultimately generating comprehensive evaluation metrics for performance insights. The user may change or include any part of the experimental setup like scenario, vehicle (e.g. model), sensor... modifying the configuration file.

The tool supports evaluation in two simulators, CARLA and Gazebo, through integration with ROS 1 Noetic. ROS manages communication between the application and the simulators, allowing for reusable code between the simulators' handlers.

Behavior Metrics communicates with the simulators using the publish/subscribe design pattern of ROS (details in Fig. 5.2). For example, the application subscribes to the sensor nodes of the ego vehicle to extract the raw data that are then processed by the robot controller and it publishes messages to control the vehicle that are translated to the actual movement of the vehicle in the simulation. Behavior Metrics enables actions like playing or pausing the simulation and controlling simulator processing steps (simulation speed). The raw sensory and simulator data undergo processing to generate evaluation metrics for assessment.

The vehicle controller (Fig. 5.3) is responsible for the ego vehicle motion. It reads the sensory input provided by the sensors attached to the vehicle, like the camera, bird-eye-view images, ground-truth segmentation camera, or odometry, and processes them. Based

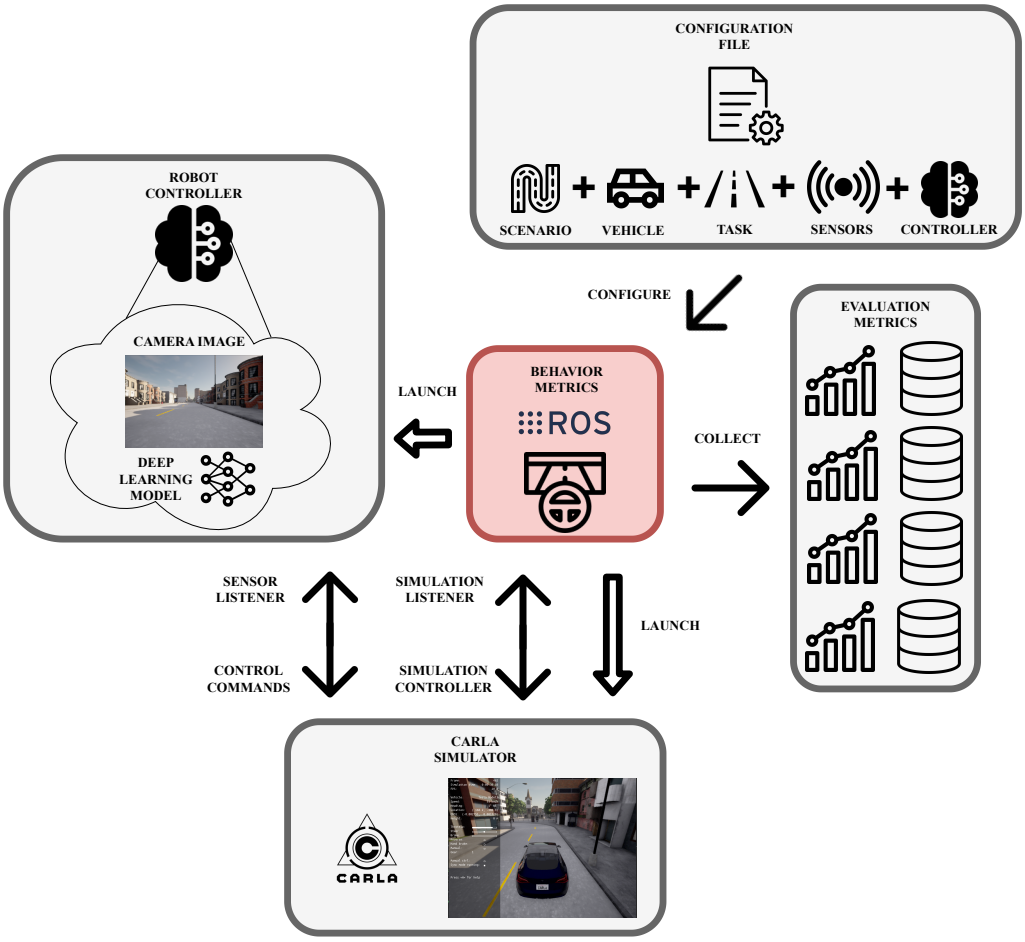


Figure 5.1: Behavior Metrics tool architecture. The configuration file describes the setup of the evaluated experiment.

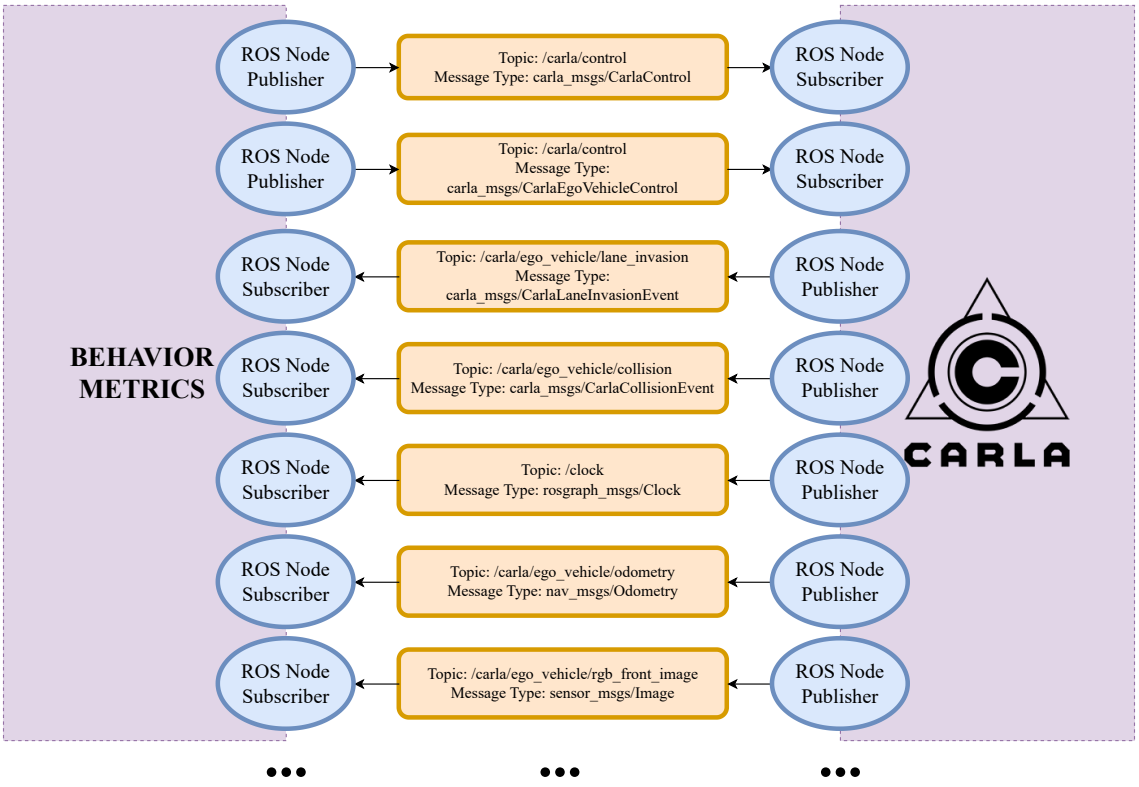


Figure 5.2: Some of the connections between Behavior Metrics and CARLA.

on the knowledge extracted from the input, it iteratively generates the control outputs that are commanded to the actuators. Control commands may be generated using a deep learning model, an RL policy, or even an explicitly programmed algorithm. This abstraction layer facilitates the use of different types of vehicles without any code modification. Behavior Metrics supports the most common deep learning frameworks (TensorFlow and PyTorch).

It provides control over the simulation time speed, allowing for the selection of either asynchronous or synchronous time modes, and even managing the simulation iteration time-step. By default, the simulator operates asynchronously, making simulator time independent of Behavior Metrics and its vehicle robot controllers. Simulated time becomes crucial for low-resource systems requiring more time for controller iterations. Spending excessive time in this process could result in vehicle control malfunctions, even with correct decisions. Considering the vehicle speed and safety standards needed for autonomous driving, the time spent per iteration is crucial. This flexibility enables researchers to test solutions in a broader range of conditions, including simulating systems with limited resources.

Behavior Metrics is compatible with all CARLA towns and allows the management of traffic conditions (traffic lights, traffic signs), simulation start and end points, and weather conditions. With this approach, Behavior Metrics can be used for a full autonomous

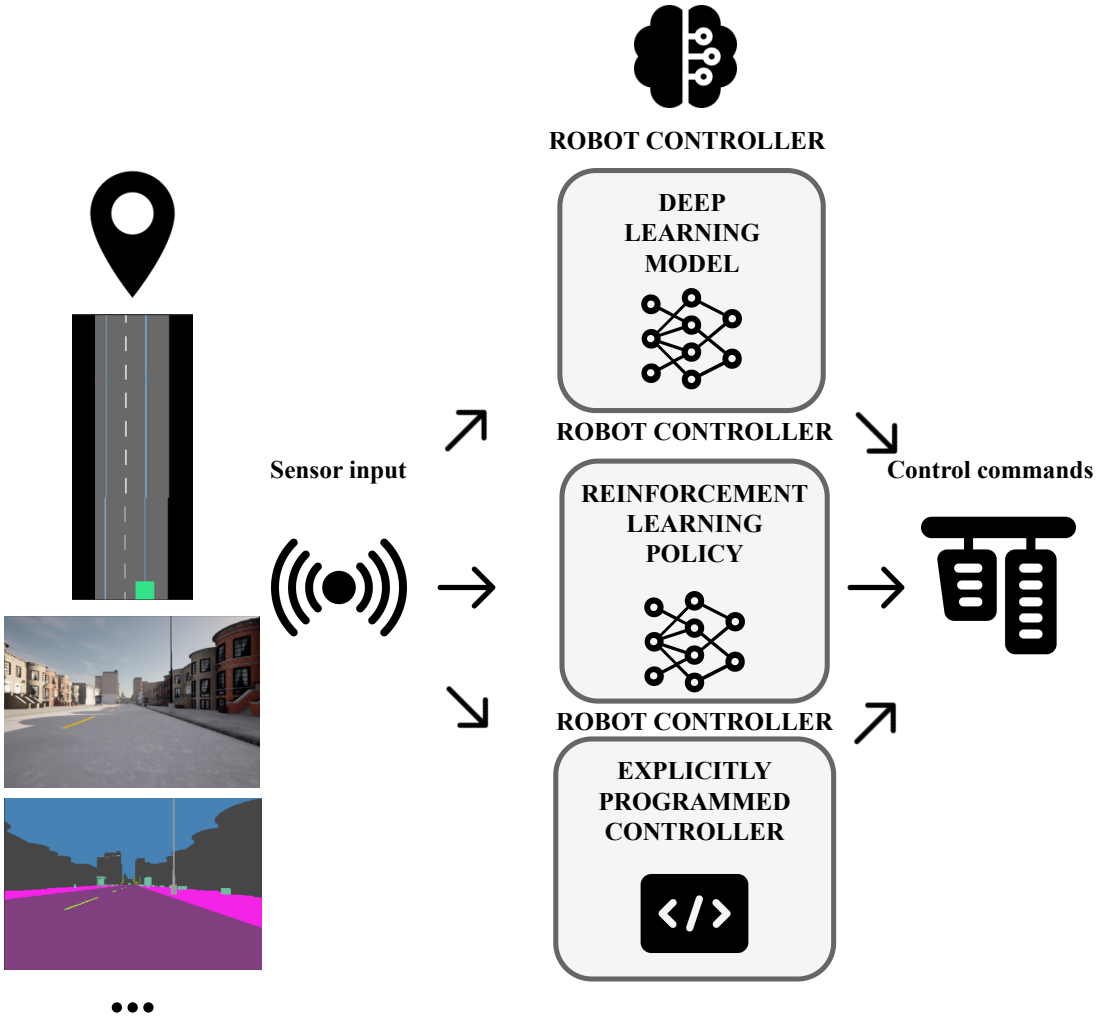


Figure 5.3: Details of the robot controller, three types are supported.

driving agent evaluation or an evaluation of a specific driving task like lane following or driving in traffic. This precise level of control is particularly valuable for researchers focusing on specific driving tasks, where detailed control is essential.

The software is designed to be highly versatile and cross-platform. It achieves this using Docker [241], which facilitates effortless sharing across various operating systems. It is encapsulated within a Docker image, enabling deployment on the most prevalent operating systems. Moreover, the software's core functionality is native to Linux, enabling direct usage on Linux computers without relying on the Docker image. This approach enhances user experience and flexibility.

5.2.1. Supported driving tasks

Behavior Metrics supports three distinct driving tasks with varying difficulty levels, contributing to research advancements across different aspects of an autonomous driving application: lane following, driving in traffic, and route navigation. The first involves accurately staying within the lane without encroaching on adjacent lanes. The second consists of an ego vehicle and a series of vehicles dispersed along the route, requiring the vehicle to correctly follow the lane while maintaining a safe distance from potential front vehicles. The third combines the previous ones adding a starting and ending point to the experiment setup. The vehicle will drive between goal points while following the lane and maintaining a safe distance from front vehicles. This task also considers traffic lights, traffic signs, and intersections. Other driving tasks can be easily defined along with their metrics since the software tool is provided open-source and its design is modular.

5.2.2. GUI and headless evaluation modes

The application provides two evaluation modes, the graphical user interface (GUI) mode and the scripted mode (headless). The GUI mode generates a user interface, implemented using the PyQt5 framework, based on a configuration file describing the simulation environment (scenario, ego vehicle, traffic,...). Using this mode, users can seamlessly execute experiments while visually monitoring the ongoing evaluation process (Fig. 5.4). Alongside the classic simulator view, this interface displays sensor information and provides convenient buttons for starting, stopping, or restarting the evaluation. This mode is typically employed for qualitative autonomous driving solution evaluation. After completing the experiment, quantitative evaluation results are graphically displayed in a separate window and saved in files for future analysis.

In headless mode (Fig. 5.5) the user defines a configuration file with the evaluated task, all the scenarios, robot controllers, and models to be evaluated, as well as the number of experiment repetitions. Behavior Metrics conducts the experiments as a batch, without user intervention. No graphical part is displayed during the evaluation, and the results are directly saved to files for subsequent analysis. In addition to results from each

Figure 5.4: Behavior Metrics GUI architecture using CARLA simulator. It displays two separate windows: the application GUI and the simulator.

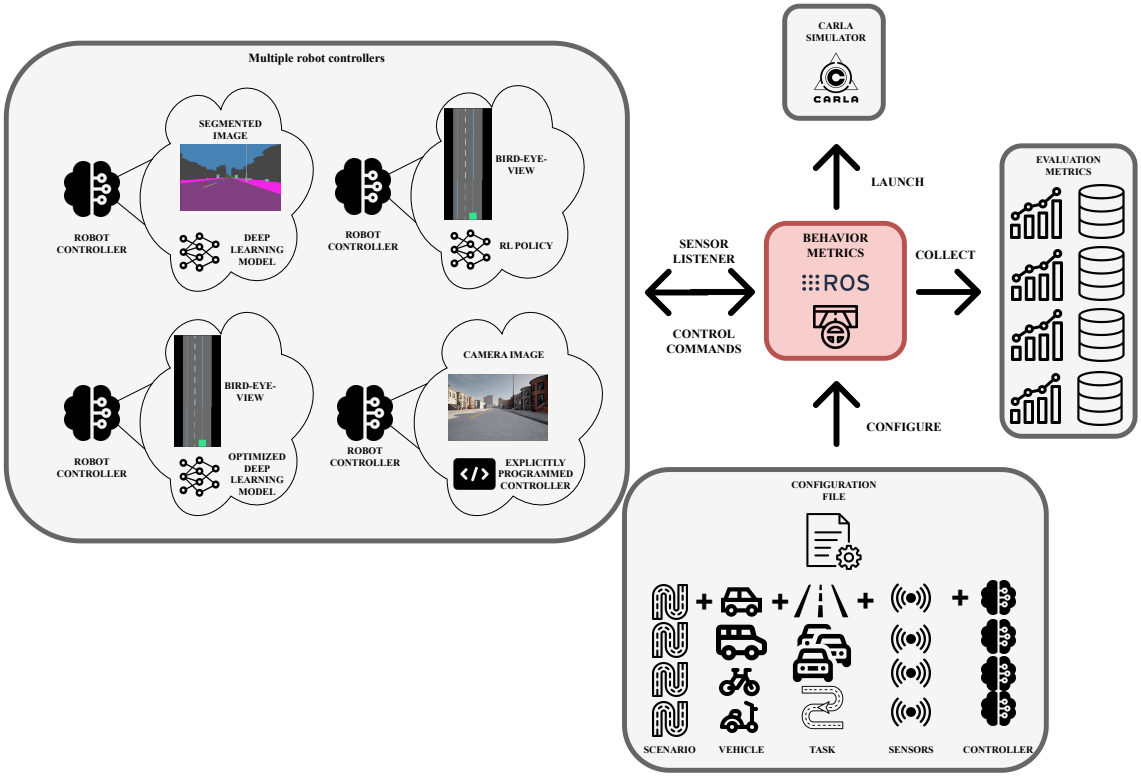


Figure 5.5: Behavior Metrics headless evaluation mode.

of the experiments, this mode generates combined results for all the run experiments together, making it easier for the researcher to compare directly how a specific controller behaves. Setting up configuration files in this manner enables Behavior Metrics to support the creation of extensive benchmarks that can be automatically executed.

5.2.3. Autonomous driving evaluation metrics

Behavior Metrics generates a set of quantitative evaluation metrics that complement those directly provided by the simulator and other evaluation frameworks like the CARLA Leaderboard, offering a more informative and complementary perspective on the behavior of a specific controller. The supplied metrics have been selected as they have been required in several research works. Adding more metrics, such as the CARLA Driving Score, is pretty straightforward as long as the raw data are generated by the simulator.

- **Mean position deviation per km (MPD)**: average deviation, in meters, of the ego vehicle from the center of the lane that it is traversing. Calculated using the mean of all the points obtained using the minimum Euclidean distance (*MinED*) of each traversed position (*EgoVehiclePosition*) to the center of the lane (*centerOfLane*).

$$\text{MPD} = \frac{1}{N} \sum_{i} \text{MinED}(\text{EgoVehiclePosition}_i, \text{centerOfLane}) \tag{5.1}$$

- **Effectively completed distance**: distance completed during the experiment in meters that passes through checkpoints that bind its starting and end points. These checkpoints are centered on the lane followed during the experiment, so it is an indicator of how consistently the vehicle drives on the lane.

- **Vehicle longitudinal jerk per km (VJ)**: metric that defines if the vehicle drives smoothly or makes jerks in velocity during the experiment. It indicates whether the conduction is aggressive or smooth. Approximated by calculating the mean of the differences between the current and previous speeds(*VehicleSpeed*).

$$\mathrm{VJ} = \frac{1}{N}\sum_{i}(\mathrm{VehicleSpeed}_i - \mathrm{VehicleSpeed}_{i-1}) \tag{5.2}$$

- **Robot controller iteration frequency.**

- **GPU inference frequency**: number of GPU iterations per second when the robot controller has a deep learning model that uses the GPU as the core computational element.

- **Collisions per km**: number of collisions of the ego vehicle during the experiment per kilometer.

- **Lane invasions per km**: number of lane invasions infractions committed by the ego vehicle per kilometer.

- **Distance to the front vehicle**: this metric indicates how close the ego vehicle circulates to other front vehicles and gives insight into whether it follows safety standards or not. The distance is divided into four categories: great distance (20-50 meters), medium distance (15-20 meters), short distance (6-15 meters), and dangerous distance (0-6 meters), and it is provided as the percentage of experiment time that the ego vehicle spends on each category (see Figure 5.6).

- **Route completion percentage**: for route navigation task, percentage of route completed for each of the conducted experiments.

- **Average speed**: achieved by the ego vehicle during the experiment.

- **Successful experiments**: this metric is tuned depending on the task. In general, an experiment is considered successful when the safety drive conditions are met but for each specific task, this metric is slightly tuned. For a lane following, for example, Behavior Metrics considers that the vehicle drives at a constant speed and that it does not deviate above a threshold from the middle of the lane. For driving in traffic, Behavior Metrics also considers that the car distance to the front vehicle is not dangerous. For route navigation, Behavior Metrics measures whether the vehicle has reached the goal position following the user's directions.

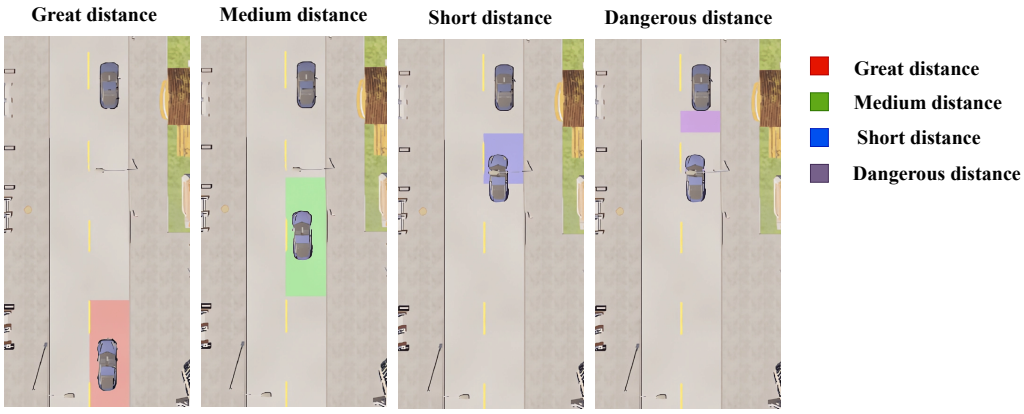


Figure 5.6: Great, medium, short, and dangerous distances to the front car.

5.3. Illustrative examples

The project repository contains example files for each component required for running example evaluations, including the configuration files, robot controllers, and imitation learning-based deep learning models for supported driving tasks in different simulators. It includes examples for TensorFlow and PyTorch frameworks.

5.3.1. GUI application example

Using the configuration file edited by the researcher, Behavior Metrics generates the experiment visual setup, including the Behavior Metrics and the simulator windows [242] (Fig. 5.4). The configuration file is editable to customize the scenario, vehicle, task, included sensors, or vehicle controller. The Behavior Metrics GUI incorporates functionality for initiating the evaluation, watching the sensor state, and the simulated ego vehicle performance. Researchers can then commence the simulation and experiment recording. Upon completion, researchers stop the recording and the simulator, and Behavior Metrics visually presents the results and saves them in log files for subsequent detailed analysis.

5.3.2. Headless application example

In this case, the configuration file comprises a list of vehicle controllers, scenarios, and experiment repetitions to evaluate. Behavior Metrics evaluates each of the combinations in an unattended manner, without graphical information while evaluating [243] (Fig. 5.5). Once the experiments conclude, Behavior Metrics furnishes results for each case and the aggregated outcomes for all combinations.

5.4. Impact

This contribution impacts the field of autonomous driving, which is a relevant research topic as proved by the presented state-of-the-art (see Chapter 2) and the CARLA Leaderboard challenge [11]. It provides a common framework for testing autonomous driving solutions (deep learning models, RL algorithms, etc.), and supports different driving tasks.

The two evaluation modes, GUI and headless, streamline the process of evaluating and rapidly testing ideas, as well as supporting the iterative development of autonomous driving solutions. Furthermore, it facilitates a comprehensive evaluation for the comparison of different solutions and models. It leverages CARLA, a highly regarded autonomous driving simulator, thereby amplifying its impact.

Behavior Metrics has been effectively employed to evaluate end-to-end solutions exploring various concepts. These include assessing the significance of utilizing vehicle controllers with memory and kinematic input, which is described in Chapter 6, studying the impact of model optimization on controller iteration speed, and analyzing the resulting behavior of the vehicle, which is described in Chapter 7 and has also been published in a journal [244]. It has proved its capabilities in the evaluation of solutions for the driving in traffic task, as described in Chapter 8, with specific metrics, such as distance to the front vehicle. It has also served as an assessment tool for Google Summer of Code open-source autonomous driving project [245] on route navigation capabilities through metrics like route completion.

5.5. Conclusions

In this chapter, we have introduced and described Behavior Metrics, an open-source evaluation software designed for assessing autonomous driving solutions. It constitutes one of the contributions of this thesis. We highlight the software's potential utility for researchers in assessing autonomous driving solutions across various driving tasks. the included metrics, complementing simulator-provided metrics and unique to our solution, provide a more comprehensive understanding of vehicle performance, contributing to the development of improved solutions for diverse driving scenarios.

This software offers two distinct pipelines, setting it apart from other solutions: GUI mode and headless mode. They empower researchers to qualitatively and quantitatively test their solutions, facilitating comparisons with alternative approaches. Additionally, our software supports state-of-the-art simulators and deep learning frameworks broadening its accessibility within the research community.

In the subsequent chapters, we present additional contributions to this thesis that leverage Behavior Metrics for the assessment and experimental validation of their research questions. The software is open-source, its code can be audited, modified, or extended for

[11] https://leaderboard.carla.org/

the particular needs of the project. The source code is accessible via [238].

Chapter 6

Enhancing end-to-end autonomous driving control though kinematic input and memory-based architectures

In this chapter, we introduce one of the contributions of this thesis [246], which is under peer-review in a journal at the time of writing this document [247]. We have already discussed the importance of monitoring the traffic using computer vision and how its implications for road safety. After that, we have explored how we can evaluate and research solutions for this monitoring and also for autonomous driving solutions. In this chapter, we get deeper into the autonomous driving field.

We explore and compare various approaches to enhance the capabilities of an end-to-end system for autonomous driving based on imitation learning adding visual memory and kinematic data input. The comparison relies on fundamental error metrics (MAE, MSE) and several external complementary fine-grain metrics based on the behavior of the ego vehicle at several test scenarios in the CARLA simulator. The problem focused on a lane-following application using different urban scenario layouts and visual bird-eye-view input. The memory addition covers architectural modifications and different types of sensory input. We show experimentally that incorporating visual memory capabilities and kinematic input data makes the system more robust and able to handle a wider range of challenging situations in terms of reduction of collisions and speed self-regulation. All the work we present in this chapter is open-source, including model architectures, trained model weights, comparison tools, and the dataset.

6.1. Introduction

We have already described the importance that the autonomous driving field has currently and is expected to have in the coming years. As we discussed in Chapter 2, the solutions for autonomous driving are typically divided into two groups: end-to-end and modular

approaches. In the first one, the raw input data is directly translated to control commands or trajectory predictions in a single forward pass while in the second one, several modules communicate with each other to finally generate the output. The use of end-to-end solutions has been around for a few years, from PilotNet. In this contribution, we explore how the introduction of memory capabilities and kinematic data affects the end-to-end system, contributing positively or negatively to the overall solution. For it, we consider architectural changes and different types of input data.

We use the previously presented (see Chapter 5) software tool Behavior Metrics [238] for the quantitative and qualitative assessment of the solutions. We introduce and compare several deep learning architectural modifications that incorporate memory models and different sensory inputs to explore the potential benefits of different memory approaches in solving the problem of end-to-end autonomous driving through imitation learning. We also explore the inclusion of kinematic data as input to the models. The research hypothesis that we explore is that adding visual memory capabilities to the deep learning architecture and kinematic input data improves the quality of the generated robot control behavior in terms of the system's robustness for never-seen situations and speed self-regulation. We analyze how much the addition of memory and kinematic data enhances driving behavior and when it can have a significant impact. Our study is supported by a series of experiments conducted in various simulated urban scenarios using the state-of-the-art CARLA simulator for autonomous driving. The autonomous driving task explored is following the lane. All models, architectures, and datasets mentioned are open-source [248].

6.2. Kinematic-infused and visual memory end-to-end control based on imitation learning

This section introduces the system developed, which consists of different deep learning architectures (see Fig.6.1), some of them with inner memory capabilities, trained using imitation learning for a lane-follow problem and with a range of sensory input data. We have used 8 deep learning architectures based on an end-to-end approach as shown in Fig.6.1. They use as input at least a sensory image and generate motor commands for an ego vehicle in a reactive control loop.

We focus our contributions on the implications in the final behavior of the robotics system of the addition of visual memory and kinematic data to the models. The perception data used as input is a simplified processed data from the sensory data. Instead of directly using the frontal camera of the vehicle, for this work we used a bird-eye view of the scenario, removing part of the complexity that the system needs to perceive (shadows, weather, different textures...). The bird-eye view used is a segmented image including only the key components of the scene (see Fig 6.2) for an example. In this case, we only need the car position information and the lanes that surround the vehicle. The approaches are trained and tested in the variety of towns that CARLA includes.

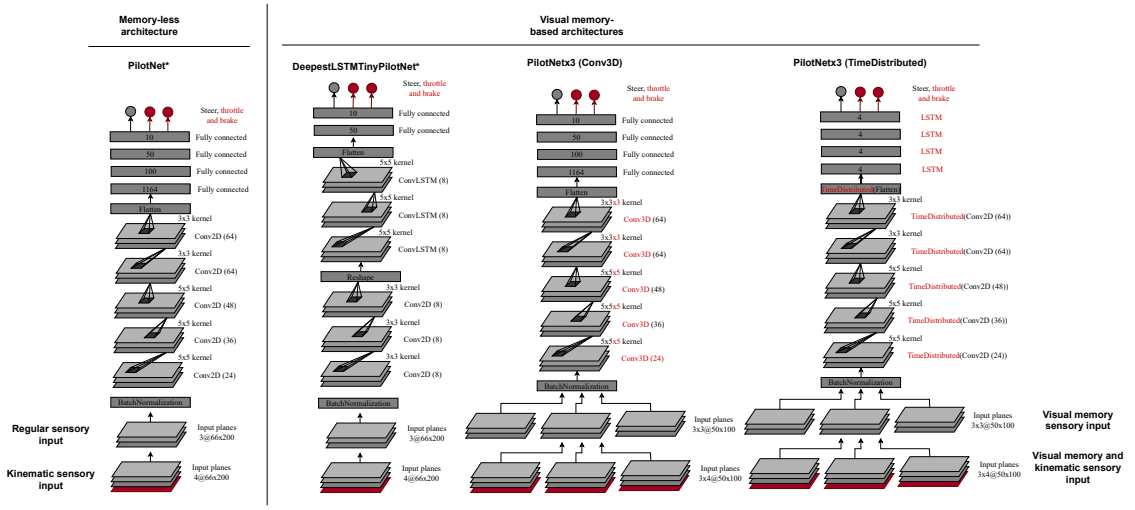


Figure 6.1: Details of the deep learning architectures compared in this work. One of them is memory-less and the other three are visual memory-based. A variation of each of them also receives kinematic data input. Layers and input data marked in red are the modifications proposed in this work based on baseline architectures.

An imitation learning approach is used for training the neural architectures. In this approach, an expert agent drives along the towns while the sensory information and the behavior are recorded (behavior cloning). The sensory data in this scenario includes the bird-eye view and the behavior includes normalized information about control commands (throttle, steer, and brake). Using the information extracted from the expert agent as supervised output for training, the final training agent should mimic the behavior of the former, if the dataset varies enough. To give the models an understanding of different situations, four urban scenarios were used for training the models: Town01, Town03, Town05, and Town07 (see Fig. 6.3 for a top-view of each scenario). These towns include different numbers of lanes, turn layouts, and road types like urban or highways. Town01 is used for testing the trained models, as explained in depth in Section 6.4. The expert agent used is the rule-based autopilot included in the simulator, which can drive in different urban scenarios and has access to privileged simulation data. We used only one vehicle model to maintain a similar visual structure and physical behavior when driving.

We have explored four different deep learning architectures: one without memory, and three with visual memory. To augment our investigation, we have introduced additional kinematic input data to the baseline models, resulting in a total of eight distinct models (see Fig6.1 for a detailed view of each architecture and group).

6.2.1. Memory-less deep learning architecture

In the first group, we consider the architectures whose input only includes the visual sensory information ad the current time, a single image, and no architectural modules that could be considered as memory, such as LSTM cells. We include here PilotNet*.

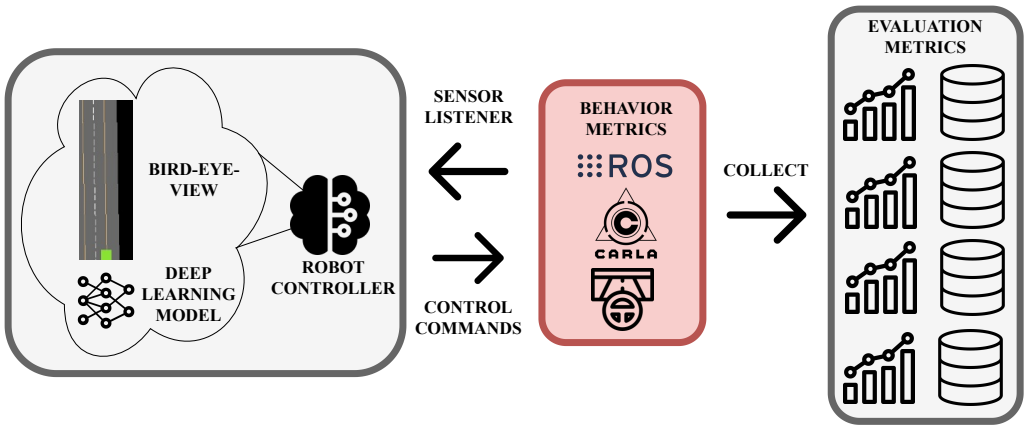
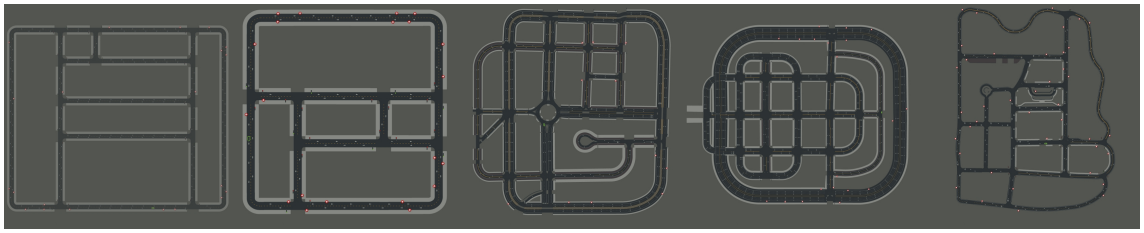


Figure 6.2: End-to-end autonomous driving pipeline using Behavior Metrics software and a robot controller based on a deep learning model that controls the vehicle based on its sensory data.



(a) Town01. (b) Town02. (c) Town03. (d) Town05. (e) Town07.

Figure 6.3: Set of urban environments in CARLA used.

PilotNet [159] was proposed in a previous work on end-to-end imitation learning for steering control. In this case, we have extended them to support throttling and braking (PilotNet*), considering that this information is also available from the expert agent. This architecture is specialized in understanding the context of an input sensory image and generating control commands for the ego vehicle.

- **PilotNet***: a powerful network that combines a convolutional backbone with some connected layers and an output of the control commands.

6.2.2. Deep learning architectures with visual memory

In the second group, we describe three architectures with visual memory. We include here DeepestLSTMTinyPilotNet* and two architectures especially created for this work. DeepestLSTMTinyPilotNet [213] is an architecture that was proposed in a previous work on end-to-end imitation learning for steering control where they only used one image as input and two architectures created for this work that are extensions of PilotNet*. Again, we have extended them to support throttling and braking (DeepestLSTMTinyPilotNet*), considering that this information is also available from the expert agent. The two architectures created for this work (PilotNet*x3 (Conv3D) and PilotNet*x3 (TimeDistributed))

receive visual sensory information from the current time instant and additionally, information from previous instants. We study whether this additional information helps vehicle control in some scenarios or situations. We explore two different variations for extracting knowledge from the visual data input, *Conv3D* and *TimeDistributed*.

- **DeepestLSTMTinyPilotNet*:** update of PilotNet architecture model making it smaller, reducing the number of convolutional and fully connected layers. It has some ConvLSTM layers that add some memory information. It only uses an image as input, instead of taking full advantage of the LSTM modules using several images as input.
- **PilotNet*x3 (Conv3D):** based on PilotNet*. In this variation, the convolutional part is replaced by a 3D convolutional backbone that extracts the temporal information and understands the content of the visual data. The second part of the architecture maintains the PilotNet* structure and again generates the collection of control commands for the vehicle.
- **PilotNet*x3 (TimeDistributed):** based on PilotNet*. In this variation, the model extracts information from the provided visual data from each image, combining the extracted features after the convolutional backbone and using LSTM modules for extracting the temporal information. The final fully connected layers included in PilotNet* are removed from this approach since the LSTM modules are enough for finally generating the control commands for the vehicle and understanding the global context.

6.2.3. Deep learning architectures with kinematic data as input

The previous four architectures have been also extended with an additional variation of sensory input using *kinematic data*. In this variation, the same architectures are used with a modification in the input sensory data, including information about the current ego vehicle velocity (kinematic data). This exploration is motivated by the widespread availability and easy access of this data in vehicles. We also consider the high safety standards needed for an autonomous driving system and recognize the pivotal significance assigned to the system's speed in this matter. The additional information is included as an additional channel to the visual data, without modifications to the described layers. For the architectures with visual memory that receive more than one image as input, the ego vehicle speed is added as a new channel in the images, including the current velocity at the specific instant of each frame.

6.2.4. Training

The training procedure varies slightly between the approaches considering their data structure, although we use the same amount of data for all the presented models. For

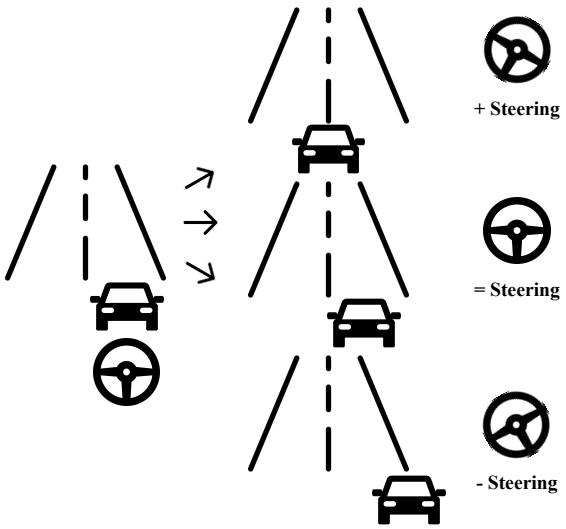


Figure 6.4: Affine image date augmentation example. From the training example in the left, new examples are generated modifying the steering command accordingly.

the architectures considered memory-less, all the data collected from the expert is divided and shuffled as in a common machine learning workflow. We collected data at a rate of 20 images per second, so (t, $t-5$, and $t-10$), which is the input for models that receive more than one frame, is half a second. For the memory-based architectures, we first generated mini-sequences using a sliding window of three data points with temporal relationships before shuffling.

A collection of image data augmentations is included in the training procedure. We include modifications of brightness, contrast, gamma channel, hue saturation, PCA color augmentation, gaussian blur, and horizontal affine transformations. This augmentation is conducted using Albumentations [176]. We found horizontal affine augmentations to be important in the final behavior of the model and its generalization. Using imitation learning, the model can learn the behavior displayed in the dataset but struggles when the test data differs even a little bit from the training data distribution, as empirically proved in previous works [174]. With horizontal affine augmentation, the model can generalize better and provide good behavior in test scenarios that are slightly different from the rest of the data distribution. For example when approaching a turn a few centimeters away from the center of the lane (see Fig. 6.4). We use mean squared error as the loss function during training.

6.3. Measuring end-to-end imitation learning for robot control

For measuring the quality of robot behavior in autonomous driving, the common metrics used in machine learning are not enough to understand whether a model behavior is proficient or not. Typically, MSE or MAE are used for calculating the loss of the model during training and are commonly used. They are good indicators, but not enough to assess robot

application quality as they only measure the similarity of the model output and the supervised output at each instant. They do not take into account the future effect of control decisions, so a deep learning model with low error loss may result in poor robot behavior. In an end-to-end control system, previous decisions and current vehicle situations play an important role in the next evolution of the situation. A previous inadequate decision a few seconds ago could lead to a very difficult situation now. For assessment of such a control system, the external global holistic metrics do take into account such effects and others and provide a more reliable indicator of the quality of the robot behavior. They are typically application-specific.

In addition to the common metrics, we use for the experimental validation Behavior Metrics, a software tool that is a contribution of this thesis and that has been introduced in Chapter 5.

6.4. Experimental validation

In this section, we present a series of experiments for the validation of the models and understanding of the implications of memory in the behavior of the robot in different situations. As described in previous sections, we have four different deep learning models trained on imitation learning with two variations for each one. We use Behavior Metrics with CARLA as software for the experimental validation of the modules, along with the typical loss metrics discussed previously.

All the different experiments are easily reproducible, with the model's weights, architectures, and Behavior Metrics available open-source[246]. TensorFlow has been used for programming and training the different deep learning architectures. 2 Nvidia GeForce RTX 3090 GPUs were used as hardware when running the experiments.

The number of control decisions per timestamp is important in this type of robotic scenario but we do not study its implications experimentally in the present work as we already study it in Chapter 7. We consider a scenario where the number of iterations of the controller is high enough for the correct control of the vehicle.

We provide six experiments, where the eight models are evaluated to understand their differences. The models are tested using a never-seen scenario (Town02). In addition, we explore a lane-following scenario with no other vehicles of obstacles involved. Traffic lights and signals are also ignored in these experiments since they remain out of the scope of this study, the implications of including memory in the robot control. in the case of an intersection, the vehicle learns to follow a policy of going straight through it, based on the dataset provided by the expert agent and the imitation learning policy. The starting position is random among a set of points for each town, considering that the vehicle can drive lane-following for at least a few hundred meters without interfering with a situation that differs widely from the training dataset. We do not provide experimental results comparison with the original PilotNet or DeepestLSTMTinyPirloNet architectures from

Model	Visual memory	Kinematic input	MAE test	MSE test
PilotNet*	✘	✘	0.0507	0.0177
PilotNet*	✘	✔	0.0332	0.0086
DeepestLSTMTinyPilotNet*	✔	✘	0.0662	0.0196
DeepestLSTMTinyPilotNet*	✔	✔	0.0456	0.0094
PilotNetx3* (Conv3D)	✔	✘	0.0295	0.0082
PilotNetx3* (Conv3D)	✔	✔	0.0074	0.0079
PilotNetx3* (TimeDistributed)	✔	✘	0.0289	0.0077
PilotNetx3* (TimeDistributed)	✔	✔	0.0069	0.0086

Table 6.1: MAE and MSE metrics comparison for each trained model using test data from the dataset. Four different architectures are tested with different input data considerations: bird-eye view (BEV) and velocity sensory data. ✔: supported. ✘: unsupported.

the original papers since our architectures feature distinct output configurations, as elaborated previously in Section 6.2.2. Conducting a fair comparison is not feasible due to these inherent differences.

6.4.1. Comparison of models using common ML metrics

In Table 6.1, the best values obtained for MAE and MSE in the test set of the supervised dataset for each of the models are displayed. These values are better for the models whose input is kinematic sensory data, with an 86% reduction in MAE and 53% in MSE. This suggests that models with kinematic data are better at imitating the supervised datasets. They adeptly encapsulate the correlation between inputs and outputs.

These results are relevant indicators, but not enough for a reliable comparison of the models' final control behavior due to the high variety of situations that the vehicle could encounter at simulation test time and other variables that are involved in its quality such as the number of control commands per second that the model can generate. Inferring a single non-ideal control command to the robot actuators in a key moment may have worse consequences in robot performance than many non-ideal commands in harmless moments. To understand the complete behavior of the robot controller, a comparison in simulation should be conducted. Even so, we include this comparison with the supervised dataset since it is standard in machine learning research although for robotics applications these metrics are not conclusive to indicate good performance, or good robot behavior. We need further experimental validation for the validation of the models, which is conducted in the following subsections.

6.4.2. Behavior in test scenario with top speed regulation

In this experiment, all the models are evaluated in the test scenario (*Town02*), as a lane-following vehicle. Each experiment is conducted five times starting from a random position in the test scenario. This is the most simple and common scenario, where the vehicle follows a lane using a reactive controller which calls the inference of a neural model on each iteration to generate the motor commands. The expert agent used for recording the supervised samples included in the training dataset has a top speed of 30 km/h. In this experiment, the models without kinematic sensory data also include a limit of 30 km/h to make a fair comparison, the vehicle speed is truncated when this limit is reached. In Section 6.4.3 the comparison without this top speed limitation is also studied.

In Table 6.2, we can see the results for the different models (columns) and the measured metrics (rows) in a test circuit, *Town02*. The *Successful experiments* metric is the most informative, it represents the number of experiments completed by the vehicle without collisions and without exceeding the maximum speed (30 km/h) out of the five runs. We also require that the agent reaches the average speed of the expert agent which is between 25 and 30 km/h to be considered successful.

The differences in this point are small. All the architectures can complete the experiments successfully, without collisions, and with an adequate average speed. The *Effective completed distance* is similar for all of them and the *Positional deviation mean per km* from the center of the lane is low. The *Vehicle jerk* for both control commands and velocity is also low, which translates to smooth and safe driving. The *Controller iterations frequency* is also adequate for the experiment. Although some of the models experiment some lane invasions, they are not problematic since the numbers are low and they do not cause collisions.

6.4.3. Studying the model without top speed limitation

In the previous experiments, some robot controllers needed a maximum speed limit like the one provided by the expert agent (30 km/h) to drive correctly. In this new experiment, we explore how removing that top boundary affects the behavior of the models. The rest of the experimental setup remains unchanged.

In Table 6.3, we can see the results of this experiment on *Town02*. We exclusively account for experiments without collisions in the table for all metrics, except for *Collisions per km* and *Lane invasions per km*. We can observe that models without memory or with only visual memory capabilities are more prone to collisions when the speed limit is not controlled. Looking at their maximum speed or average speed, we can understand that they are not able to learn how to maintain a safe speed (30 km/h), which leads to failure in all the experiments. For the models with kinematic sensory data input, the results are the same as in the experiment in Section 6.4.2, since the robot controllers using these models were already able to drive without a top speed limit. The interpretation of these results is

Map	**Town02**							
Model	**Pilotnet***		**DeepestLSTMTinyPilotNet***		**Pilotnetx3* (Conv3D)**		**Pilotnetx3* (TimeDistributed)**	
Visual memory	✘	✘	✔	✔	✔	✔	✔	✔
Kinematic input	✘	✔	✘	✔	✘	✔	✘	✔
Effective completed distance (m)	820.6	868.9	830.6	846.6	902.6	875.2	889.0	852.3
Position deviation mean per km (m/km)	0.25	0.26	0.22	0.33	0.24	0.25	0.29	0.29
Controller iterations frequency (Hz)	18.32	18.40	18.25	18.05	17.10	17.05	17.65	17.51
Vehicle jerk in control commands per kilometer	0.31	0.19	0.16	0.15	0.18	0.12	0.19	0.12
Vehicle jerk in velocity per kilometer	0.34	0.33	0.30	0.31	0.33	0.49	0.34	0.51
Average speed (km/h)	24.71	26.78	25.01	26.41	27.51	26.77	27.15	26.21
Max speed (km/h)	31.35	30.08	31.30	29.92	31.57	31.25	31.50	30.98
Experiments with collisions	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
Collisions per km	0	0	0	0	0	0	0	0
Lane invasions per km	0	0.46	0	3.08	0	0.46	0	0
Successful experiments	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**	**5/5**

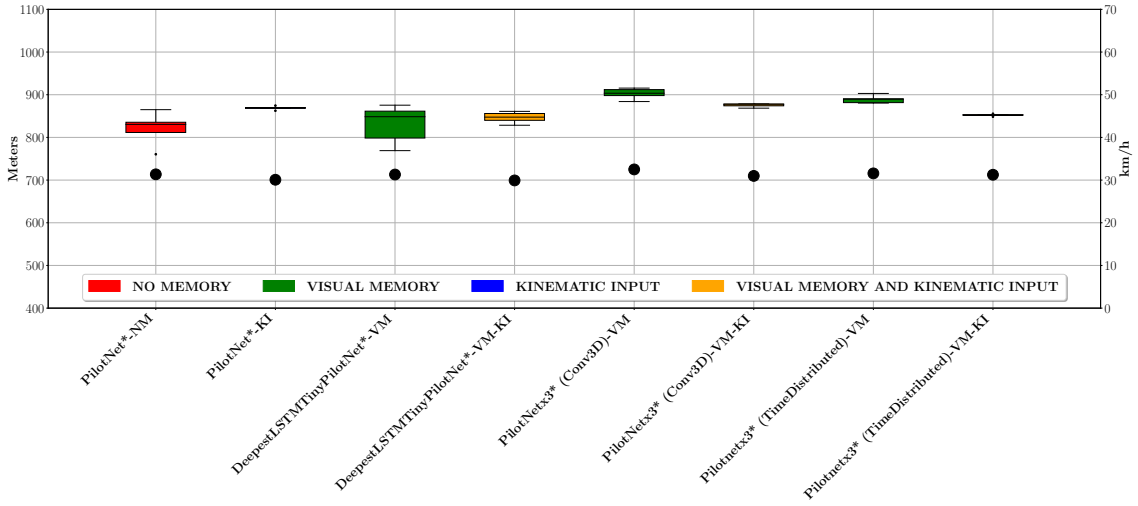
Table 6.2: Comparison of models (columns) in different test environments considering some measured metrics (rows) provided by Behavior Metrics. Values in **bold** highlight the most interesting results. ✔: supported. ✘: unsupported.

Map	**Town02**							
Model	**Pilotnet***		**DeepestLSTMTinyPilotNet***		**Pilotnetx3* (Conv3D)**		**Pilotnetx3* (TimeDistributed)**	
Visual memory	✘	✘	✔	✔	✔	✔	✔	✔
Kinematic input	✘	✔	✘	✔	✘	✔	✘	✔
Effective completed distance (m)	**946.3**	868.9	**962.0**	846.6	**994.8**	875.2	**1083.0**	852.3
Position deviation mean per km (m/km)	**0.23**	0.26	**0.21**	0.33	**0.28**	0.25	**0.28**	0.29
Controller iterations frequency (Hz)	**17.34**	18.40	**16.11**	18.05	**17.09**	17.05	**17.65**	17.51
Vehicle jerk in control commands per kilometer	**0.28**	0.19	**0.15**	0.15	**0.18**	0.13	**0.17**	0.12
Vehicle jerk in velocity per kilometer	**0.26**	0.33	**0.22**	0.31	**0.25**	0.49	**0.21**	0.51
Average speed (km/h)	**30.37**	26.78	**30.13**	26.41	**31.72**	26.77	**38.82**	26.21
Max speed (km/h)	**53.20**	30.08	**47.93**	29.92	**50.15**	31.25	**59.18**	30.98
Experiments with collisions	**5/5**	0/5	**1/5**	0/5	**3/5**	0/5	**4/5**	0/5
Collisions per km	**2.09**	0.0	**0.32**	0.0	**0.95**	0.0	**1.45**	0.0
Lane invasions per km	**2.24**	0.46	**0.71**	3.08	**0.87**	0.46	**1.05**	0.0
Successful experiments	**0/5**	**5/5**	**0/5**	**5/5**	**0/5**	**5/5**	**0/5**	**5/5**

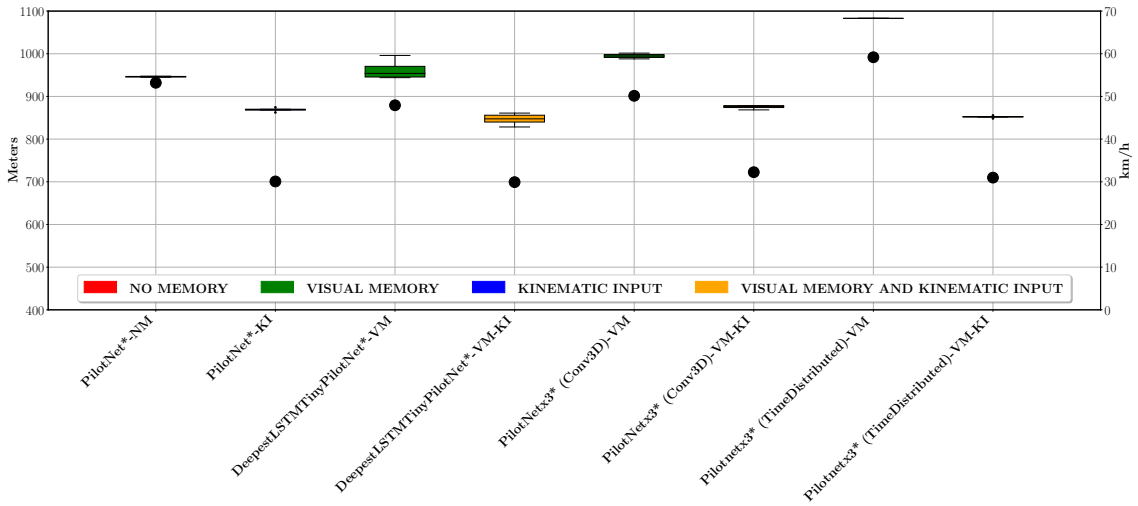
Table 6.3: Comparison of models in different test environments without top speed limit considering metrics from Behavior Metrics. **Bold** values (excluding *Successful experiments*) indicate changes in results from previous experiment results. Values in **red bold** and **bold** for *Successful experiments* highlight the most interesting results. ✔: supported. ✘: unsupported.

that the kinematic data input is key for the robot to understand its state precisely and must be included in the model as input for proficient behavior.

In Fig. 6.5, we provide the detail of *Effective completed distance* and *Max speed* metrics for each model for the case with and without top speed restriction. We can see that models with visual memory and kinematic input can traverse a bit longer effective distances maintaining a safe top speed. We can also see that the standard deviation is small and the number of atypical values is very low. The models' behavior is always similar. The black dots represent the mean maximum speed in the experiments. It remains similar for the cases where the top speed is controlled (top graph) whereas it generates extremely high value for models without kinematic input when it is not controlled (bottom graph).



(a) Effective distance completed with 30 km/h restriction.



(b) Effective distance completed without 30 km/h restriction.

Figure 6.5: Effective distance completed with 30 km/h restriction (top) and without (bottom). The right y-axis shows the vehicle's maximum speed (represented using black dots). NM: no memory. VM: visual memory. KI: kinematic input.

6.4.4. Taking the control of a fast-moving car

In this experiment, the robot controller with the deep learning model is suddenly connected to a vehicle that is already running at high speed, in this case, 50 km/h and 70 km/h. Once the vehicle reaches that speed, the model starts generating control commands for the car (throttle, steer, and brake) and we test whether the model can control that unseen extreme situation or not. In this situation, the vehicle should react fast to regain control over the car. This situation falls outside of the training dataset distribution.

In Table 6.4, the results of the experiment are shown. We only consider architectures with kinematic data as input, since we have already proved it to be necessary for proficient control. We can see that the only models able to gain back control and reduce the speed are the ones with visual memory and kinematic input for the 50 km/h case and only one of them for the 70 km/h case. This is a clear sign of the advantages of adding both types of features. In certain scenarios, both visual memory and kinematic input data help take back control of the car. In the experiments, the vehicle reduces the speed to the learned behaviors (less than 30 km/h) and then starts driving as usual, as shown in Fig. 6.6. We can also see that a model with both memory types can take back the control, reducing the speed to the known point (30 km/h). On the contrary, for the model without memory, the ego vehicle continues driving at this top speed and control is not possible, quickly causing a collision. For the extreme case of 70 km/h, we can see that only PilotNetx3*(Conv3D) can control the car and reduce the speed to a known state. This could be attributed to the enhanced memorization capabilities that Conv3D provides.

6.4.5. Robustness to sensory manipulation

The next experiment tests the robustness of the models to a series of perturbations in the sensory data. In this case, we imagine a case where input data suffers some alterations, as it could occur in a real-world situation, and test how the memory helps in the control problem. Since the common input data for all the models is the visual data, we alter that information and study its implications considering the memory capabilities. In this case, we drop out randomly some parts of each of the visual data used as input. We again only consider architectures with kinematic data input since we have already proved it to be necessary.

In Table 6.5, the results of these experiments are displayed for *Town02* for a percentage of dropout of 50% and 90% (see Fig. 6.7 for an example). We again only consider *Succesful experiments* those without collisions and with an average speed close to the one provided by the expert agent. We can see that with a percentage of 50% of dropout, only two models are successful, PilotNet* and PilotNetx3*(TimeDistributed). The explanation for these results could be that DeepestLSTMTinyPilotNet* and PilotNetx3*(Conv3D) are reliant on the visual data that they receive (the first one includes*ConvLSTM layers* and the second one *Conv3D layers*) whereas the successful models while relying on visual data,

Map	**Town02**			
Model	**Pilotnet***	**DeepestLSTMTinyPilotNet***	**Pilotnetx3* (Conv3D)**	**Pilotnetx3* (TimeDistributed)**
Visual memory	✘	✔	✔	✔
Kinematic input	✔	✔	✔	✔
Speed	**50 km/h**			
Experiments with collisions	5/5	0/5	0/5	0/5
Average speed	-	27.18	27.07	26.82
Collisions per km	46.51	0.0	0.0	0.0
Successful experiments	0/5	**5/5**	**5/5**	**5/5**
Speed	**70 km/h**			
Experiments with collisions	5/5	5/5	0/5	5/5
Average speed	-	-	29.92	-
Collisions per km	27.25	29.95	0.0	26.95
Successful experiments	0/5	0/5	**5/5**	0/5

Table 6.4: Comparison of models in a high-speed scenario where the model takes control when the ego vehicle is already at a speed of 70 km/h. For the *Average speed*, we only consider experiments without collisions. This experiment is tested in Town02. Values in **bold** highlight the most interesting results. ✔: supported. ✘: unsupported.

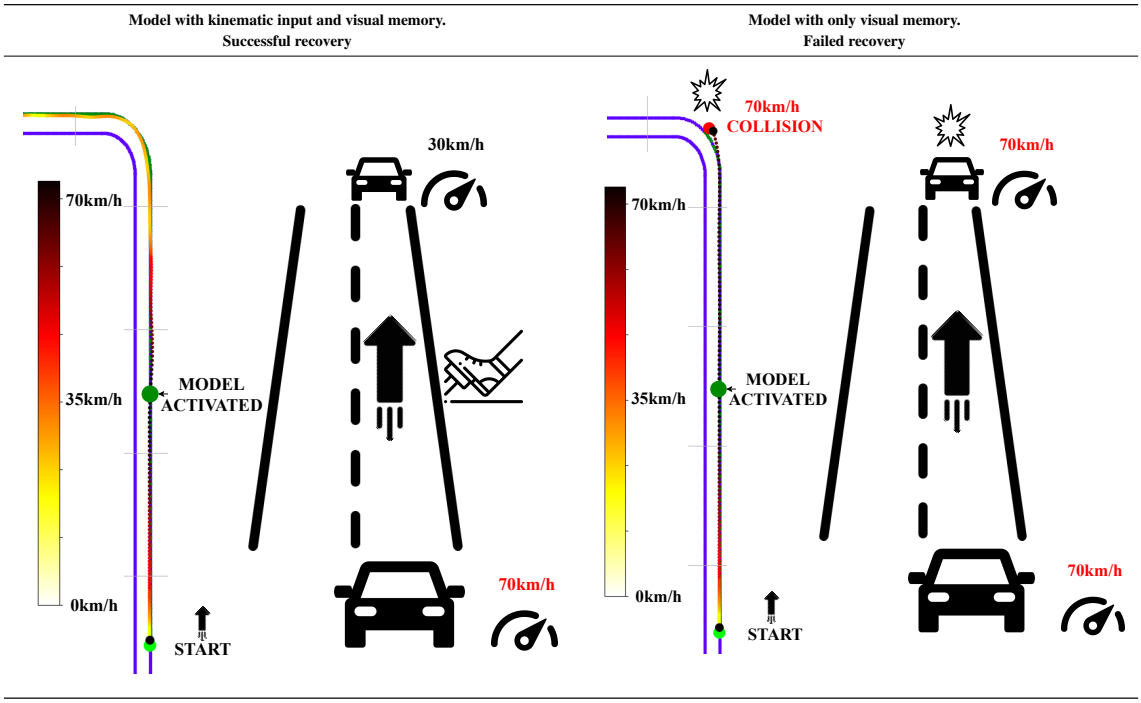


Figure 6.6: Example of activating the vehicle autonomous driving system at a high-speed situation. On the left, the model with visual memory and kinematic input can restore the speed to the known point and continue driving. On the right, the model with only visual memory is not able to restore the speed and it collides due to the high speed.

they are more prone to consider kinematic input. For the extreme case of 90%, only PilotNetx3*(TimeDistributed) is still successful, which can be attributed to the visual memory capabilities. Each model with visual memory includes different ways of introducing it, and some of them are more important in certain extreme scenarios such as this one.

6.4.6. Visual memory length and density comparison

In this experiment, we evaluate the model's memory capabilities in terms of the length and density of the visual input data. Memory length refers to the amount of information that the model receives. In this case, we evaluate the implications of adding more visual input data. In Section 6.2.4, we describe that the visual memory used for training the models is (t, $t-5$, and $t-10$) considering that the sensors and controller run using 20 frames per second. Considering that the vehicle receives 20 frames per second, the default visual memory used is 0.5 seconds long. In this experiment, we evaluate other possible visual lengths (5 and 9 frames). 5 frames is 1 second of memory and 9 frames is 2 seconds if we consider that the frame rate is the same. Similarly, memory density refers to the time gap between frames. For the default configuration, we use (t, $t-5$, and $t-10$). For the experiments, we test (t, $t-1$, and $t-2$) , (t, $t-10$, and $t-20$) and (t, $t-20$, and $t-40$). The experiments are conducted using the models with visual memory that receive several frames and considering the top speed boundary (PilotNetx3*(Conv3D)

Map	**Town02**			
Model	**Pilotnet***	**DeepestLSTMTinyPilotNet***	**Pilotnetx3* (Conv3D)**	**Pilotnetx3* (TimeDistributed)**
Visual memory	✘	✔	✔	✔
Kinematic input	✔	✔	✔	✔
Percentage	**50%**			
Experiments with collisions	0/5	1/5	5/5	0/5
Average speed	26.12	22.54	-	25.96
Collisions per km	0.0	1.52	72.75	0.0
Successful experiments	**5/5**	0/5	0/5	**5/5**
Percentage	**90%**			
Experiments with collisions	5/5	0/5	5/5	0/5
Average speed	-	5.19	-	25.18
Collisions per km	23.68	0.0	12.31	0.0
Successful experiments	0/5	0/5	0/5	**5/5**

Table 6.5: Comparison of model performance modifying the input sensory information. For the *Average speed*, we only consider experiments without collisions. Values in **bold** highlight the most interesting results. ✔: supported. ✘: unsupported.

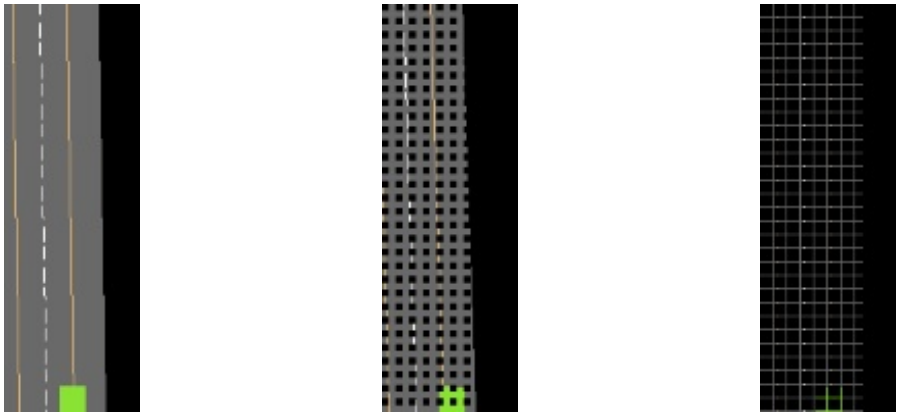


Figure 6.7: Example of input data: normal (left), broken 50% (middle) and broken 90% (right).

Map	**Town02**					
Model	**Pilotnetx3* (Conv3D)**			**Pilotnetx3* (TimeDistributed)**		
Visual memory	✔	✔	✔	✔	✔	✔
Kinematic input	✘	✘	✘	✘	✘	✘
Memory length (frames)	3	5	9	3	5	9
Collisions	0	0	1.0	0	0.8	1.0
Average speed	25.18	25.64	-	26.08	26.65	-
Position deviation mean per km	1.13	1.75	-	1.12	2.02	-
Collisions per km	0.0	0.0	1.30	0.0	0.0	6.25
Successful experiments	**5/5**	**5/5**	0/5	**5/5**	1/5	0/5

Table 6.6: Comparison of model performance with different visual memory lengths. For the *Average speed* and *Position deviation mean per km*, we only consider experiments without collisions. Values in **bold** highlight the most interesting results. ✔: supported. ✘: unsupported.

and PilotNetx3*(TimeDistributed)).

In Table 6.6, we present the results for the different proposed memory lengths. We can see that adding extra frames does not help for these models and they start failing when adding them. They are more prone to collisions and if we look at the *Position deviation mean per km*, we can see that adding more frames leads to more position deviation. Since the models are simple and based on PilotNet, we can attribute these results to models that are simple and that can not understand a lot of frames. Possibly, by modifying the architectures further, they would be capable of understanding a broader range of frames, but for this scenario, they are not needed.

In Table 6.7, we present the results for the different studied memory densities. A memory with a small density (t, $t-1$, and $t-2$) can drive successfully and we can see that when the space between frames is widened, the models are more prone to collisions and its *Position deviation mean per km* is worse. Similar behavior is observed for the *Position deviation mean per km* when the space between frames is too little (t, $t-1$, and $t-2$). The best option for these models and scenarios is (t, $t-5$, and $t-10$).

Finally, after the six presented experiments, a summary of the results obtained in the experiments is displayed in Table 6.8. We have proved that adding kinematic input data and visual memory improves the general behavior and robustness of the deep learning

Map	**Town02**			
Model	**Pilotnetx3* (Conv3D)**			
Visual memory	✔	✔	✔	✔
Kinematic input	✘	✘	✘	✘
Memory densities (frames)	$t, t-1, t-2$	$t, t-5, t-10$	$t, t-10, t-20$	$t, t-20, t-40$
Collisions	0.0	0.0	0.4	0.6
Average speed	24.96	25.18	26.59	26.07
Positions deviation mean per km (m/km)	1.39	1.13	1.35	1.89
Collisions per km	0.0	0.0	5.13	5.12
Successful experiments	**5/5**	**5/5**	3/5	2/5
Model	**Pilotnetx3* (TimeDistributed)**			
Visual memory	✔	✔	✔	✔
Kinematic input	✘	✘	✘	✘
Memory densities (frames)	$t, t-1, t-2$	$t, t-5, t-10$	$t, t-10, t-20$	$t, t-20, t-40$
Collisions	0.0	0.0	0.2	0.2
Average speed	25.94	26.08	26.63	26.01
Positions deviation mean per km (m/km)	1.12	1.12	1.47	1.59
Collisions per km	0.0	0.0	2.97	3.01
Successful experiments	**5/5**	**5/5**	4/5	4/5

Table 6.7: Comparison of model performance with different visual memory densities. For the *Average speed* and *Position deviation mean per km*, we only consider experiments without collisions. Values in **bold** highlight the most interesting results. ✔: supported. ✘: unsupported.

Experiment \ Type	Visual	Visual memory	Kinematic input	Visual memory and kinematic input
Regular lane-follow (Section 6.4.2)	✔	✔	✔	✔
Experiment without top speed (Section 6.4.3)	✘	✘	✔	✔
High speed experiment (Section 6.4.4)	✘	✘	✘	✔
Sensory Robustness experiments (Section 6.4.5)	✘	✘	✘	✔

Table 6.8: Comparison summary of model performance across presented experiments. The addition of at least kinematic input data improves the final behavior and adding both types generates gains in certain scenarios. ✔: successful. ✘: failure.

model in the end-to-end control of an autonomous car for a lane-follow application. The most robust models have been obtained by combining both visual memory and kinematic input data.

6.5. Conclusions

In this chapter, we have presented one of the contributions of the thesis, where we present and study four different deep learning architectures and a variation of each one for end-to-end robot control based on imitation learning for an autonomous driving problem. We have studied and proved how adding visual memory and kinematic input data to the models enhances the quality of the final control behavior for following the lane. These architectures are PilotNet*, DeepestLSTMTinyPilotNet*, PilotNetx3* (Conv3D), and PilotNetx3* (TimeDistributed), with their variation with also kinematic input. Specifically, we have studied how adding visual memory and kinematic input data to the models improves the performance in certain situations, such as taking control of a fast-moving car in high-speed scenarios never seen before or self-regulating the vehicle speed correctly. The models have been tested extensively in various simulated urban scenarios with various layout designs, proving the research hypothesis widely.

When adding kinematic input data, the system controls the speed better. When adding both visual memory and kinematic input data, the vehicle can drive in even more situations and is more robust to sensor failure or controlling never-seen situations like high-speed experiments. We have proved that adding at least kinematic sensory data can help in the final system compared to the situation where only instant visual perception is used, which

leads to a less complete context understanding.

We have also studied different visual memory lengths and densities for extracting insight into how it affects the control system even for these simple deep learning architectures. We have proved that a lower density of frames can cause failed experiments when using simple architectures and that the length of the memory generates a comparable influence, causing failures when adding an excessive number of frames.

After this exploration conducted in this chapter, in the following chapter, we explore the optimization of the deep learning models for end-to-end robot control for autonomous driving using state-of-the-art techniques.

Chapter 7

Optimization of end-to-end autonomous driving control

In this chapter, which is another contribution of this thesis, we explore and compare a variety of alternatives for model optimization to solve the visual lane-follow application in urban scenarios with an imitation learning approach. This contribution has been published as a journal article and also as a conference paper [244]. The optimization techniques include quantization, pruning, fine-tuning (retraining), and clustering, covering all the options available in the most common deep learning frameworks. The optimizations provided by TensorRT which are specific to the hardware are also explored. For the comparison, following the same line as in previous chapters, we use offline metrics such as mean squared error and inference time and additionally, we evaluate them in an online setup using Behavior Metrics with CARLA, the software tool introduced in Chapter 5.

7.1. Introduction

In autonomous driving, vision-based solutions are usually generated using deep learning models, which are high-demanding computational solutions. An important component with them is the available computing hardware as the performance of robot applications depends not only on the quality of the control decisions but also on their frequency, the higher the better. Some autonomous vehicles or robots are equipped with high-performance hardware, others are not. Updating to faster-computing hardware is beneficial but it is not always a real option. A possible solution is to optimize the deep learning model with different techniques [217].

These optimization techniques have already been discussed in Chapter 2. In this chapter, we introduce and discuss several common model optimization techniques for end-to-end autonomous driving control, in particular for the visual lane-follow application. We apply and compare them to a baseline model to understand how the optimizations impact the system performance and efficiency using an imitation learning approach [175].

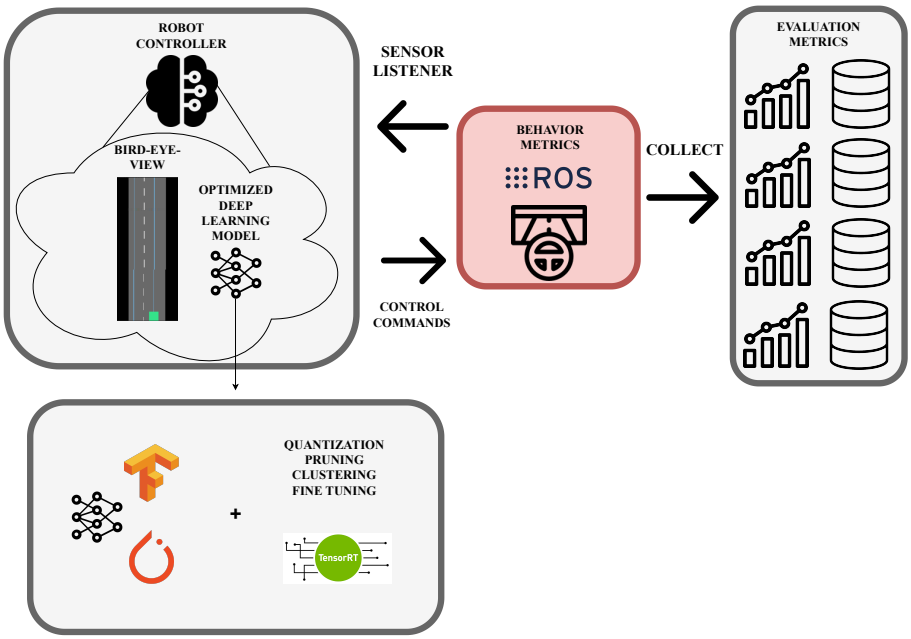


Figure 7.1: End-to-end autonomous driving pipeline using Behavior Metrics software and a robot controller based on a deep learning model that drives the vehicle based on its sensory data.

We generate a new supervised dataset from expert data because the already available ones are not directly suitable for the lane-follow application. Some studies have already addressed similar questions [223] but they do not consider deep learning models as possible controllers, which is our focus and part of the innovation. Our study includes two well-known deep learning frameworks (TensorFlow-Keras and PyTorch) with their optimizations toolkits (TensorFlow Model Optimization Toolkit [221]) and the optimizations provided by Nvidia's TensorRT framework [222] for this particular hardware. The research hypothesis is that by including optimization techniques, configuring and adapting them accordingly to the deep learning model and the problem setup, we can obtain similar quality levels of autonomous driving with smaller and faster models as compared to the baseline model. We leave outside this study the finding of the best expert agent (we use a good enough one) for driving and the comparison between classic and neural control, since the focus is on the implications of the deep learning model optimizations alone. We provide a series of experiments that validate this hypothesis, including both offline and online model comparisons in test scenarios using the state-of-the-art CARLA simulator for autonomous driving. The online experimental validation is conducted using Behavior Metrics (see Fig. 7.1). We provide all models, architectures, modified software, and datasets as open-source [249].

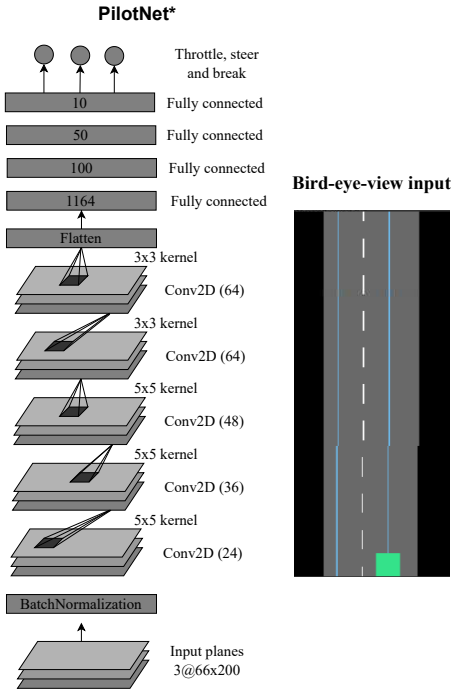


Figure 7.2: PilotNet* architecture detail (left) and bird-eye view input example (right).

7.2. Optimizing end-to-end imitation learning models for lane-follow robot control

This section outlines the baseline deep learning model for lane-follow robot control using imitation learning. It incorporates state-of-the-art optimization techniques prevalent in deep learning model development and details the training process.

7.2.1. Baseline architecture

The baseline deep learning architecture is based on PilotNet end-to-end model [159], replicating the architecture provided in the paper in both TensorFlow and PyTorch frameworks. The model used is PilotNet*, the same architecture that has already been introduced in Chapter 6. PilotNet* provides three control commands as outputs instead of the one of PilotNet. Instead of only generating steering commands, as in the original work, PilotNet* generates throttle, steering, and brake. Our architecture modifies the baseline one only on the final part of the former model. In Fig. 7.2, a detailed diagram of the modified architecture is provided.

As in the architecture presented in the previous Chapter 6, the perception data used as input to the deep learning model is the bird-eye view of the vehicle in the scenario (see Fig. 7.2 for an example). This bird-eye view simplifies the perception task that the model has to conduct. Instead of having the immense entropy of a front-camera image, from which the model has to extract relevant features, the bird-eye view is a segmented image that includes only the pertinent classes for this particular problem (vehicles, pavement, road lanes...), reducing the complexity. Even though the perception task is simplified,

the results and conclusions presented here are general, and applicable for more complex perception setups (frontal camera vision) following the same ideas. They are simplified only for the sake of the focus on model optimization.

7.2.2. Dataset and training

Following the imitation learning approach, the dataset used for training the deep learning model is extracted from CARLA where an expert agent drives a car traveling through the scenario (Town 01) keeping the lane. At the same time, the control decisions and the bird-eye images are recorded. The raw dataset, which is the same we utilized in Chapter 6 is imbalanced since the majority of cases involved in autonomous driving are usually driving straight forward and the amount of other cases, such as turn situations, remains small. To reduce this bias, some techniques are already common, such as DAgger [174]. In this case, we record turns more times to oversample the dataset with such cases. In addition, we also use common machine learning training techniques like shuffling, normalization, and data augmentation (modifications of brightness, contrast, gamma channel, hue saturation, PCA color augmentation, gaussian blur, and horizontal affine transformations). Horizontal affine transformation is really important for this particular problem. For this transformation, the input visual data is displaced some points horizontally, generating more examples that could be found online by the vehicle. Considering this transformation, the output is also altered accordingly.

The optimization techniques used are (for a detailed diagram of some of them see Fig. 7.3):

- **Quantization:** approximation of neural network inner values that use floating-point numbers to low bit width numbers. This optimization usually changes the floating-point numbers to float16 or even int8 precision.
- **Pruning:** technique based on removing unnecessary connections or parameters of the deep learning architecture for reducing the size of a neural network.
- **Fine-tuning (retraining):** some techniques also involve a few fine-tuning steps for generating the optimized model. This process of fine-tuning is combined with the rest of the optimizations to generate more precise final results.
- **Clustering:** group similar weights in a neural network. Similar to pruning but in this case the weights are combined and represented with a single centroid value.

All these techniques improve the model efficiency, accelerating computations and reducing the model size. A reduction in model size can be critical in computation systems with sharp memory constraints. The acceleration of computations is also critical for systems with low computational capacity. From these base techniques, each deep learning framework has support for some of them and their combinations, as detailed in Table 7.1. Applying each optimization requires tuning for the particular problem and model.

Optimization type	Framework	Quantization	Pruning	Fine-tuning	Clustering
TensorFlow					
Baseline (No optimizations)	TensorFlow	✘	✘	✘	✘
TensorFlow Model Optimization Toolkit (TF Lite)					
Baseline in TF Lite (No optimizations)	TF Lite	✘	✘	✘	✘
Dynamic Rang Quantization	TF Lite	Int8+Float16	✘	✘	✘
Integer Quantization	TF Lite	Int8	✘	✘	✘
Integer (float fallback) Quantization	TF Lite	Int8+Float16	✘	✘	✘
Float16 Quantization	TF Lite	Float16	✘	✘	✘
Quantization Aware Training	TF Lite	Int8	✘	✔	✘
(Random sparse) Weight pruning	TF Lite	✘	✔	✘	✘
(Random sparse) Weight pruning quantization	TF Lite	Int8	✔	✘	✘
Cluster preserving quantization aware	TF Lite	Int8	✘	✔	✔
Pruning preserving quantization aware	TF Lite	Int8	✔	✔	✘
Sparsity and cluster preserving quantization aware training (PCQAT)	TF Lite	Int8	✔	✔	✔
TensorFlow TensorRT					
(TensorRT) Float32 Quantization	TF TensorRT	Float32	✘	✔	✘
(TensorRT) Float16 Quantization	TF TensorRT	Float16	✘	✔	✘
(TensorRT) Int8 Quantization	TF TensorRT	Int8	✘	✔	✘
PyTorch					
Baseline (No optimizations)	PyTorch	✘	✘	✘	✘
PyTorch Model Optimization Toolkit					
Dynamic Range Quantization	PyTorch	Int8+Float16	✘	✘	✘
Static Quantization	PyTorch	Int8	✘	✘	✘
Quantization Aware Training	PyTorch	Int8	✘	✔	✘
Local Prune	PyTorch	✘	✔	✔	✘
Global Prune	PyTorch	✘	✔	✔	✘
Prune+Quantization	PyTorch	Int8	✔	✔	✘
PyTorch TensorRT					
(TensorRT) Float32 Quantization	PyTorch TensorRT	Float32	✘	✔	✘
(TensorRT) Float16 Quantization	PyTorch TensorRT	Float16	✘	✔	✘
(TensorRT) Int8 Quantization	PyTorch TensorRT	Int8	✘	✔	✘

Table 7.1: Summary of optimization configurations and their supported techniques, including the development framework. ✔: supported. ✘: unsupported.

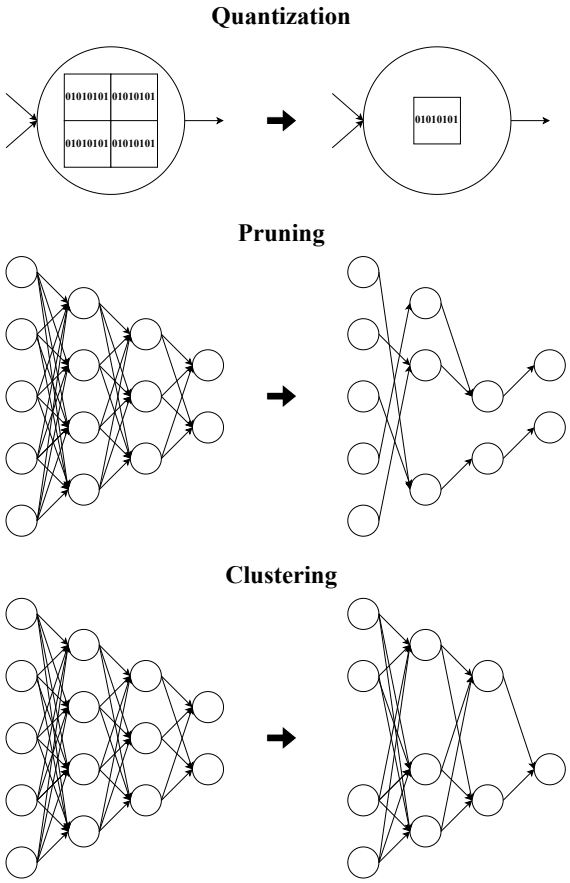


Figure 7.3: Optimization techniques diagrams.

7.3. Experiments

This section presents the experiments conducted for the evaluation of the differences between non-optimized and optimized deep learning models. The baseline model, PilotNet*, has been implemented in two of the most popular deep learning frameworks, TensorFlow and PyTorch, to also understand their differences. With these two models, we applied a series of framework-level optimizations and their combinations, tuning each of them appropriately to gain insight into each optimization's advantages. In addition, hardware-level optimizations were also explored using the Nvidia TensorRT framework that enhances model performance for Nvidia GPUs. We do not include a comparison with prior baseline methods since to the best of our knowledge this is the first work that compares different optimization techniques for end-to-end control in autonomous driving based on visual perception.

The hardware used for this experimental validation includes 2 Nvidia GeForce RTX 3090 GPUs. All the presented experiments are easily reproducible, with all the components released open-source [249], including the models' weights, architectures and dataset, software comparison tool, and simulator. Since the optimizations are state-of-the-art and build the enhancements on some precise components of the models, the results may vary depending on how modern and powerful the test hardware is, but the ideas

described here are general and applicable to any type of hardware and software setup.

Experiments were conducted to gauge the impact of optimized models in robot control and assess how control decision quality and frequency influence vehicle behavior. Various optimizations were compared individually to determine their significance in performance. The experiments focused on visual lane-following in urban scenarios without traffic or obstacles, emphasizing model optimization's importance. For clarity, only one CARLA scenario with random starting positions was used although the results apply to similar deep learning end-to-end control tasks.

7.3.1. Model performance offline evaluation table

Offline evaluation of the models gives a general idea of how the model performs. In Table 7.2, we present the offline evaluation results for each of the optimized models. This evaluation is conducted using batches of 64 images. During online testing, for each inference, only one image is used as input. Considering the use of a GPU for inference, the maximum gain in inference time is expected to be achieved with batches of several images instead of only using one image for each timestamp because of the parallelism of the GPUs.

In general, the model size is reduced when optimizing (compressed). The explanation for this fact is that the optimizations reduce the complexity of the model, hence reducing the space that it needs for storage. The optimization that causes the most reduction is found for the models using int8 quantization (see Table 7.1 for details), due to the lower precision of the numbers in the network, they need less memory space. For each optimization, we fine-tune its parameters to their utmost limits, carefully balancing them on the threshold just before a noticeable decline in quality occurs. For example, adjusting aggressively clustering and pruning parameters to their maximum settings. The precise parameter values are contingent upon each specific optimization and combination (comprehensively documented in the accompanying open-source code).

Looking at the GPU inferences frequency, optimized models generate much better results than models without optimizations with a maximum gain of 135 times of improvement (Baseline TF compared to (TensorRT) int8 quantization, which is the best result for TensorFlow framework). If we do not consider TensorRT framework, models with int8 quantization (see Table 7.1 for details) generate the best results in terms of GPU inferences frequency with 50 times faster results at best. Other combinations of optimizations do not improve the frequency further. Since the operations are conducted with a lower precision because it uses int8, the calculations are much faster and require less memory. If we consider TensorRT, the frequency increases to the highest point (135 times faster at best). It is due to the hardware-level optimizations of TensorRT, which are specific for certain hardware combinations and include the already presented optimization techniques with fine-grained adjustments for the hardware. Similarly to considering model size, the introduction of clustering or pruning is ineffective in increasing further the GPU inference

frequency when quantization is involved.

With the MSE, we can check whether the model-generated results are close to the supervised examples or not. Looking at the MSE evaluation results, the values remain close to the baseline model, improving the results for certain optimizations. This is important since we can generate models of a similar quality in terms of MSE with lower size and more GPU inference frequency. In this case, the best results appear when combining several optimizations or all available options, e.g., PCQAT (see Table 7.2 for detailed values).

The only outlier appears in the TensoRT TensorFlow int8 quantization, which generates worse MSE values. An explanation for this is that the model optimization limit has been surpassed, causing the final performance to drop significantly.

To summarize, the offline evaluation results show that optimizations provide more efficient models and the combination of all the optimizations generates the best results when considering model size, MSE, and GPU inference frequency. These results are repeated for both PyTorch and TensorFlow deep learning frameworks and their combinations with TensorRT. It is also important to point out that by comparing the TensorRT framework with deep learning framework-level optimizations, TensorRT generates the best results, although it is hardware-specific and more complicated to apply for different hardware combinations.

7.3.2. Robot control online evaluation table

In this experiment, we evaluate the baseline and each optimized model while driving a car on an urban test scenario inside the CARLA simulator in regular conditions and generate evaluation metrics with Behavior Metrics. Each model runs inside a vehicle controller, driving for two minutes, five times, and starting from a random position in the map, which allows the agent to drive that amount of time without encountering junctions or other unconsidered scenarios in that run. An illustrative video with the baseline model driving is available at [250]. In Table 7.3, we can see a summary of the results for each model. We select only the most informative evaluation metrics retrieved from Behavior Metrics for this particular experiment.

The controller frequency is always lower than the GPU inference frequency. The controller is the core computational system that drives the vehicle and performs several operations at each iteration. It receives the bird-eye image and transforms it before giving it to the model for inference. Since the deep learning model is inside the controller, the GPU inference frequency (model inferences alone) will always be higher than the controller frequency. We can see that the number for GPU inference frequency is similar to the ones obtained in the offline evaluation but slightly lower. This difference comes from the introduction of other computational loads like the simulator and the number of images used as input batch.

Optimization type	Model size (MB)	MSE test	GPU inferences frequency (Hz)
TensorFlow			
Baseline (No optimizations)	18.35	0.015	22.38
TensorFlow Model Optimization Toolkit (TF Lite)			
Baseline in TF Lite (No optimizations)	6.09	0.01590	610.91
Dynamic Range Quantization	1.54	0.01593	764.11
Integer Quantization	1.54	0.01591	1104.86
Integer (float fallback) Quantization	1.54	0.01588	1216.82
Float16 Quantization	3.05	0.015902	614.68
Quantization Aware Training	1.54	0.01317	1181.89
(Random sparse) Weight pruning	6.09	0.00919	609.92
(Random sparse) Weight pruning quantization	1.53	0.00911	755.86
Cluster preserving quantization aware	1.54	0.01202	1195.73
Pruning preserving quantization aware	1.54	0.00932	1179.22
Sparsity and cluster preserving quantization aware training (PCQAT)	1.54	0.00859	1182.76
TensorFlow TensorRT			
(TensorRT) Float32 Quantization	6.29	0.01079	2579.90
(TensorRT) Float16 Quantization	6.29	0.01079	2368.63
(TensorRT) Int8 Quantization	6.35	0.04791	2954.75
PyTorch			
Baseline (No optimizations)	6.09	0.00435	229.83
PyTorch Model Optimization Toolkit			
Dynamic Range Quantization	1.94	0.01206	675.54
Static Quantization	1.61	0.01207	1367.17
Quantization Aware Training	1.61	0.01109	853.94
Local Prune	6.12	0.01085	695.05
Global Prune	6.12	0.01096	705.23
Prune+Quantization	1.61	0.01094	852.60
PyTorch TensorRT			
(TensorRT) Float32 Quantization	6.12	0.00957	4377.41
(TensorRT) Float16 Quantization	6.12	0.00957	3986.85
(TensorRT) Int8 Quantization	6.18	0.00969	4058.54

Table 7.2: Offline evaluation of baseline models and their optimized versions.

The controller frequency is always better for optimized models. Having more GPU inference frequency is an indicator of possible higher controller frequencies. We can observe that the numbers are better but considering the GPU inference frequencies, we could expect even better controller frequency results. This situation occurs due to the initialization times of the deep learning model. When using optimized models, in the first iteration the deep learning model is loaded into the GPU so it is much slower than all other iterations. Considering the big difference in GPU inference frequency, we can ignore this difference, since it is expected to be negligible in the long term, for instance in longer experiments.

Both baseline models generate excellent results for the number of successful runs, completing all of them. The important point here is that the optimized models retrieve similar results, finishing the experimental evaluations with efficient results too, but with a higher model inference frequency. Looking at the number of GPU inference frequencies, similarly, as we have observed in the previous experiment, int8 quantized models (see Table 7.1 for details) are faster for both frameworks (see Fig. 7.4), generating 47 times faster inferences for TensorFlow and 2.3 times faster at best for PyTorch. Looking at the results of the position deviation and average speed metrics, are similar to those obtained using the baseline models. The rest of the insights for the optimization selection described in the previous experiment are also applicable in the online setting.

We have proved that the optimized models drive properly in an online test evaluation, without a reduction in the quality of the decisions and increasing their pace. The optimized models have the same overall behavior quality and are faster and smaller. The results are similar for PyToch and TensorFlow, which proves that the ideas apply to several configurations. If we consider TensorRT framework optimizations, they achieve the best number considering GPU inference frequency while maintaining the overall behavior quality, measured as successful runs. Considering int8 TensorRT optimizations, neither of them works correctly, which can again be explained because of trying to improve the model too much and reaching its optimization limit, which reduces the final online performance.

A summary of the results is shown in Table 7.4, including the improvement rate observed over the baseline model. We select the best-optimized models for each deep learning framework as the model that combines the majority of optimization techniques (quantization, pruning, fine-tuning, and/or clustering) since we have proved that t is the best-performing one. We observe improvement for each optimized option compared to the baseline, obtaining a gaining of a maximum of 47 times improvement in an online evaluation of inference frequency and 5 times of controller frequency. We have proved that optimizing the model is key for robot control efficiency.

Model	Controller frequency (Hz)	GPU inferences frequency (Hz)	Position deviation mean per km (m)	Average speed (km/h)	Successful runs
TensorFlow					
Baseline (No optimizations)	15.94	20.25	0.33	20.5	100%
TensorFlow Model Optimization Toolkit (TF Lite)					
Baseline TF Lite	70.36	438.30	0.25	23.5	100%
Dynamic Range Quantization	74.65	568.24	0.24	24.3	100%
Integer Quantization	75.43	771.83	0.24	23.1	100%
Integer (float fallback) Quantization	76.49	868.64	0.26	23.1	100%
Float16 Quantization	69.63	432.42	0.26	22.8	100%
Quantization Aware Training	77.17	823.85	0.25	-	100%
(Random sparse) Weight pruning	70.03	437.44	0.25	23.0	100%
(Random sparse) Weight pruning quantization	74.27	575.97	0.26	27.1	100%
Cluster preserving quantization aware	75.46	850.83	0.24	24.2	100%
Pruning preserving quantization aware	76.27	837.16	0.24	26.5	100%
Sparsity and cluster preserving quantization aware training (PCQAT)	79.50	867.15	0.24	24.2	100%
TensorFlow TensorRT					
(TensorRT) Float32 Quantization	79.10	1067.79	0.44	22.6	100%
(TensorRT) Float16 Quantization	62.03	967.09	0.49	22.9	100%
(TensorRT) Int8 Quantization	39.89	874.78	-	22.0	0%
PyTorch					
Baseline (No optimizations)	69.92	437.93	1.41	20.9	100%
PyTorch Model Optimization Toolkit					
Dynamic Range Quantization	72.28	579.97	0.24	23.1	100%
Static Quantization	77.74	1020.53	0.26	21.6	100%
Quantization Aware Training	76.37	1022.46	0.29	21.4	100%
Local Prune	70.43	474.88	0.28	24.1	100%
Global Prune	70.80	475.53	0.26	23.2	100%
Prune+Quantization	78.07	1014.47	0.26	25.8	100%
PyTorch TensorRT					
(TensorRT) Float32 Quantization	67.78	1030.00	0.45	20.1	100%
(TensorRT) Float16 Quantization	68.94	1036.99	0.43	21.2	100%
(TensorRT) Int8 Quantization	76.11	1496.99	4.76	20.1	20%

Table 7.3: Comparison of models and their optimized versions in a test environment considering some measured metrics provided by Behavior Metrics.

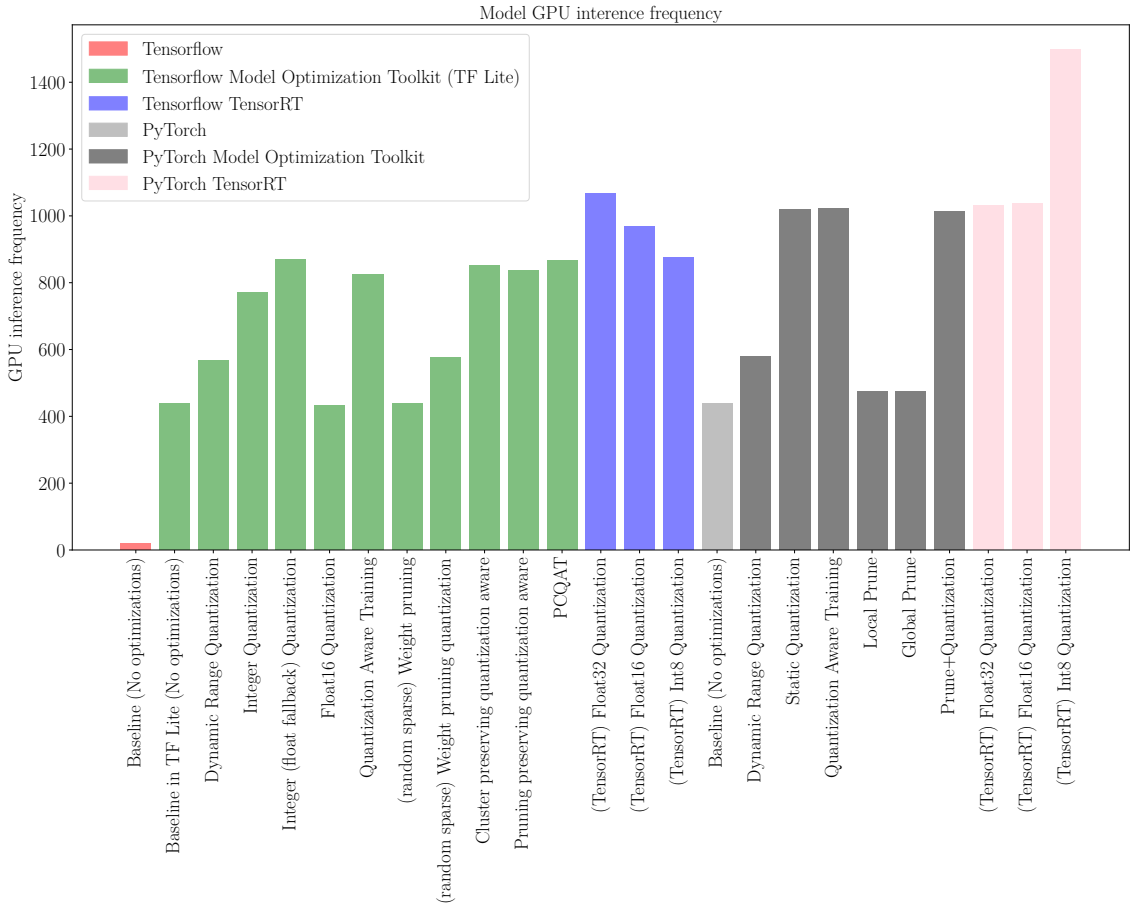


Figure 7.4: Detail of each model's GPU inference frequency.

7.3.3. Inference frequency and quality of decisions in robot control performance

Most successful robots in the real world have an important reactive part, which usually has one main control loop. This loop runs iterations at a given frequency, reading sensor measurements and commanding low-level decisions to the actuators on each iteration. For instance, in autonomous driving, there is usually a reactive local navigation controller. When following an end-to-end approach, on each iteration, the sensor data are provided as the input of the deep learning model, it is called for inference, and its outputs are commanded to the vehicle actuators. The overall quality of the robot's behavior depends both on the quality of the control decisions at each iteration and on the frequency of those iterations.

In this experiment, the visual lane-follow driving was the robotics application and the robot controller was the baseline deep learning model PilotNet*, as it is a successful one. All the control decisions were taken with that deep learning model and their quality is assumed to be good. The settings, worlds, starting points, etc. were the same as in the previous experiment, but many different frequencies of the controller were enforced, and the corresponding overall quality of the lane-follow application was measured, in terms of successful runs.

Model	Optimization	Model size (MB)	MSE test	Offline GPU inferences frequency (Hz) (improvement)	Controller frequency (Hz) (improvement)	Online GPU inferences frequency (Hz) (improvement)	Successful runs
TensorFlow							
Baseline	-	18.35	0.015	22.38	15.94	20.25	100%
Best optimized TF Lite	**Sparsity and cluster preserving quantization aware training (PCQAT)**	1.54	0.00859	1182.76 (x52)	79.50 (x5)	867.15 (x42)	100%
Best optimized TF TensorRT	**(TensorRT) Float16 Quantization**	6.29	0.01079	2368.75 (x100)	62.03 (x3)	967.09 (x47)	100%
PyTorch							
Baseline	-	6.09	0.00435	229.83	69.92	437.93	100%
Best optimized PyTorch	**Prune+Quantization**	1.61	0.01094	852.60 (x3)	78.07 (x1.2)	1014.47 (x2.3)	100%
Best optimized PyTorch TensorRT	**(TensorRT) Float16 Quantization**	6.12	0.00957	3986.85 (x17)	68.94 (x1)	1036.99 (x2.3)	100%

Table 7.4: Comparison summary of best models performance with the improvement rate observer.

To enforce different low controller frequencies, extra dummy computation loads were introduced on each iteration. To enforce high controller frequencies, the simulator itself was stopped and resumed so more controller iterations took place at each second of simulated time.

The results of this experiment are shown in Fig. 7.5. For that lane-follow application, we found a lower limit of 10 Hz for the controller frequency, a turning point in the robot's performance. Below that frequency threshold, the robot simply fails. Above it, the performance is pretty much the same. It is important to note that the only difference is the frequency of the decisions, their quality is always good as they come from the same deep learning model. The particular number of the frequency threshold for other applications and scenarios may depend on many factors, such as the car speed (faster cars will require a higher control frequency threshold), the complexity of the robot, and the dynamism of the environment, but this profile seems to appear in many robotics contexts. For this experiment, a simulated Tesla Model 3 with a stable speed of 30km/h driving in scenario Town02 is used.

The results in Fig. 7.5 also show the potential benefit of optimizing the deep learning models. For a given computing hardware, achieving faster models may increase the robot controller frequency, and so increase the robot application performance. In limited hardware, it may be even critical when regular models fall below the frequency threshold and optimized ones may be above it.

Finally, with the available hardware GPUs already used in all the previous experiments, we measured the performance at the maximum inference frequency of both the baseline PilotNet* and of the best-optimized model from it. The baseline model reached 437Hz with PyTorch, far from the minimum 10Hz. The best optimized model reaches 1014Hz. They are shown as vertical dotted lines in Fig. 7.5. Similar increments were obtained with TensorFlow. The particular values are not relevant, but they illustrate the advantage of optimizing the deep learning model. In this particular visual lane-follow application, the autonomous car moving slowly, is not critical, but in more demanding

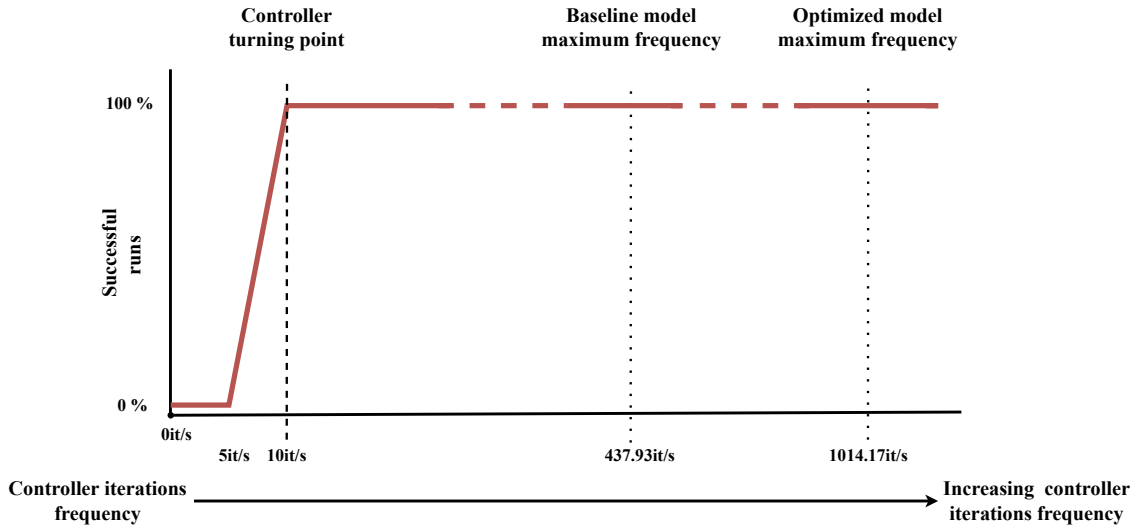


Figure 7.5: Quality of the robot behavior (measured as % of successful runs) vs the frequency of the control decisions

applications, higher car speeds, or even this one running on limited computing hardware, may make the difference.

7.4. Conclusions

In this chapter and published contribution [244], we have presented and studied several optimization techniques for deep learning models and applied them for the end-to-end visual control of an autonomous vehicle based on imitation learning. The particular application is lane-follow driving in urban scenarios. We have used optimized models and proved experimentally that optimizations improve the final system performance thanks to the speed-up in controller iteration frequency without losing quality on the control decisions. We have applied these optimizations individually and combined, implementing all variants in two different deep learning frameworks (PyTorch and TensorFlow). Hardware-specific optimizations (TensorRT) were applied too.

These models have been tested and validated offline and online in a state-of-the-art simulator for autonomous driving, CARLA. The experimental results show the impact of each optimization on the final robot application performance. Combining all the optimization techniques and tuning them in consonance with the baseline model, the optimized deep learning models drive with a similar quality of the control decisions but much faster. The inference frequency of the best-optimized deep learning model is 47 times faster than the baseline model in the online evaluation. And the robot controller with it runs 3 times faster, so the optimizations have proven its relevance. The optimized models can be used in a broader variety of hardware, especially low-resource and edge devices. This adaptability extends beyond the conventional scope of autonomous driving research, constituting a significant aspect of our innovation. We have proved these premises ex-

perimentally using Behavior Metrics. These metrics complement the common MSE on the supervised dataset and those directly provided by CARLA. Altogether they provide a more informative description of the performance of the system.

The dataset, models' weights and architectures, and comparison software tools are provided as open-source materials for the research community, which makes easy the replication of the presented results [249].

After exploring adding memory to the deep learning models that control the vehicles autonomously and optimize them, in the following chapter we explore extending the end-to-end coverage to more complex scenarios, particularly with traffic.

Chapter 8

End-to-end vision-based autonomous driving in traffic

This chapter presents another contribution to the thesis, which is under peer-review in a journal at the time of writing this document [251]. We have explored different autonomous driving ideas, specifically adding memory to the model to enhance its performance and optimizing the autonomous driving systems prioritizing fast, small, and reliable models. In this contribution, we present a shallow end-to-end vision-based deep learning approach for autonomous driving in traffic scenarios. The primary objectives include following the lane and maintaining a safe distance from possible preceding vehicles. In previous contributions, we did not consider other vehicles but in this one, they are a primary component of the research. The approach again leverages imitation learning, creating a supervised dataset for robot control from the same expert agent demonstrator in CARLA simulator. This dataset encompasses three different versions complementary to each other and we have made it publicly available along with the rest of the materials. The PilotNet neural model is utilized in two variants: the first one with complementary outputs for brake and throttle control commands along with dropout; the second one incorporates these improvements and adds the vehicle speed. Both models have been trained with the aforementioned dataset. The experimental results demonstrate that the models, despite their simplicity and shallow architecture, including only small-scale changes, successfully drive in traffic conditions without sacrificing performance in free-road environments, broadening their area of application widely. Additionally, the second model adeptly maintains a safe distance from leading cars and exhibits satisfactory generalization capabilities to diverse vehicle types. A new evaluation metric to measure the distance to the front vehicle has been created and added to Behavior Metrics; an open-source autonomous driving assessment tool built on CARLA that performs experimental validations of autonomous driving solutions.

8.1. Introduction

We have already discussed extensively the importance of the field of autonomous driving research to increase safety on the streets or improve the navigation of people in urban environments by optimizing the number of possible vehicles. We have described the PilotNet network in detail and have also enhanced its possible behavior with extensions (PilotNet*) testing its memory capabilities and optimizing it for hardware constraint devices.

Even so, we would need to improve the range of applications of the end-to-end autonomous driving system to be successful in real urban scenarios where the number of possible tasks is enormous and includes following the lane, considering other traffic agents, navigation... Also, previously we used a simplified perception system that leveraged bird-eye-view. In a real scenario, we would need to use other sensors available since the simplified bird-eye view used in previous chapters is not feasible. In this case, our research question faced in this chapter examines whether a visual end-to-end deep learning imitation learning shallow model can successfully drive an autonomous driving vehicle following the lane in urban scenarios without colliding with other agents. Consequently, we prioritize simplicity in our models. We introduce and experimentally compare two variants of the PilotNet model, called PilotNet* and PilotNet**. Both variants can drive in simulation in the state-of-the-art simulator CARLA, keeping the lane and the latter also managing traffic conditions with front vehicles. These models generate throttle, brake, and steering control from the visual information provided by a frontal camera installed onboard the vehicle. The models are trained using an imitation learning procedure from a supervised dataset, generated from data collected from an expert agent driving in a training scenario. Adding the throttle and brake to the baseline, the model can drive following the lane in urban scenarios, also considering turns. Including the speed into the model, the vehicle can also negotiate scenarios with leading vehicles, stopping when encountering a vehicle in front and resuming driving when the short-term path is clear again. The models are validated experimentally under different conditions in test urban scenarios, using a variety of front vehicles and proving the generalization of the model to never-seen situations. The online experimental validation is conducted using the comparison software tool Behavior Metrics (see Fig. 8.1. for architectural details), which we have updated including evaluation metrics suitable for the presented problem of follow-lane in traffic situations. As we have already addressed, these metrics complement the common offline evaluation metrics used in machine learning, adding a broader context to each evaluated model's benefits and possible limitations. We consider online evaluation as the one conducted in simulation and offline evaluation as the one conducted comparing the model to supervised data. We keep other autonomous driving tasks out of the scope of the present work, such as negotiating road intersections or considering traffic signals.

One contribution of the paper presented in this chapter is the proposal of slight modifications to the shallow baseline PilotNet model, which demonstrates that with small-scale changes, it is possible to expand its application area widely. They allow the autonomous

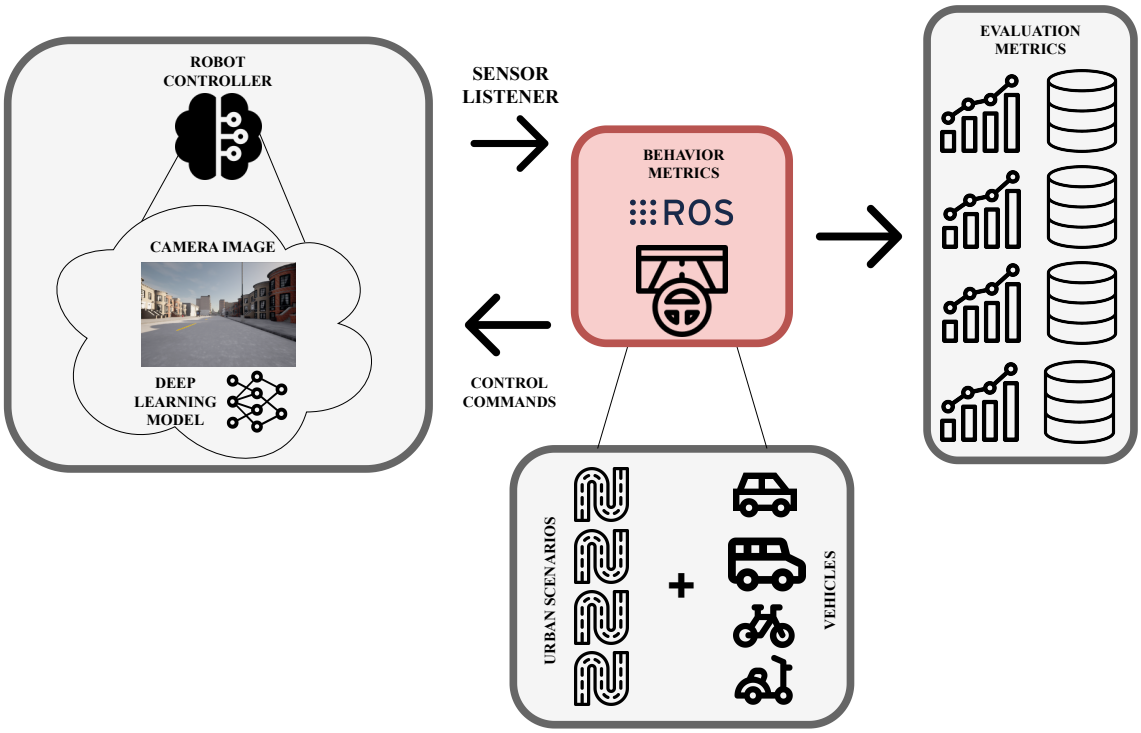


Figure 8.1: Behavior Metrics evaluation software tool architecture with different urban scenarios and vehicles.

car to deal with different ahead vehicles in the same lane successfully, including those it has never encountered before, slowing down or stopping before them, resuming the movement, and following them when the safety standards are satisfied. We experimentally validate this assumption extensively in Section 8.3 and its generalization to new scenarios. Another contribution is the new fine-grain metric in the assessment tool that measures the distance to other vehicles based on the data extracted from CARLA, which gives a better intuition about how the robot controller behaves. We provide all models, architectures, and datasets as open-source, along with the comparison software tool [252] for validation and extension, which are also contributions. As a result, researchers may leverage this advancement to extend or develop an enhanced version of the models, architectures, dataset, or software.

8.2. Imitation learning for driving in traffic

In this section, we will discuss the three primary components of our work. Including the generated dataset versions for imitation learning, the modifications of the baseline deep learning model PilotNet created for this work, and the training procedure followed in the development of the final models.

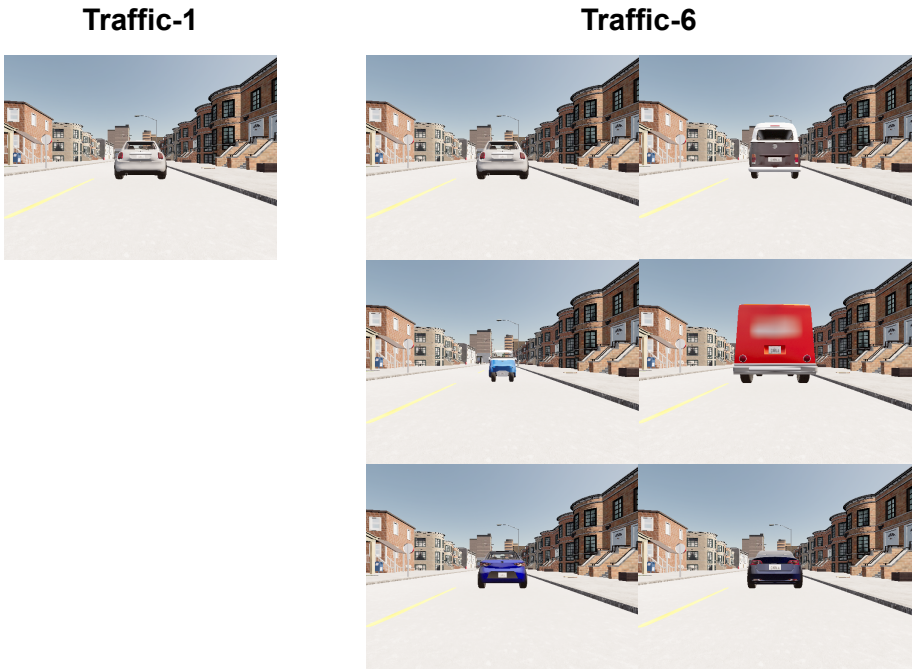


Figure 8.2: Detail of included vehicles in each dataset version.

8.2.1. Dataset and versions

The supervised dataset is collected on an urban scenario (Town02) of the CARLA simulator [186] using an imitation learning approach. The expert agent is an integral component of the simulator with access to privileged simulator data and it bases its behavior on handcrafted rules. It is set to follow a specific route keeping the lane and covering the urban scenario while the visual data is generated by the camera, and the corresponding control commands are recorded. The agent's maximum velocity is approximately 30 km/h.

The route includes turning situations and some encounters with front vehicles but without possible intersection situations or consideration of traffic lights/signals, which are not in the scope of this work. For simplicity, we focus only on urban scenarios rather than including highways. Nevertheless, the ideas presented here are also applicable to these and other possible scenarios. Through this process, we generated a dataset of 140K images along with supervised demonstration data. Approximately, 47% of them are images containing other vehicles present in it.

We include three dataset versions that complement each other. The first, Traffic-0, does not consider any traffic factors. The second version, Traffic-1, includes the former and traffic examples with only one type of front vehicle, a typical small urban car. The third version, Traffic-6, encompasses both previous versions and examples with five other vehicle types including two vans and three urban cars of different sizes and colors (see Fig. 8.2. for details about which vehicles are included in each dataset version).

For training and testing, the ego vehicle is equipped with an onboard RGB camera

that is oriented forward, capturing images with dimensions of 480 pixels in height and 650 pixels in width (480x650).

The image that the deep learning model processes contains a significant amount of information, but from that frontal image of the urban scenario, not all the content is relevant. We crop the image to reduce its complexity. By cropping the image to exclude elements such as the sky and buildings, which are not needed for generating control commands in this simplified context, unnecessary data is effectively removed. In addition to the cropping, the image is compressed. As a result, the input images are reduced to a size of 66 pixels in height and 200 in width, representing only about 4% of the original image size. This pre-processing optimizes the data and reduces the dataset size facilitating faster training.

8.2.2. Baseline model and its modifications

The two models proposed for this project are variations of the baseline PilotNet model [159] (see Fig. 8.3. for details about each architecture), built using the Tensorflow [120] framework. The baseline PilotNet network consists of 9 layers which include a batch normalization layer, 5 convolutional layers, and 3 fully connected layers. The first part of the model is responsible for extracting features from the input visual data and the latter part generates the final control commands from that extracted features.

We introduce enhancements to the baseline model, which are specific to the current project. The primary motivation behind these modifications is to investigate whether a shallow, established model can significantly broaden its applicability through minor enhancements.

In the first variant, called PilotNet*, an extra output is added alongside the steering command. This new output generates control signals for the throttle and brake. The architecture also includes regularization techniques, specifically batch normalization and dropout layers [253] with a 0.1 rate. The batch normalization layers were already proposed in the original PilotNet work while the dropout layers are inserted between the final dense layers of the PilotNet* model to make it more suitable for supervised data and prevent overfitting. The dropout rate is determined through experimental validation. Increasing it further has been found to yield catastrophic results. The objective behind the development of this model lies in determining whether a unified architecture can effectively handle multiple outputs of varied natures, such as throttle, steering, and brake, thereby significantly enhancing the overall behavior of the model.

In the second model, called PilotNet**, we build upon the PilotNet* model by including the vehicle's velocity for each time step. This velocity is added to the input alongside the image, resulting in a final input image of shape (66, 200, 4). The extra channel is uniformly filled with the normalized speed, scaled between 0 and 1. In real cars, this speed may be taken from an onboard speedometer. The rationale behind this addition is

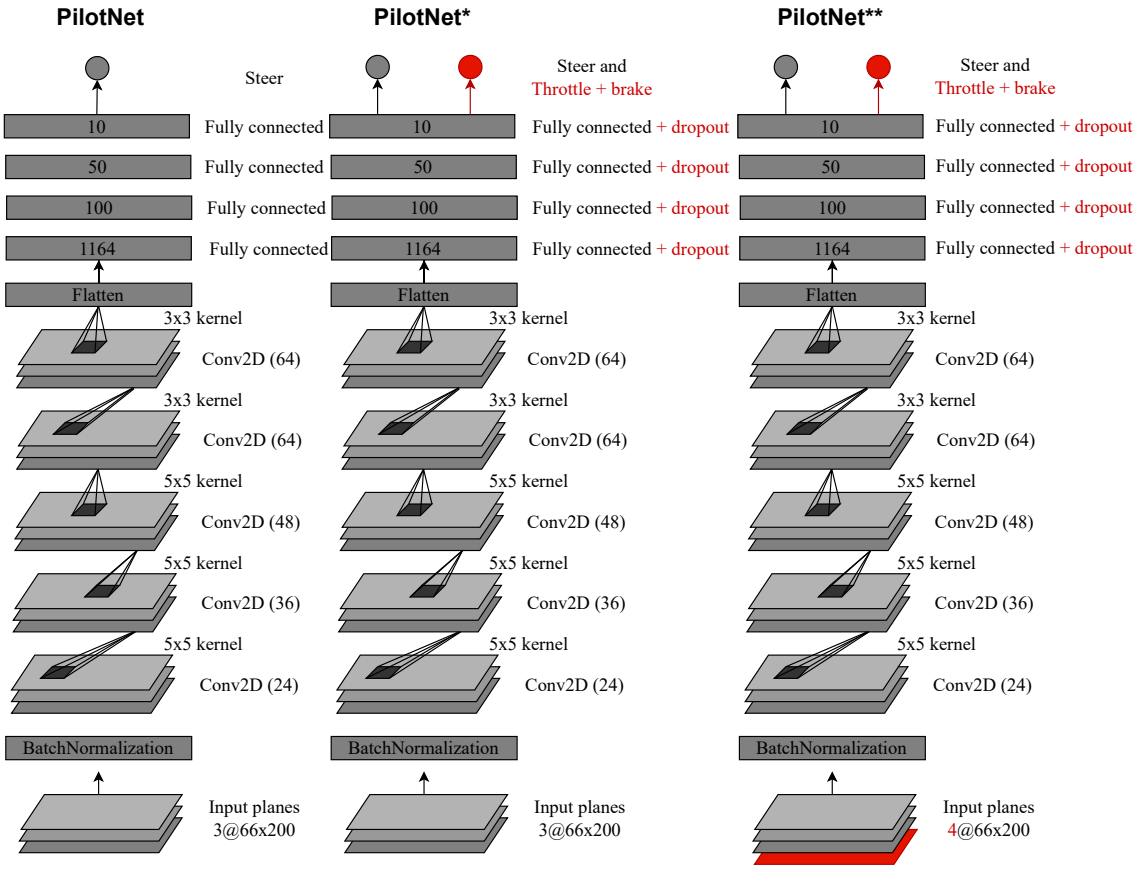


Figure 8.3: PilotNet baseline model and its variations PilotNet* and PilotNet**. In red, introduced changes are highlighted.

to ascertain whether a model equipped with knowledge of its speed outperforms a model lacking such information and to examine its impact on the system's performance in the presence of other vehicles in the scenario. In Fig. 8.3. details about each architecture are displayed, showing their differences.

8.2.3. Training procedure

During the training procedure of each of the models, we introduce a dataset pre-processing stage. In this stage, the data is transformed and prepared for training, including regularization techniques such as data augmentation. We also include early stopping as a regularization technique. In this case, we include data augmentation techniques for adjusting brightness and contrast, modifying the image colors such as hue, saturation, and value components, adding blur, and simulating various weather conditions like rain, snow, fog, sun flare, and shadows. All these techniques are common in data augmentation frameworks, like Albumentations [176] which is the one used here.

The generated dataset is imbalanced, including a lot of straight-lane samples where the steering is not relevant. This situation is common in autonomous driving and some techniques have been already presented to address this issue, like Dagger [174]. To over-

Table 8.1: Offline evaluation measures of the model's performance.

Model **Training dataset**	**PilotNet*** **Traffic-1**	**PilotNet**** **Traffic-1**	**PilotNet**** **Traffic-6**
MAE	0.04685	0.04121	0.03842
MSE	0.01138	0.00930	0.00706

come these situations, we introduce oversampling of turn situations and generate more data for particularly relevant urban areas, such as curves. By doing so, the dataset is more balanced.

For training, the hardware used includes an Nvidia 3060 GTX GPU. This hardware is the same used in the experimental validation of the presented models.

Table 8.1 presents the mean squared error (MSE) and the mean absolute error (MAE) of the three proposed models. The loss function used for training is MSE. While these metrics offer insight into how the models have been trained and their capabilities, they do not suffice to draw a definitive conclusion regarding the overall performance and generalization capabilities of each model in a robotics closed-loop problem such as this one. To further expand the scope of assessment, the use of online evaluation metrics is crucial. Behavior Metrics [238] assists in this endeavor by examining in detail the behavior of each model in real-time scenarios and providing detailed insights into their respective performance and adaptability.

To gain a better understanding of PilotNet**'s behavior and to effectively spot any unusual behaviors, we utilize activation heat maps, a technique explored in prior studies [147]. This visualization method allows for an understanding of where the neural network places its attention, providing useful insights into the decision-making process. In this case, we obtained the activation heat map using the Grad-CAM [254] algorithm for PilotNet** trained with Traffic-6. Fig. 8.4. visually represents the more relevant parts detected by the activations of the last convolutional layer of the network. PilotNet** recognizes the lane markings and edges and highlights the wheels of the front car. The attention placed on the wheels is a result of training with diverse vehicles, enabling effective generalization and reducing the risk of collisions with any on-road vehicle.

8.3. Experiments

In this section, we conduct an experimental validation of the two models: PilotNet* and PilotNet** for the task of follow-lane in traffic situations. These experiments provide relevant results about each model behavior in three different conditions: the first two focused on typical executions without and with traffic, and the third one testing the models' generalization capabilities for obstacle avoidance with a varied set of vehicles.

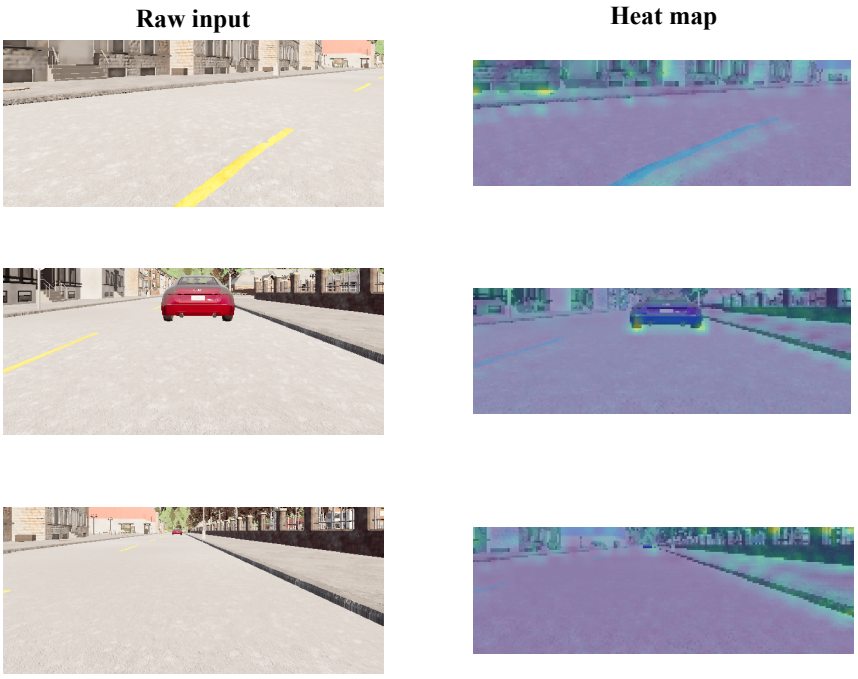


Figure 8.4: Activation heat map visualization from the last CNN layer of PilotNet** model.

For these experiments, we have used the CARLA simulator and Behavior Metrics evaluation tool running at 10Hz. The hardware used for the experiments includes an Nvidia 3060 GTX GPU. The evaluation takes place in Town02 where the training data was collected; and in Town01, which was never used to train the model. We provide both scenarios to showcase the differences between a circuit already employed in the dataset generation and a never-seen scenario. For each experiment, models are trained with a particular version of the dataset, also showcasing their differences.

The car starts from a fixed position in the urban scenario, driving clockwise and anti-clockwise. These starting points form a closed loop route, so the vehicle can complete a lap reaching the starting point again after some time.

We omit a comparison with prior baseline methods as our research question specifically focuses on whether a shallow visual-based end-to-end deep learning model, utilizing imitation learning, can autonomously navigate without colliding with other vehicles. While existing state-of-the-art models are designed for a broader range of tasks, they often incorporate more sensors, deeper and more complex architectures, and utilize large datasets. The contribution presented in Chapter 7 underscores the significance of small and efficient deep learning networks for autonomous driving, networks suitable for deployment across various devices with differing hardware capabilities and we prioritize the simplicity of our models.

Table 8.2: Metrics for two different towns and models in free-road conditions. Success rate: the higher the better; the rest: the lower the better.

	Town01			**Town02**		
Model	**PilotNet***	**PilotNet****	**PilotNet****	**PilotNet***	**PilotNet****	**PilotNet****
Training dataset	**Traffic-1**	**Traffic-1**	**Traffic-6**	**Traffic-1**	**Traffic-1**	**Traffic-6**
Success Rate (%)	100	100	100	100	100	100
MPD	0.33	0.3	**0.19**	0.84	0.49	**0.32**
Lane Invasions	14.884	10.02	**4.75**	26.56	15.4	**3.42**

8.3.1. Typical execution without traffic

In this experiment, the model underwent three clockwise laps and three anticlockwise laps, traversing designated routes within each town.

This experiment tests the follow-lane capacities of the models. Specifically, we consider two models to see if they can still effectively follow the lane in the absence of oncoming cars. The PilotNet* model trained with the Traffic-1 version of the dataset and the PilotNet** model trained with the Traffic-1 and Traffic-6 versions of the dataset. The results for these experiments are provided in Table 8.2, including the evaluation metrics presented in the previous section. All three models drive successfully keeping the lane without any missed attempts, but we can already see some differences. We can see that PilotNet** trained with Traffic-6 and Traffic-1 is better in terms of mean position deviation and lane invasions number than PilotNet*. This fact proves that adding speed is valuable even in the simplest task of following the lane without traffic. Furthermore, for the PilotNet**, training with a diverse dataset like Traffic-6, compared to Traffic-1, can improve the vehicle's perception of its surroundings in the town and reduce the possibility of confusing certain urban sections with front vehicles, making its driving more confident and smooth.

8.3.2. Typical execution with traffic

In this experiment, we test the models in a more difficult situation, with one front vehicle to understand the implications of this new scenario. The typical execution with traffic consisted of a total of 12 runs, with the same settings as without traffic. We evaluate the model's performance in a full lap simulation, specifically focusing on both lane-keeping and obstacle avoidance. The additional vehicle moves independently around the scenario, always preceding the ego vehicle, and it will drive slowly to disturb the ego vehicle as much as possible. Although the ego vehicle ignores traffic lights and signals, the additional front vehicle considers traffic lights and follows their indications. As in the previous experiment, the models are trained with the Traffic-1 and Traffic-6 versions of the dataset. The front vehicle used in the Traffic-1 dataset is the same as the one used for this particular experiment. In Table 8.3, the results are shown. We can see a clear difference

Table 8.3: Metrics for two different towns and models in in-traffic conditions

	Town01			**Town02**		
Model	**PilotNet***	**PilotNet****	**PilotNet****	**PilotNet***	**PilotNet****	**PilotNet****
Training dataset	**Traffic-1**	**Traffic-1**	**Traffic-6**	**Traffic-1**	**Traffic-1**	**Traffic-6**
Success Rate (%)	0	16	**81**	0	83	**100**
MPD	43.12	18.07	**0.26**	50.57	1.97	**0.32**
Lane Invasions	28.87	25.84	**6.54**	69.65	21.15	**1.48**

now between PilotNet* and PilotNet** where the latter outperforms the former. This difference in performance dealing with other vehicles ahead is what makes the PilotNet** a promising model to be trained with Traffic-6 to generalize to different types of front vehicles and is more promising than PilotNet* which fails in many runs.

We observe that the Success rate is higher for PilotNet** trained with Traffic-6, completing each experiment in Town02 and the majority of the experiments in Town01, which is the actual test scenario. On the contrary, PilotNet* results are abruptly worse, without any successful experiment. These results show that PilotNet* always collides with the front vehicle at some point in the experiment. The results for the rest of the evaluated metrics follow a similar pattern. Looking at MPD, we can see that PilotNet** generates great results, so we can consider that it can follow the lane correctly while keeping a low number of lane invasions.

Comparing both PilotNet** models trained with different datasets, we observe that the version trained with Traffic-6 outperforms the one trained with Traffic-1. This is evidenced by its more robust understanding of the environment, effectively distinguishing between front vehicles and structures such as buildings. The discrepancy in performance between Town02 and Town01 may stem from its inability to isolate the front vehicle from the urban background

This experiment, not only highlights the importance of having a much more varied dataset so that the model can drive with other vehicles in never-before-seen towns but also proves the importance of adding the speed to the model architecture for optimal performance in traffic situations. The rationale behind this approach is that by incorporating the speed into the model, the ego vehicle can enhance its control over its speed with greater precision compared to scenarios where this information is not considered. This level of control allows the ego vehicle to swiftly respond to nearby vehicles by reducing its speed when necessary, contributing to overall safety and smoother driving behavior.

In contrast, PilotNet* does not consider the speed of the ego vehicle during the experiment. It generates control decisions solely based on instantaneous visual data, which poses challenges in adjusting its velocity. The visual input appears to be deficient in critical information when encountering another vehicle, as the ego vehicle lacks crucial data regarding its state and current speed.

Table 8.4: Metrics for the distance to the front vehicle.

	Town01	
Model	**PilotNet****	**PilotNet****
Training dataset	**Traffic-1**	**Traffic-6**
Dangerous distance	6%	**2%**
Short distance	25%	16%
Medium distance	27%	30%
Great distance	42%	52%
Success rate	16%	**86%**

8.3.3. Generalization for different front vehicles

In the final experiment, we test the generalization capabilities of the models to never-seen front vehicles. For this experiment, we discarded the PilotNet* model because of its poor performance in the previous experiment. Instead, we use the PilotNet** trained for the second experiment with the Traffic-1 dataset, and we compare it to a fine-tuned version of PilotNet** trained with the Traffic-6 dataset.

To assess the generalization capabilities, it was unnecessary to conduct full-lap trials. Instead, we tested the ego vehicle with a series of vehicles in front, which moved at a significantly slow pace, stopping at red traffic lights and resuming driving when the lights turned green.

In this experiment, a total of 12 distinct vehicles were employed, comprising 8 novel vehicles unseen during the training phase, and 4 vehicles previously encountered in the training dataset (see Fig. 8.5 for details of used vehicles). We conducted tests with each leading vehicle traversing the same route in both clockwise and anticlockwise directions. Finally, we carried out this experiment three times to ensure the results were reliable. This gave us a total of 72 runs for each model concluding with a total of 144 runs. Each run gave us two types of metrics: Success rate and Distance to the front car. The dangerous distance should be avoided as it is deemed unacceptable in real-world scenarios which we consider unsafe. Complying with these distances is crucial for safe driving in real traffic. The Success rate metric measures cases where the ego vehicle encounters the front vehicle without any collisions, indicating the ego vehicle's ability to detect and respond to obstacles in its path.

Table 8.4 presents the ratio of the ego vehicle's distance behind the front car for a one-kilometer distance, allowing us to assess the behavior of the ego vehicle. It may be observed that the PilotNet** trained with Traffic-6 does a better job than PilotNet** trained with Traffic-1 by spending less time at dangerous distances from the front car.

Additionally, we can analyze how the vehicle transitions from a far distance to a close distance by examining the values in Table 8.4. The PilotNet** [255] trained with Traffic-6 exhibits a progressive descent of the speed from greater distances to closer ones, spending

Table 8.5: Success rate metric for each of the 12 vehicles.

	Town01	
Model	**PilotNet****	**PilotNet****
Training dataset	**Traffic-1**	**Traffic-6**
vehicle.mini.cooper_s	50%	83%
vehicle.volkswagen.t2	0%	100%
vehicle.micro.microlino	50%	100%
vehicle.carlamotors.carlacola	0%	100%
vehicle.jeep.wrangler_rubicon	0%	100%
vehicle.citroen.c3	17%	100%
vehicle.toyota.prius	0%	83%
vehicle.dodge.charger_police	33%	83%
vehicle.kawasaki.ninja	0%	100%
vehicle.diamondback.century	0%	50%
vehicle.ford.ambulance	50%	66%
vehicle.carlamotors.firetruck	0%	66%

more time the farther it is from the front car and reducing its distance as it approaches, which leads to the expected stopping behavior. On the other hand, the PilotNet** trained with Traffic-1 shows almost the same amount of time spent on medium and short distances. It lacks progressive advancement and struggles when confronted with an obstacle ahead.

Table 8.4 also shows the Success rate of each PilotNet** and the better generalization capacities from the PilotNet** trained with Traffic-6 compared to the PilotNet** trained with Traffic-1.

The fact that this performance is retrieved from Town01 proves that the models can generalize to other never-seen scenarios. Again, we have also proved the importance of adding speed to the architecture for a better understanding of the world that the vehicles use for making their control decisions.

PilotNet** trained with Traffic-6 dataset demonstrates its capabilities in lane-keeping and adaptive behavior when encountering vehicles of different shapes and colors. The experimental evaluation shows that it demonstrates an ability to understand its surroundings and has proven its effectiveness in maintaining appropriate distances and avoiding collisions with various vehicles, such as other cars and motorcycles, including vehicles that the model has never seen before in training (displayed on the last 8 rows of Table 8.5). The reasoning behind this result is that, due to exposure to a wider range of vehicles, the model trained with Traffic-6 demonstrates improved generalization capabilities, as it gains a deeper understanding of the concept of a vehicle and develops more optimal strategies when encountering them.

However, there are also some limitations when faced with previously unseen road

Figure 8.5: Detail of vehicles used for experimental validation.

users, such as cyclists, ambulances, and firetrucks (see Fig. 8.5.). The Success rate for each front vehicle, as displayed in Table 8.5, not only highlights the overall superiority of the PilotNet** trained with Traffic-6 compared to Traffic-1, as indicated in Table 8.4 but also underscores its mentioned limitations. Particularly noticeable are the lower Success rates associated with the Diamondback Century (bicycle), ambulance, and firetruck.

Due to its reliance on the visual cues provided by the wheels of the leading vehicle, as outlined in Fig. 8.4, the ego vehicle encounters difficulty in accurately determining whether to stop or proceed when confronted with a cyclist ahead. This challenge arises from the narrower tires typically found on bicycles compared to those of conventional road vehicles.

The challenge posed by ambulance and firetruck detection, as depicted in the last two images of Fig. 8.5, arises from the partial concealment of their wheels from the ego vehicle's perspective. This ambiguity complicates the model's capacity to consistently distinguish between car tires and other objects, although it occasionally manages to stop for such vehicles.

8.4. Conclusions

In this contribution, we present a proposal for safe autonomous driving in traffic scenarios following an end-to-end vision-based approach with imitation learning and deep learn-

ing. We have generated a new supervised dataset with many examples of the onboard camera images and the corresponding control commands recorded from the expert agent demonstrations in free-road or in-traffic conditions within the CARLA simulator.

We have developed and described two deep learning models based on PilotNet, adding in PilotNet* new dropout layers and outputs for controlling throttle, and brake, in addition to the previous steering option and also including in PilotNet** the speed as input.

These shallow networks slightly modified from the baseline have been trained using this supervised dataset using data augmentation and balancing the raw dataset. They have been experimentally evaluated and validated, Beyond low values of the loss function in the test dataset, the system has been validated online with the state-of-the-art Carla simulator, in several Towns, using objective, holistic, and quantitative metrics from the Behavior Metrics tool. For instance: mean position deviation from lane center, lane invasions, and distance to the front vehicle.

The experimental results show that the PilotNet** model, when trained with the Traffic-6 dataset, successfully drives the var in traffic conditions without sacrificing performance in free-road conditions. It also keeps safety distance from the oncoming cars and even properly generalizes to several types of front vehicles, including vehicles never seen before in the training stage. These results are observed with PilotNet** model proving that those model modifications, despite being slight and applied to an apparent simple model, are good contributions and enough to achieve the new desired 'drive in traffic' capability beyond the basic lane-following behavior.

Chapter 9

Conclusions and future research

In this chapter, we review the initial research goals and contributions outlined in Chapter 1 and subsequently offer the principal conclusions derived from this thesis. Furthermore, we delineate potential avenues for future research within the field.

9.1. Conclusions

In this thesis, we have proposed several research goals in autonomous driving and traffic monitoring. We have addressed them and made several contributions to these fields. We started with *a comprehensive literature review* of the different fields involved in computer vision, robotics, and artificial intelligence. Particularly, we have reviewed state-of-the-art literature for object detection solutions based on deep learning and traffic monitoring solutions. After that, we have reviewed the latest ideas inside the autonomous driving field and specially the end-to-end approach for generating solutions. This has included a review of the possibilities of the addition of memory to the system or other types of inputs like the speed. Also, works regarding imitation learning, which is the primary training technique used for deep learning models development have been reviewed. After that, we explored the assessment of autonomous driving solutions in simulation and understood the importance of the fine-grained metrics in addition to the baseline ones provided by the simulator or even other leaderboards. Finally, we have explored the optimization of deep learning models for autonomous driving, where we have presented the set of available state-of-the-art techniques and the implications of their use.

Following the literature review, we have presented our first contribution, TrafficSensor, in Chapter 3. This system is *a solution for vehicle monitoring using deep learning*. The system is capable of recognizing seven different classes. To address this goal, a new dataset was developed, and four different deep learning object detection models were evaluated. The system proved robust against different visual conditions and possible image disturbances. From the comparison of models, YOLOv4 outperformed the rest and was selected as the backbone for the construction of the monitoring application presented.

With the insights extracted from this research contribution, we elucidated some pivotal concepts that lay the foundation for the rest of the research contributions. We discerned that the robustness provided by deep learning solutions helps in the perception within the domain of road traffic. Additionally, we also understood the importance of the assessment of deep learning solutions efficiently.

Addressing these questions, we presented Detection Metrics, an original contribution of this thesis. This open-source software is targeted toward *the massive and unattended automatic assessment of deep learning visual object detection models*. This contribution was used for the previous TrafficSensor validation and it was also validated in another experiment. This software presented two main workflows for working with object detection models and large visual datasets. One of the most relevant contributions is found in the headless evaluation, which evaluates automatically several models independently for a batch of visual datasets, generating object metrics that help in the research process. We proved experimentally that this software application is useful in both traffic monitoring scenarios and perception modules inside an autonomous driving system.

Continuing with the development of the research and the lessons learned from these previous contributions, we introduced Behavior Metrics, *an open source tool for the online assessment of autonomous driving systems*. This tool is based on the importance of generating fine grained evaluation metrics for autonomous driving tasks to help researchers and practitioners evolve their solutions. It supports different autonomous driving tasks that include lane following, driving in traffic, and point-to-point navigation, with their particular evaluation metrics. It is designed to be easily extended with more tasks. This solution complements the simulator-provided metrics, offering two pipelines (headless and GUI mode) facilitating the experimental validation of solutions. This software tool was used for the following presented contributions.

We continued exploring and contributing to the autonomous driving field. We presented four different architectures and a variation of each of them for end-to-end autonomous driving based on vision and imitation learning. Beyond achieving the *basic visual follow lane application using imitation learning*, the study explored the *implications of the addition of visual memory and kinematic data* to some apparently simple deep learning architectures and exploring how these changes impacted on their behavior for a lane following scenario. This study included ablation studies and exploration of extreme situations to get more details about the areas where these new models were more interesting. The study proved that the kinematic input data is key for the system's inner understanding of the world and to regulate itself in terms of speed. The addition of visual memory and kinematic data also proved interesting results in extreme situations like controlling the vehicle at extreme speeds where the input data was corrupted.

Continuing the exploration of end-to-end autonomous driving systems, we then studied the possibilities that *optimization of neural models* adds to the performance of these models and how they affect the final behavior. These models proved experimentally that

optimizations improve the final system performance thanks to the speed-up in controller iteration frequency without losing quality on the control decisions. The study included the available optimization techniques available at the most common deep learning development frameworks, PyTorch and TensorFlow, along with hardware-level optimization provided using TensorRT. These models proved to be faster and smaller in size, enabling its use on a broader range of hardware setups. This was one of the main concerns that motivated this study, since many deep learning model solutions are typically much hardware-intensive and require hardware that is very expensive or even the system has to be deployed on smaller-capabilities hardware (edge devices).

In the last chapter and based on the previous contributions, we presented a proposal for safe autonomous *driving in traffic* scenarios following an end-to-end vision-based approach with imitation learning and deep learning. For this contribution, we generated a new dataset from an expert agent driving in free-road and in traffic conditions using the CARLA simulator. Based on the ideas learned from previous contributions, we developed a deep learning model capable of driving in traffic situations. The autonomous vehicle adjusts its velocity according to preceding vehicles, halts when they do, and resumes motion upon their restart. This model, despite its simplicity and slight modification from its baseline form, was able to drive autonomously meeting the high safety standards even in traffic. These traffic included different types of vehicles with different visual appearance and morphologies, proving that simple models with small modifications can broaden their area of applicability significantly. Simplicity in this case is also a key factor and something that we seek. The end-to-end model has amplified its area of application along the thesis while maintaining a simplicity focus.

9.2. Results summary

In this section, we summarize the key findings and contributions extracted from this thesis linking them with the research goals outlined in Chapter 1.

- **[RG1] Study the state of the art of autonomous driving systems, focused on end-to-end systems and related fields:** we conducted a detailed and informed literature review in Chapter 2. This literature review included a general exploration of computer vision and object detection techniques based on deep learning. Then, a review of autonomous driving solutions and the most advanced ones for addressing it using end-to-end approach. After that, the tools for the development of autonomous driving systems were explored, including the assessment of solutions, simulators, datasets, and benchmarks. Finally, we explored in detail state-of-the-art of optimization of deep learning models and the most useful techniques.
- **[RG2] Generation of a traffic monitoring tool based on object detection:** TrafficSensor application was developed and presented for monitoring vehicles in the real-world based on computer vision with commodity hardware. A combination of

object detection and tracking was proposed and validated experimentally using the state-of-the-art model YOLOv4 to generate the system. The validation was conducted extensively using Detection Metrics, which constitutes another contribution of the thesis. This system was presented in Chapter 3.

- **[RG3] Generation of an open-source object detection assessment software to validate solutions and advance research:** in Chapter 4, we presented Detection Metrics. This is another point that we had as a research goal and that has been successfully addressed. We generated an open-source software tool for assessing object detection solutions effectively and unattended. With this tool, a researcher could compare easily different models of a set of diverse object detection datasets and generate reporting data to help discriminate between solutions.

- **[RG4] Generation of an open-source autonomous driving assessment software to conduct the experiments and generate quantitative data about the behavior of the autonomous driving solutions:** we successfully completed this task creating the software tool Behavior Metrics. This tool was presented in detail in Chapter 5, and constitutes one of the contributions of this thesis. It is capable of assessing different autonomous driving solutions for a diverse set of tasks that include lane following, driving in traffic, and point-to-point navigation. The fine-grained metrics help the researcher in the validation of autonomous driving solutions, complementing the simulators' metrics.

- **[RG5] Generate end-to-end autonomous driving agents for visual lane following that enhance their behavior based on the addition of visual *memory* and *kinematic* input to visual deep learning imitation learning models:** we explored and presented different architectures and variations for end-to-end autonomous driving based on vision and imitation learning. In this research, we explored different possibilities for adding memory capabilities to the architectures and also explored the addition of kinematic data as input. This study explored how these complements on a fairly shallow model affected the system performance and how they could amplify the capabilities in a lane following application. An ablation study supported this study and we also explored extreme situations, proving that the addition of memory capabilities and kinematic data caused the model to amplify its scope of applicability and improved its general behavior. This research was presented in Chapter 6

- **[RG6] Development and study of optimized end-to-end autonomous driving control models which improved their performance with the latest deep learning optimization techniques:** this research goal was addressed in Chapter 7, where we explored the possibilities that the optimization of deep learning models could add to the final development of autonomous driving solutions. With this in-depth study, we generated optimized models exploring the most used optimization techniques available at deep learning frameworks and even exploring hardware-specific

optimization techniques with TensorRT. The generated models proved to be faster and smaller while maintaining great decision quality.

- **[RG7] Enhancement of shallow models for end-to-end autonomous driving control including traffic situations with other vehicles in the same road:** the last research goal was explored in detail in Chapter 8. The goal included the addition of traffic to the simulated scenario and the study of the enhancement of fairly shallow models to make them capable of addressing this task considering different types of vehicles. We successfully achieved this research goal with a model capable of driving in traffic situations while maintaining a safe distance from preceding vehicles.

9.3. Research contributions

Some of the results presented in this thesis have been published and shared with the scientific community through collaborative papers with other researchers:

Journal and conference papers:

- S. Paniego, R. Calvo-Palomino, and J. Cañas, "Behavior Metrics: An Open-Source Assessment Tool for Autonomous Driving Tasks," *Software X*, vol. 26, pp. 101702, 2024 doi: 10.1016/j.softx.2024.101702. [Online].
Available: https://doi.org/10.1016/j.softx.2024.101702 [239]

- S. Paniego, N. Paliwal, and J. Cañas, "Model optimization in deep learning based robot control for autonomous driving," *IEEE Robotics and Automation Letters and IEEE International Conference on Robotics and Automation (ICRA)*, vol. 9, no. 1, pp. 715–722, 2024. doi: 10.1109/LRA.2023.3336244. [Online]. Available: https://doi.org/10.1109/LRA.2023.3336244 [244]

- S. Paniego, V. Sharma, and J. M. Cañas, "Open source assessment of deep learning visual object detection," *Sensors*, vol. 22, no. 12, 2022. doi: 10.3390/s22124575. [Online]. Available: https://www.mdpi.com/1424-8220/22/12/4575 [234]

- J. Fernández, J. M. Cañas, V. Fernández, and S. Paniego, "Robust real-time traffic surveillance with deep learning," Computational Intelligence and Neuroscience, vol. 2021, p. 4 632 353, Dec. 2021. doi: 10.1155/2021/4632353. [Online]. Available: https://doi.org/10.1155/2021/4632353 [224]

Manuscripts Under Peer-Review: (visited in March 2024):

- Enhancing End-to-End Control in Autonomous Driving through Kinematic-Infused and Visual Memory Imitation Learning. Sergio Paniego, Roberto Calvo-Palomino, and José María Cañas

- Autonomous Driving in Traffic with End-to-End Vision-based Deep Learning. Sergio Paniego, Enrique Sinohara, and José María Cañas

Preprints:

- S. P. Blanco, S. Mahna, U. A. Mishra, and J. Canas, Memory based neural networks for end-to-end autonomous driving, 2022. arXiv: 2205.12124 [cs.RO]. [256]

I have also engaged in additional research endeavors, although they do not contribute to this thesis:

Workshop papers:

- P. F. de Cabo, R. Lucas, I. Arranz, S. Paniego, and J. M. Cañas, "RL-studio: A tool for reinforcement learning methods in robotics," in ROBOT2022: Fifth Iberian Robotics Conference, Springer International Publishing, Nov. 2022, pp. 502–513.doi: 10.1007/978-3-031-21062-4_41. [Online]. Available: https://doi.org/10.1007%2F978-3-031-21062-4_41. [257]

9.4. Future work

The presented thesis opens several avenues for future exploration based on the contributions described in this document. In this section, we will explore some of the possible future works that we would like to continue exploring.

9.4.1. Point-to-point end-to-end navigation using *input commands*

We have described and generated contributions of end-to-end autonomous driving models capable of driving following the lane autonomously and considering traffic in the scenario. Following this idea and extending it, we would like to work on the addition of further capabilities to the system while maintaining its simplicity. In this case, we would like to add the ability to understand *input commands* that indicate the action to be done in the following intersection or desired exit [172]. This action can be extracted from a set of fixed commands that the model receives and adapts its behavior based on them. These commands can be one of *continue straight*, *turn to the left*, *turn to the right*, and *follow the lane*. *Follow lane* is used for following continuously the lane in situations where intersections are not present and *continue straight* when an intersection is near. The model would continuously receive one of them based on the current situation and adapt. We have already explored this idea as part of a Google Summer of Code project developed in the summer of 2023 [245] generating a shallow model [12] with an image, ground-truth segmentation, and other measurements as input and with the controls as outputs (see Fig. 9.1

[12] https://www.youtube.com/watch?v=PsmpY6ZeT4I&ab_channel=JdeRobot

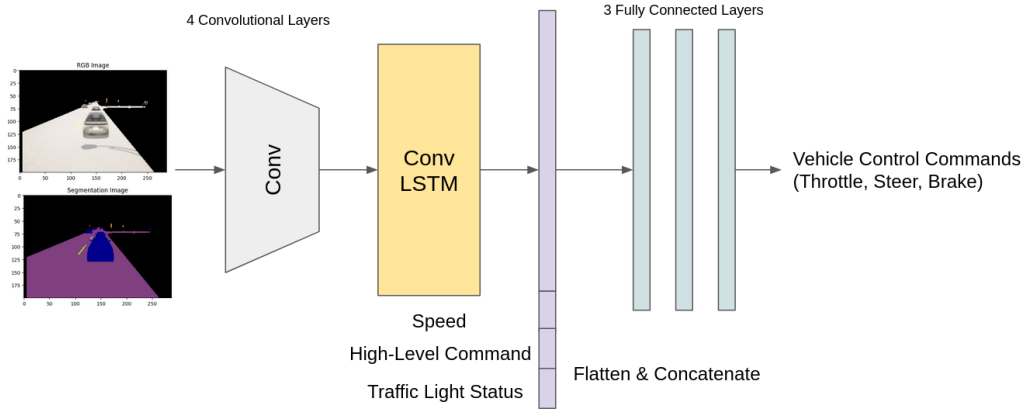


Figure 9.1: Architecture developed for point-to-point end-to-end navigation.

for a detail of the architecture). We continue developing the system considering other previous works that are current state-of-the-art that only use one camera as input [163], following the case of our development.

9.4.2. Transferring current end-to-end solutions to a real-world vehicle

The contributions presented in this thesis are developed in simulated environments. While we understand the importance and validity of such contributions, we would like to further develop and validate the current ideas in real-world scenarios. We would like to transfer the knowledge and contributions of the end-to-end model from simulation (see Fig. 9.2 for an example of the simulation with the vehicle) to a real vehicle (see Fig. 9.3) following the same philosophy of starting from a simple problem like a lane following scenario and then adding complexity to the environment. The end-to-end approach has experienced rapid growth in research and industry [26], as we have proved in this PhD thesis with several contributions so we would like to continue with this line transferring the extracted knowledge to a real-world vehicle. For this purpose, we are currently collaborating with the Autonomous Mobility and Perception Lab (AMPL) from Universidad Carlos III de Madrid. In this project, we are exploring the challenges and possibilities of developing end-to-end imitation learning models for autonomously driving in real-world scenarios with real 1:1 vehicles [258] [259].

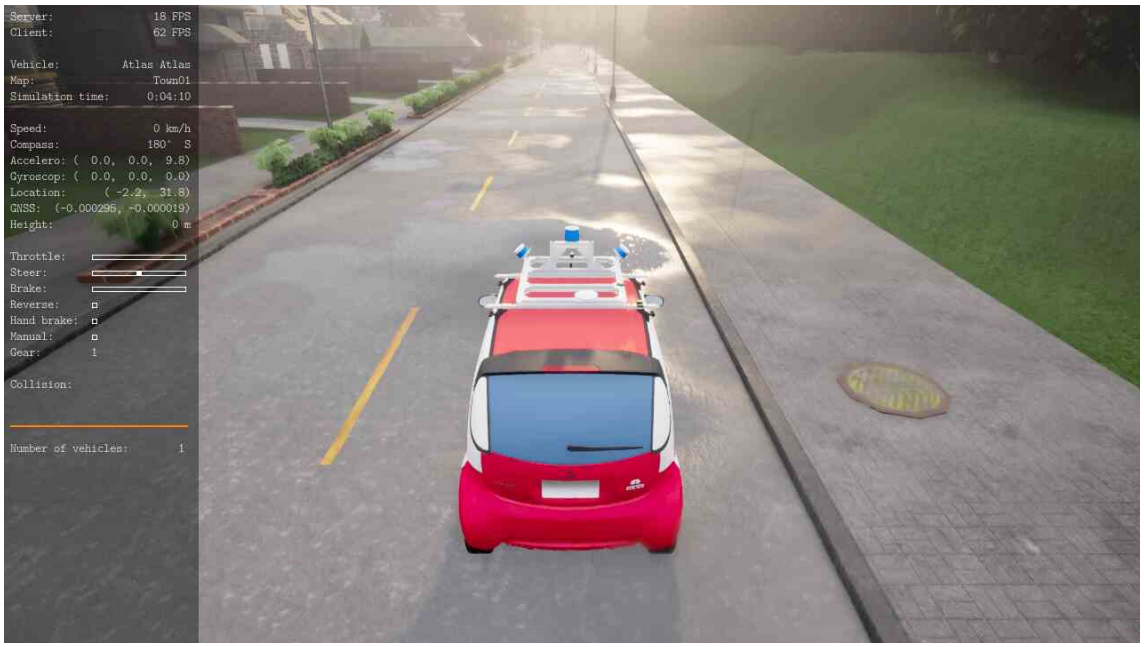


Figure 9.2: Simulation vehicle used for validating the solutions.

9.4.3. End-to-end autonomous vehicle driving modulated with text-based instructions

In recent years, a lot of impactful research projects related to natural language processing (NLP) and Large Language Models (LLMs) have been released. The revolution started with the introduction of Transformers [260] and from them, many LLMs have been developed like BERT [261], GPT-3 [262], Llama 2 [263] or Mistral 7B [264]. These models base their functionality on text data. We understand the importance of text as an input source for the modulation of end-to-end autonomous driving models and would like to explore it in a subsequent project. Several publications have already developed this idea like LMDrive [265], GPT-4V Takes the Wheel [266], LLM-Driver [267], or LingoQA [268]. The core idea consists of an end-to-end model that is capable of driving in an urban scenario and uses the user text inputs as a source for adapting its behavior. This idea is related to the commands of the previously presented (see Subsection 9.4.1) but for this one, we would like to even further explore the field and generate a system with deeper adaptation. This idea seems like a natural step in the development of a fully autonomous driving system that can be deployed in the streets and is close to what would be a common interaction of a taxi user with the taxi driver. While using a taxi, we usually command the driver directly speaking so it seems a natural direction of research exploration. Ideally, this future system should include speech recognition and natural language processing to understand the instructions and then modulate the driving system. We may leverage models like BERT [261] for understanding the natural language command and translate it to a fixed set of commands following the previously presented idea (see Subsection 9.4.1). After that step is completed, we may even consider more advanced models like Llama2, or Mistral. This project is expected to be addressed as part of Google Summer of Code

Figure 9.3: Real-world vehicle used for transferring the solutions.

2024 [13].

9.4.4. Exploration of end-to-end autonomous driving in aerial vehicles

Another line that we would like to explore after concluding this thesis is the usage of other types of autonomous vehicles that are controlled with the same end-to-end approach developed throughout this document. In this case, our target is aerial vehicles (e.g. drones). The idea of using an end-to-end system for driving drones has already been explored [269] [270]. We are currently exploring it in a bachelor thesis [14]. Our goal would be to translate our current end-to-end solutions to aerial vehicles and use the knowledge extracted during this PhD thesis for making a smooth adaptation to drones. These aerial vehicles have more degrees of freedom in comparison to ground vehicles and they have high-speed dynamics that are a challenge for developing a successful application

9.4.5. Exploration of end-to-end autonomous driving in unstructured environments

The current developments focus on urban highways and road scenarios, generating research that is useful in these particular conditions. For this future line, we would like

[13] https://jderobot.github.io/activities/gsoc/2024
[14] https://www.youtube.com/watch?v=jJ4Xdin1gg4&ab_channel=JdeRobot

to explore the usage of end-to-end systems in unstructured environments like forests. In those cases, the environmental complexity increased heavily and the solutions developed here would need to be rethought. We would also focus on the perception in these particular scenarios. The terrains are uneven and the perception becomes more complex since we have to consider vegetation (moss, leaves, forest...), different vehicles (caravan, truck, motorcycle...), or constructions (fences, bridges, tunnels...) among others. Some datasets for this approaches are already available like GOOSE [271] and Rellis 3D [272]We are currently working on this line inside a project by Agencia Estatal de Investigación de España (*(GAIA) Gestión integral para la prevención, extinción y reforestación debido a incendios forestales, Proyectos de I+D en líneas estratégicas en colaboración entre organismos de investigación y difusión de conocimientos TRANSMISIONES 2023. Ref PLEC2023-010303 (2024-2026)*).

9.4.6. Exploration of reinforcement learning approaches for end-to-end autonomous driving

The ideas presented in this thesis for building autonomous driving systems follow an end-to-end imitation learning approach. For this future avenue, we would like to explore reinforcement learning as a possible parallel path for also constructing autonomous driving systems and also combinations of reinforcement learning and deep learning, which is usually referred as deep reinforcement learning (DRL). Pure deep learning approaches exhibit some difficulties for autonomous driving in cases that are out of the data used for training. When a vehicle finds a situation not present in the training data, resulting in rapid performance degradation. To address this problem, the reinforcement learning approach can help explore the situations that are not recorded in the dataset [181]. We have already explored reinforcement learning in a previous contribution [257] in different settings and we would like to explore it further by developing the autonomous driving system and comparing its capabilities to the current end-to-end imitation learning model. For instance, using Behavior Metrics for far evaluation, which already has functionality for evaluating these types of approaches integrated. The development of end-to-end autonomous driving systems based on reinforcement learning [179] or inverse reinforcement learning [273] has already been explored previously, so we would build upon previous contributions. Again, the idea is to develop a solution, incrementally adding more capabilities.

Chapter 10

Resumen en castellano

El campo de la conducción autónoma busca generar vehículos que conduzcan de forma segura sin intervención humana. Este campo combina, entre otros, la inteligencia artificial, la visión por computador y la robótica. Ha recibido mucha atención en los últimos años, tanto en la academia como en la industria, y se prevé que esta tecnología tenga un impacto amplio en la vida diaria en los próximos años [1]. El desarrollo de este campo podría potencialmente generar beneficios en diferentes ámbitos, desde una mejora en la seguridad en el tráfico como una movilidad optimizada tanto para los individuos como para las mercancías. El desarrollo en este sector viene de diferentes ámbitos, por ejemplo de la inteligencia artificial, que en los últimos años ha experimentado un crecimiento enorme. Este progreso ha sido posible gracias al acceso a unidades de procesamiento gráfico (GPUs), la disponibilidad de conjuntos de datos de gran calidad y el desarrollo de algoritmos de aprendizaje profundo. Muchos campos se han beneficiado de este progreso, especialmente la robótica y la visión por computador.

La motivación principal de la adopción de la conducción autónoma es su potencial para mitigar el problema prevalente de los accidentes de tráfico en las carreteras. Según la NHTSA, el 94% de los accidentes de vehículos a motor fueron causados por error humano, según un estudio conducido entre 2005 y 2007 [2]. Otro de los problemas importantes es el número de accidentes de tráfico con heridos. Se estimaron 1.35 millones de muertes en 2016 causadas por accidentes de tráfico, según la WHO [3].

Existen una gran variedad de razones para promover la investigación en conducción autónoma. Podemos pensar en la reducción de estrés del conductor, la mejora de la productividad o de la movilidad al liberar a los humanos de la actividad de conducir. Los costes deberían reducirse al introducir la propiedad compartida de los vehículos como una opción más. La monitorización de tráfico también puede ayudar a la hora de reducir los accidentes de tráfico.

Algunos ejemplos actuales de compañías de conducción autónoma ya sugieren esa mejora en la seguridad. Por ejemplo, Waymo ha presentado un estudio en el que prueban que en 3.8 millones de millas conducidas (unos 6 millones de kilómetros), el Waymo

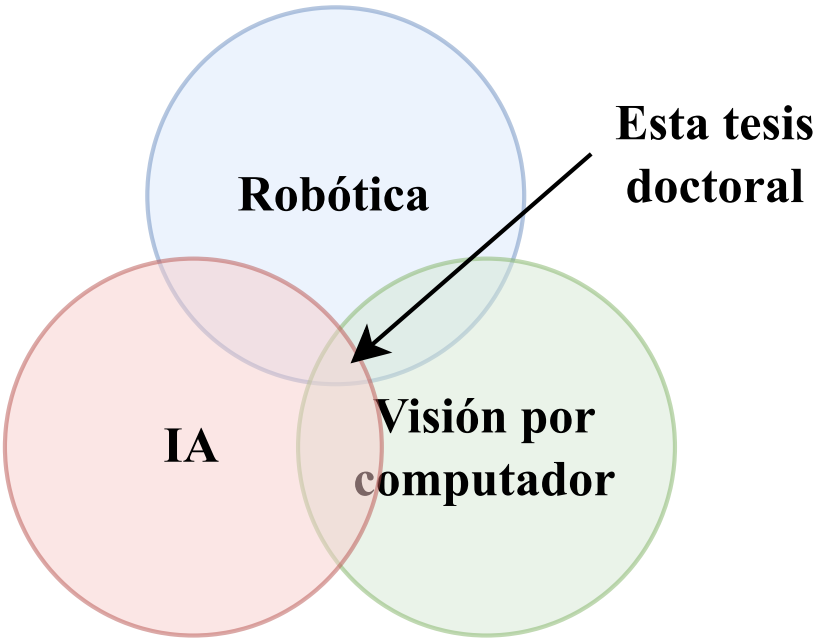


Figure 10.1: Diagrama de Venn con los campos que son el núcleo de esta tesis.

Driver se vio envuelto en 0 reclamaciones por lesiones corporales, en comparación con la base de los conductores humanos de 1.11 reclamaciones por millón de millas [4]. De forma similar, la empresa Cruise publicó un informe [5] en el que afirman que sus vehículos autónomos en San Francisco redujeron un 65% el número de colisiones en comparación con los conductores humanos.

Esta tesis de doctorado está enmarcada en la robótica impulsada por la inteligencia artificial con visión por computador, precisamente en el dominio de aplicación de la conducción autónoma. Por ese motivo, las contribuciones que se presentan buscan avanzar en este campo de investigación (ver Figura 10.1).

10.1. Introducción

En esta sección, proporcionamos un contexto sobre las ideas que son clave en esta tesis.

Conducción autónoma

El contexto sobre la conducción autónoma está dividido en diferentes temáticas que se consideran relevantes para sentar la base sobre la que presentar las contribuciones de investigación.

Los vehículos autónomos pueden considerarse robots móviles en cuanto al hardware y software que incluyen. A nivel de hardware, un coche moderno incluye sensores (GPS, LIDARs, IMUs, cámaras...) y actuadores avanzados (acelerador, volante, freno), como un robot. También incluye unidades computacionales como CPUs y GPUs, esenciales para

procesar los datos recolectados por los sensores.

Los niveles de autonomía de un vehículo autónomo se dividen en 6 categorías [7] según el estándar SAE J3016, que van desde el nivel 0 en el que no hay ningún tipo de automatización hasta el nivel 5, en el que el vehículo es completamente autónomo en cualquier situación.

En el nivel 0, no existe automatización. En el nivel 1, el vehículo realiza una tarea de forma autónoma como mucho en cada instante, como podría ser el control lateral o longitudinal. En el nivel 2, la automatización es parcial, pudiendo realizar dos tareas de forma autónoma simultáneamente. En el nivel 3, la automatización comienza a verse de forma más marcada, con una conducción autónoma en una gran variedad de situaciones, solamente requiriendo la intervención humana en casos extremos o cuando se detecta un fallo. En el nivel 4, la intervención humana no es necesaria, siendo el vehículo totalmente autónomo en ciertas zonas. En el nivel 5, el más avanzado, la automatización es total en todas las condiciones.

A día de hoy existen soluciones que implementan soluciones de nivel 4, como compañías tipo Waymo, Cruise o Wayve. Una de las soluciones más conocidas es la de los servicios de taxi autónomo [8]. Se anticipa que estas tecnologías avanzarán más allá en el futuro, extendiendo las capacidades y aplicación de los vehículos autónomos en los próximos años [12].

Los primeros ejemplos de vehículos autónomos datan de los años 80, con el de NavLab 1 [15] en 1986. A partir de ese primer desarrollo, muchos más ejemplos se han desarrollado. Son destacables los ejemplos del DARPA Grand Challenge [14] y el DARPA Urban Challenge, competiciones organizadas por DARPA y que fueron muy relevantes para el avance del campo, siendo su primera edición en 2004. En paralelo con el desarrollo de la conducción totalmente autónoma, también se han desarrollado los sistemas de asistencia al conductor (ADAS). Estos sistemas aparecen en la década de 1970. En el año 2021, la firma Canalys estimó que el 33% de los vehículos nuevos vendidos en los principales mercados incluían este tipo de tecnologías [16].

Dentro del campo de la conducción autónoma, podemos encontrar una gran diversidad de entornos de aplicación, cada uno de ellos teniendo sus aproximaciones particulares. Una idea común para todas estas aplicaciones es que la seguridad es un requisito clave y que los sistemas deben ser robustos ante una variedad enorme de tiempos atmosféricos, condiciones lumínicas o de tráfico. Podemos encontrar escenarios urbanos, carreteras o autovías, pero también escenarios no estructurados como bosques. En las contribuciones presentadas, nos centramos en escenarios urbanos, carreteras y autovías, que son probablemente los escenarios más comunes.

Otra idea a contemplar es la gran variedad de vehículos que existen y en las que las soluciones deben funcionar. Aunque la gran mayoría de la investigación está enfocada a

[15] https://www.youtube.com/watch?v=ntIczNQKfjQ

coches, existen aplicaciones en otros vehículos como autobuses o camiones [17].

En robótica ya existen soluciones avanzadas y confiables para la navegación robótica como Nav2 [18] que pueden ser utilizadas para generar una navegación entre puntos en diferentes configuraciones y escenarios. Esto nos puede hacer pensar en la necesidad de generar soluciones especializadas para conducción autónoma teniendo en cuenta que ya existen soluciones de navegación. Aunque las soluciones para navegación en robótica muestras capacidades muy destacables, pueden encontrar retos en entornos dinámicos que se caractericen por los movimientos rápidos, como los escenarios de conducción autónoma en la que se pueden encontrar vehículos robóticos conduciendo a velocidades altas (120km/h) con otros vehículos e incluso humanos. En estos entornos, la habilidad para responder de forma rápida de estos sistemas de navegación se puede ver comprometida, limitando su aplicación en conducción autónoma.

Evaluación y métricas

Dentro del campo del aprendizaje profundo, existen una gran variedad de tecnologías enfocadas en la evolución de soluciones derivadas de la investigación. Este proceso es fundamental para garantizar la robustez, eficacia y seguridad de las propuestas. Esta idea aplica a todas las áreas dentro del aprendizaje profundo, incluyendo las ideas que se presentan aquí con relación a la robótica, la visón por computador y la conducción autónoma. En el contexto de esta tesis, nos centramos en la evaluación de soluciones de visión por computador de detección de objetos y en la evaluación de soluciones de extremo a extremo para la conducción autónoma.

Aproximaciones extremo a extremo y modulares

En el contexto del desarrollo de soluciones para la conducción autónoma, existen dos aproximaciones principales: extremo a extremo y modular (ver Figura 10.2). La mayoría de soluciones a día de hoy implementan una aproximación modular, que consiste en una serie de módulos que se comunican entre ellos y que son especialistas en una parte de la conducción en concreto (percepción, mapeo...). En el otro lado tenemos las soluciones extremo a extremo, que desarrollan un sistema que transforma directamente la entrada de datos cruda en salidas de control en un solo paso hacia delante. En esta tesis, las contribuciones presentadas siguen la segunda aproximación, ya que buscamos la simplicidad y la eficiencia a la hora de desarrollar una solución de conducción autónoma.

La primera aproximación es más propensa a propagar errores ya que un error en uno de los módulos puede afectar negativamente al resto de los subsistemas y a su salida final. Además estos sistemas son más complejos. Como ventaja, son más interpretables y fáciles de depurar. En contraposición, los modelos extremo a extremo suelen ser más simples y eficientes gracias a la combinación de tares en un mismo modelo. Además, están optimizados para una tarea común en comparación con la aproximación modular.

Tareas en un sistema de conducción autónoma

Dentro de un sistema de conducción autónoma, existen una gran variedad de subtar-

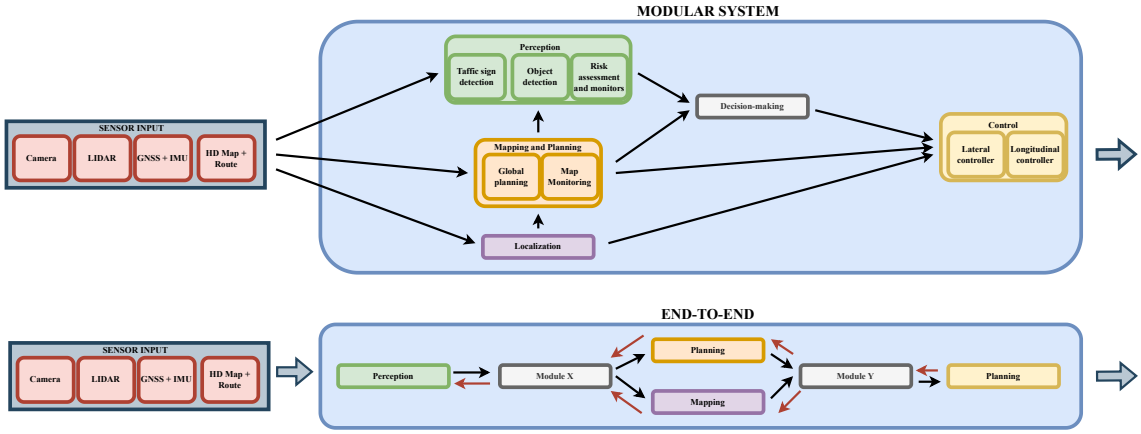


Figure 10.2: Diagrama de las aproximaciones extremo a extremo y modulares. Adaptado desde [26] y [25].

eas, que se combinan para formar el sistema de conducción autónoma final. Estas tareas pueden incluir desde seguir el carril, pasando por la negociación de intersecciones o el aparcamiento autónomo. En esta tesis, construimos nuestras contribuciones desde la tarea básica de seguimiento del carril, añadiendo incrementalmente más complejidad y prestaciones al sistema resultante y ampliando su área de aplicación.

Optimización de las soluciones de conducción autónoma

Dentro de la robótica y en especial en la conducción autónoma, las soluciones tienen que ser muy rápidas y fiables para poder ser útiles, ya que los vehículos van a velocidades altas, en entornos muy dinámicos y que incluyen otros vehículos o humanos. El rendimiento de una aplicación robótica no solo depende de la calidad de las decisiones de control, sino también de su frecuencia de iteraciones. Idealmente, buscamos generar decisiones de gran calidad a un ritmo muy alto. Algunos sistemas tienen hardware de altas capacidades que puede realizar este trabajo ágilmente de forma natural, pero existen otros entornos que no cuentan con estas características. Una solución podría ser optimizar los modelos neuronales utilizados para mejoras sus prestaciones, algo que es típico en los sistemas de deep learning en la conducción autónoma. Esto es algo que se aborda en esta tesis y que genera una de las contribuciones.

Monitorización del tráfico rodado con visión por computador

La monitorización del tráfico rodado juega un papel central en la planificación urbana, en el manejo del transporte y en la seguridad pública. Esta monitorización busca clasificar, identificar y medir la velocidad de diferentes tipos de vehículos. Este campo también sirve como puerta de entrada para entender la utilidad de las soluciones de conducción autónoma en la mejora de la seguridad y del transporte en general.

10.2. Objetivos

El objetivo final es el progreso en el campo de la conducción autónoma y con él, en los campos de la visión por computador, la inteligencia artificial y la robótica. Este objetivo general lo articulamos en siete subojetivos concretos. Los subobjetivos de investigación son:

- **[RG1] Estudiar el estado de la cuestión en el campo de la conducción autónoma, enfocado en los sistemas de extremo a extremo y campos relacionados.**
- **[RG2] Validar la visión por computador en un entorno de conducción generando una herramienta de monitorización del tráfico rodado.**
- **[RG3] Generar una herramienta software de evaluación masiva, automática y cuantitativa de soluciones de detección de objetos.**
- **[RG4] Generar una herramienta software de evaluación activa de soluciones de conducción autónoma para realizar experimentos y generar datos cuantitativos sobre su rendimiento.**
- **[RG5] Generar agentes de conducción autónoma extremo a extremo visual para seguimiento del carril que mejoren su comportamiento basándose en añadir memoria visual y entrada cinemática a modelos de aprendizaje profundo con aprendizaje por imitación.**
- **[RG6] Desarrollar modelos extremo a extremo de control optimizados para el seguimiento de carril visual que se aprovechen de las últimas técnicas de optimización en aprendizaje profundo.**
- **[RG7] Mejorar el comportamiento de control de un vehículo que conduzca de forma autónoma para el sigue carril visual con tráfico, es decir, en presencia de otros vehículos.**

10.3. Antecedentes

Esta tesis doctoral se construye encima de la base de conocimiento existente en conducción autónoma y el resto de campos relacionados. En esta sección realizamos una revisión breve del estado de la cuestión en aquellos campos de interés para las contribuciones presentadas.

10.3.1. Monitorización del tráfico rodado

Un problema clásico en visión por computador es la monitorización del tráfico rodado [27] [28] [29] [30] [31]. Esta tarea implica la clasificación y seguimiento de vehículos en imágenes, por ejemplo en autovías, para monitorizarlos. Antes de la aparición

del aprendizaje profundo, la vigilancia del tráfico utilizando cámaras estaba significativamente limitada, centrándose en tareas rudimentarias como la monitorización pasiva o el procesamiento básico automático [38] [39] [40]. Sin embargo, la emergencia del aprendizaje profundo ha supuesto un punto de inflexión, llevando a un avance significativo [41] [42] [43].

Como el objetivo de la monitorización de tráfico implica la localización y clasificación de objetos, la utilización de técnicas de detección de objetos se vuelve imperativa [75] [76]. Numerosos estudios dentro del ámbito de la monitorización del tráfico se han centrado en la detección de objetos, aprovechando predominantemente la metodología de aprendizaje profundo [77] [78] [79] [80] [81] [82] [83] [84] [85] [86].

10.3.2. Detección de objetos con aprendizaje profundo, conjuntos de datos y evaluación

El campo de la detección de objetos ha experimentado una popularidad muy significativa en los últimos años, debida a los notables avances y la proliferación de revisiones del estado del arte sobre la materia en la literatura [87] [88] [89] [90] [91] [92] [93] [94]. El campo de aplicación de la detección de objetos es muy amplio, por ejemplo se utiliza con éxito en soluciones en conducción autónoma [95] o en detección de caras [96].

Antes de la adopción de las técnicas de aprendizaje profundo, la detección de objetos se basada en técnicas tradicionales que principalmente utilizaban métodos de procesamiento de imágenes y características específicas de los objetos a identificar en las imágenes [98].

Análogo a los avances presenciados en la investigación de conducción autónoma, el progreso este campo es debido también a la aparición de conjuntos de datos de gran calidad, como COCO [99], ImageNet [100], Pascal VOC [101], Princeton [102], Spinello [103] u Open Images Dataset [104]. En conducción autónoma, tanto la detección de objetos como la segmentación semántica se utilizan en la parte de percepción del sistema. Cada una de estas tareas de visión por computador tiene sus propias métricas que sirven para medir de forma efectiva el rendimiento de posibles soluciones y realizar una comparación justa con respecto a otras aproximaciones. Estas métricas incluyen la precisión, la recuperación, la exactitud, IoU, mAP, mAR...

Todas las soluciones que ahora mismo forman parte del estado de la cuestión confían en las metodologías de aprendizaje profundo. Estas metodologías han sido indispensables en esta tesis doctoral, utilizando especialmente los *frameworks* PyTorch [124] y Tensorflow [120] en la mayoría de las contribuciones de investigación. Algunos de los modelos del estado de la cuestión de detección de objetos incluyen Faster Regional-CNN [125], Single Shot MultiBox Detector [126] [127] [128] o You Only Look Once [129], con sus mejoras incrementales con YOLOv3 [131], YOLOv4 [132], YOLOv5 [133], YOLOv9 [134]...

10.3.3. Conducción autónoma y aprendizaje por imitación

El campo de la conducción autónoma combina inteligencia artificial, visión por computador y robótica. Como ya hemos mencionado, las soluciones en este campo suelen dividirse en dos grupos: modulares o de extremo a extremo. La solución más extendida es la modular, en parte gracias a su flexibilidad [25]. Una de las críticas más recurrentes a esta aproximación es que los errores en los módulos pueden llevar a errores en cascada.

Uno de los ejemplos de código abierto más relevantes dentro de la aproximación modular es Autoware [145], que está construida utilizando ROS e incluye módulos para cada una de las actividades de conducción autónoma como percepción, planificación, control...

En cambio, las aproximaciones extremo a extremo [147] [148] [149] [150] [151] traducen directamente los datos crudos dados por los sensores del vehículo en comandos de control finales que lo manejan. Las soluciones extremo a extremo basadas en visión han ganado mucha atracción en la literatura reciente, pero aún presentan limitaciones [154] como la poca explicabilidad.

PilotNet [159] es un modelo de aprendizaje profundo para el control extremo a extremo basado en visión desarrollado por investigadores de NVidia. Esta solución emplea una red neuronal convolucional [160] para extraer las características y generar los comandos de control desde la entrada de imágenes. A partir de este modelo relativamente simple, otros modelos más actuales, complejos y potentes se han propuesto, como TCP [163] o ReasonNet [164].

Aprendizaje por imitación

El aprendizaje por imitación [168] consiste en la generación de una política de conducción aprendida a partir de los datos recolectados de un agente experto (ver Figura 10.3). Una técnica relevante dentro del aprendizaje por imitación es la clonación de comportamiento. En esta técnica, uno o más agentes expertos conducen mientras los datos que generan se guardan. Esto significa que los datos sensoriales y las acciones (controles) son guardadas. Con esos datos, se entrenan modelos de aprendizaje profundo que imitan su comportamiento [171]. Existe una gran variedad de casos de éxito de esta técnica dentro de la conducción autónoma [172] [165] [166] [167] [163] [164].

Generar un modelo de calidad necesita una gran cantidad de datos diversos. A la hora de recolectarlos, esto no suele ser así, ya que los datos que se guardan suelen ser de situaciones genéricas, sin mucho valor añadido, lo que puede conducir a generar conjuntos de datos desbalanceados. Además, el rendimiento de un vehículo autónomo se deteriora rápidamente en aquella situaciones que no estén reflejadas dentro de los datos (ver Figura 10.4). Existen técnicas para mejorar este hecho, como DAgger [174] [175]. Los vehículos también se pueden conducir utilizando aprendizaje por refuerzo [177] [178] [148] [149] [179] o su combinación con aprendizaje profundo, llamada aprendizaje por refuerzo profundo [180]. La combinación de aprendizaje por imitación y aprendizaje profundo también muestra una línea prometedora [181].

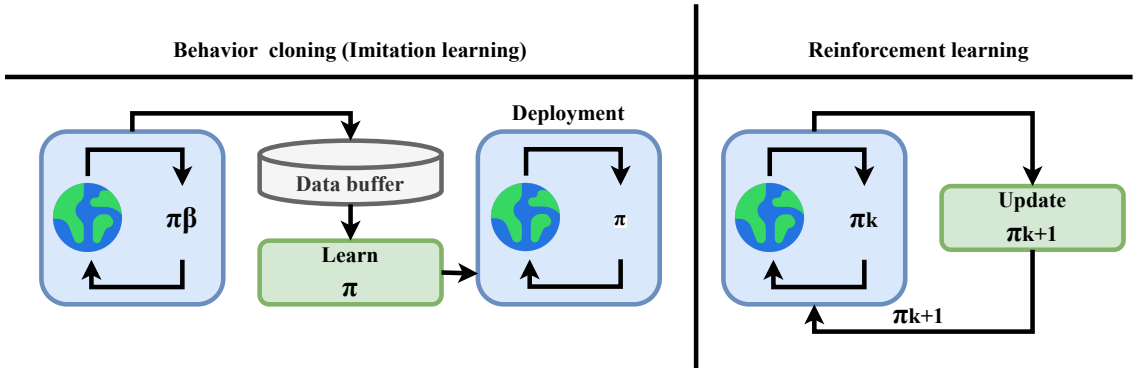


Figure 10.3: Diagrama de la clonación de comportamiento (aprendizaje por imitación) y del aprendizaje por refuerzo. Adaptado de [26].

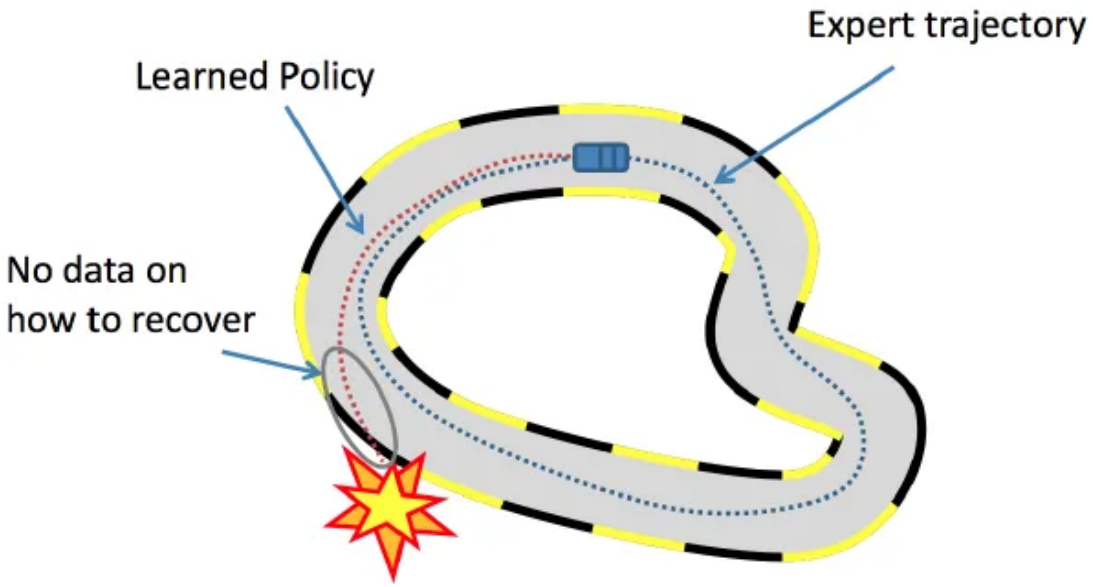


Figure 10.4: La política aprendida presenta problemas en situaciones no vistas durante el entrenamiento.

10.3.4. Simulación en conducción autónoma, conjuntos de datos y evaluación

En el desarrollo e investigación en sistemas robóticos, los simuladores son comúnmente utilizados para generar soluciones en lugar de hacerlo con robots o vehículos reales [164] [163]. Esto permite un desarrollo iterativo y barato, además de facilitar la depuración y la validación de los sistemas. Dentro de la conducción autónoma, existen una gran variedad de simuladores, entre los que destacamos destacamos Gazebo [184] y CARLA [186], aunque existen alternativas como SUMO [182], TORCS [183], DeepDrive [188], Baidu Apollo [189], Autoware [145], AirSim [190] o el simulador de conducción autónoma de Udacity [191]. Los simuladores permiten generar datos sintéticos que se pueden utilizar para generar conjuntos de datos accediendo a las etiquetas verdaderas de un modo rentable. También permiten evaluar soluciones. Algunos simuladores, como CARLA y

Gazebo, son compatibles con ROS [194] [195], el middleware de código abierto estándar en robótica.

Dentro de los conjuntos de datos, existe una gran variedad de ellos dentro del campo de la conducción autónoma. Entre ellos, para percepción visual destacan nuScenes [196], BDD100K [197], KITTI [198] [199] o Cityscapes [200], para planificación nuPlan [201] o para seguimiento de carril commaAI [158] o el conjunto de datos de Udacity [202]. Estos conjuntos de datos, junto con otros [203] [204], se han convertido en *benchmarks* para los investigadores y desarrolladores.

Dada la gran variedad de escenarios y tareas de tráfico, además de las necesidades de seguridad [205], son necesarias métricas de evaluación robustas. Las métricas comunes pueden no ser suficientes (MSE, accuracy...) en este tipo de situaciones, ya que estas métricas fuera de línea podrían fallar en capturar las dinámicas de las tareas de conducción a lo largo del tiempo. Por eso, es necesario tener un marco de trabajo complementario a estas métricas, para poder realizar una evaluación eficaz. Esta cuestión ya ha sido abordada en la literatura y es un problema conocido [206] [207] [208] [209]. Por ejemplo, con CARLA tenemos en CARLA Autonomous Driving Leaderboard [16] que es un marco de evaluación (también una competición asociada) que evalúa y hace un ranking de las soluciones de conducción autónoma en una gran variedad de tareas y situaciones. Esta forma de evaluación puede resulta demasiado gruesa o inadecuada para determinados problemas, presentando limitaciones.

10.3.5. Aproximaciones de conducción autónoma extremo a extremo basadas en memoria

Una línea de investigación dentro de la aproximación extremo a extremo busca añadir capacidades de memoria a los modelos de control. Esta memoria se puede manifestar dentro de la arquitectura de aprendizaje profundo por medio de módulos dedicados de memoria (LSTMs, ConvLSTMs...) [212] [213] [214] o directamente dentro de la entrada de datos (imágenes concatenadas, combinación de imágenes...) [13].

10.3.6. Optimización de los modelos de aprendizaje profundo para conducción autónoma

Las soluciones basadas en visión para conducción autónoma normalmente se generan utilizando aprendizaje profundo. Estas soluciones son muy exigentes a nivel computacional. Un componente importante en estos sistemas es el hardware disponible porque el rendimiento del robot móvil depende, no solo de la calidad de las decisiones sino también de su frecuencia. Algunos vehículos autónomos tienen acceso a hardware de gran capacidad, pero otros no. Por esto, se pueden utilizar técnicas de optimización de aprendizaje

[16] https://leaderboard.carla.org/

profundo para conseguir modelos más pequeños y rápidos, pero con la misma calidad en las decisiones que sus homólogos sin optimizar [217].

Hay muchas técnicas disponibles, desde cuantización [218] [219], pasando por la poda [220], el ajuste fino con técnicas de optimización consciente o la agrupación de partes. Los *frameworks* de Pytorch y Tensorflow incluyen estas optimizaciones [120] [221] [124] con diferentes niveles de soporte e incluso existe optimizaciones enfocadas a los dispositivos, como el uso de marco de trabajo de TensorRT para GPUs Nvidia [222].

10.4. Metodología y resultados

En esta sección, se presentan las contribuciones que son el núcleo de esta tesis doctoral. Cada una de ellas constituye un proyecto autocontenido y un artículo de investigación en el objetivo final de avanzar en el campo de la conducción autónoma. Para realizar la presentación de las contribuciones, describimos la metodología y resultados de cada una de ellas.

10.4.1. Monitorización del tráfico rodado con aprendizaje profundo

La monitorización en tiempo real del tráfico en autovías, carreteras o calles puede generar datos interesantes para el planeamiento urbano y para el manejo del tráfico en general. Este área ha tenido un gran desarrollo en los últimos años gracias al desarrollo del aprendizaje profundo. En esta subsección se presenta TrafficSensor, una de las contribuciones de la tesis. Esta herramienta software de código abierto emplea técnicas de aprendizaje profundo para hacer clasificación y seguimiento automático de vehículos utilizando una cámara calibrada y fija al lado de una carretera en tiempo real. Esta herramienta marca la evolución desde una versión previa con técnicas de visión artificial clásica, llamada TrafficMonitor [98], consiguiendo una mayor fiabilidad y rendimiento incluso en condiciones de iluminación o tiempo atmosférico peores.

Para conseguir este propósito, un nuevo conjunto de datos se ha creado para entrenar los modelos, que incluye imágenes reales de tráfico en condiciones lumínicas malas o en tiempo atmosférico adverso, además de imágenes en baja resolución. Este conjunto de datos incluye siete categorías diferentes de vehículo para clasificar, formando un total de 9774 imágenes. Las categorías de vehículos son: motocicleta, automóvil, furgoneta, autobús, camión, camioneta y camión cisterna.

El sistema propuesto consiste principalmente de dos módulos, uno responsable de la detección y clasificación y el otro del seguimiento de los vehículos en el tiempo. Después de una fase de test y comparación en la que se evaluaron diferentes modelos de detección de objetos, en concreto SSD MovileNetV2, SSD VGG-16, YOLOv3 y YOLOv4, estas dos últimas opciones fueron seleccionados como modelos para realizar la detección. La evaluación de los modelos se realizó en diferentes ambientes, como en condiciones de luz

pobres o de tiempo atmosférico adverso.

En la parte del sistema dedicada a seguir los vehículos, se combina un algoritmo de asociación espacial simple con un rastreador más sofisticado KLT para seguir a los vehículos en la carretera. Ambos sitemas se ejecutan dentro de un área de las imágenes de entrada designada por el usuario que se denomina área de evaluación. Varios experimentos se realizaron utilizando vídeos de tráfico exigentes para validar el sistema con datos reales. Los resultados del sistema experimental demuestran que el sistema propuesto detecta, rastrea y clasifica los vehículos en tiempo real en autovías de forma exitosa.

Para realizar la evaluación de las soluciones de aprendizaje profundo entrenadas, se utilizó la herramienta Detection Metrics, que se detalla en la siguiente subsección y en el Capítulo 4.

Esta investigación se ha publicado en un artículo científico [224] y el software de código abierto que hemos descrito también está disponible para replicarlo y extenderlo [225]. Más detalles sobre esta contribución y sobre el sistema de TrafficSensor en el Capítulo 3. Esta contribución marca el paso inicial hacia la conducción autónoma, comprendiendo las implicaciones de seguridad del campo y cómo puede ser mejorado a través de este tipo de herramientas basadas en aprendizaje profundo.

10.4.2. Evaluando arquitecturas de aprendizaje profundo para detección de objetos con métricas cuantitativas

Después de haber explorado la monitorización del tráfico con técnicas del estado de la cuestión, dentro de esta contribución se presenta y desarrolla una herramienta para la comparación y evaluación objetiva, masiva, cuantitativa y desatendida de modelos de detección de objetos. Esta herramienta es Detection Metrics, que ya fue utilizada en la contribución anterior para la parte experimental y que constituye una de las contribuciones de esta tesis doctoral. El rango de aplicación de esta aplicación no está limitado a la monitorización de tráfico rodado, sino que también se puede utilizar dentro de la percepción de los sistemas de conducción autónoma con visión a bordo. Esta contribución ha sido publicada en un artículo científico [234].

Dentro de la visión por computador, una de las tareas principales es la detección de objetos, que implica la localización y clasificación de objetos dentro de imágenes de forma precisa, como ya se ha utilizado en la contribución anterior. En los últimos años, se ha experimentado un gran avance en este campo, gracias a la disponibilidad de conjuntos de datos de gran calidad, la evolución de las arquitecturas de aprendizaje profundo, como las redes convolucionales profundas y la accesibilidad a hardware poderoso mediante las GPUs. El desarrollo de soluciones de deep learning suele implicar experimentación y desarrollo iterativo, ajustando los hiperparámetros de los modelos y realizando ajuste fino para optimizar el rendimiento. A partir de estas necesidades, presentamos Detection Metrics, una aplicación software multiplataforma que da métricas de rendimiento objetivas

y permite a los investigadores evaluar de forma sistemática diferentes modelos de aprendizaje profundo en conjuntos de datos de forma efectiva.

Detection Metrics ofrece soporte para los conjuntos de datos de detección de objetos más relevantes internacionalmente (COCO, ImageNet, Pascal VOC, Princeton RGB, Spinello y Open Images), así como los entornos de desarrollo de aprendizaje profundo más utilizados (TensorFlow, Keras, PyTorch, Caffe2 y Darknet). Permite una comparación de diferentes modelos neuronales implementados utilizando distintos entornos de desarrollo de forma justa. Este proceso es útil para el desarrollo de aplicaciones de aprendizaje profundo o para investigación. Se proporcionan varias herramientas al investigador para manejar y trabajar con diferentes conjuntos de datos y modelos, incluyendo la visualización y la conversión entre diferentes formatos comunes.

La herramienta ofrece dos formas de lanzarse principales: por medio de la interfaz gráfica (GUI) o sin ella (headless). El modo gráfico lanza una aplicación con la que el usuario interactúa para acceder a todas las funcionalidades de la aplicación. Mientras que para la segunda se lanza a partir de la consola con un archivo de configuración y permite la evaluación objetiva, masiva, cuantitativa y desatendida. La aplicación proporciona seis funcionalidades diferentes que son: Visualizador, Evaluador, Detector, Desplegador, Etiquetador y Conversor. Cada una de ellas se puede utilizar de forma individual o combinada.

Esta herramienta automatiza el proceso de validación experimental, lanzando los procesos en lotes y ahorrando tiempo al investigador. Permite crear nuevos conjuntos de datos de dominio específico a partir de la entrada de vídeo o webcam. Es de código abierto [235], por lo que puede ser auditada, extendida o adaptada para las necesidades particulares del proyecto.

Para la validación experimental de la herramienta, ya se ha descrito en la subsección anterior que fue utilizada para su validación. Además de esta, se llevó a cabo una validación experimental comparando el rendimiento de alguno de los modelos del estado de la cuestión más relevantes en detección de objetos, implementados en diferentes *frameworks*, comparando los resultados obtenidos con los dados por los propios investigadores en los artículos de investigación. Los modelos eran SSD Inceptionv2, YOLOv3, Faster RCNN Resenet101 y FasterRCNN Resnet50 FPN. En esta validación se compararon los resultados obtenidos utilizando Detection Metrics con respecto a los que proporcionaban los desarrolladores para cada uno de ellos. Se encontraron resultados consistentes, lo que subraya la efectividad de esta herramienta para la comparación objetiva de modelos de detección de objetos, incluso cuando provienen de diferentes *frameworks*.

Para conocer todos los detalles sobre esta contribución, se puede revisar el Capítulo 4.

10.4.3. Evaluación de comportamientos de conducción autónoma con métricas de grano fino

Behavior Metrics [238] es otra contribución de esta tesis doctoral y ha originado una publicación en una revista científica [239]. Es una herramienta software que ha sido desarrollada para ayudar en el desarrollo del campo de la conducción autónoma. El desarrollo y validación activa de soluciones de conducción autónoma requiere de la realización de pruebas de forma amplia en simulación. Para abordar este requerimiento, desarrollamos el software Behavior Metrics para la evaluación cuantitativa y cualitativa de soluciones de conducción autónoma para diferentes tareas de conducción. Este software provee dos canales principales de evaluación, en la misma línea que en la contribución anterior: una con interfaz gráfica de usuario (GUI) y otra sin interfaz gráfica para el test masivo y desatendido de soluciones.

Esta herramienta genera una serie de métricas cuantitativas, incluyendo métricas de grano fino para cada tarea de conducción particular (seguimiento de carril, conducción en tráfico...) que complementan a las proporcionadas por el propio simulador. Proporciona un entendimiento profundo y amplio del rendimiento de las soluciones y permite su comparación y mejora.

Tiene soporte para diferentes simuladores de conducción autónoma, en concreto Gazebo y CARLA. Gracias al soporte de CARLA, que utiliza Unreal como motor, se proporcionan simulaciones urbanas fotorrealistas, permitiendo generar soluciones con unas condiciones más cercanas a los entornos reales. Para comunicarse entre la aplicación y el simulador, se utiliza el *middleware* de robótica ROS, que facilita la integración de la aplicación con otros simuladores o con posibles vehículos reales. Además de esto, soporta los principales entornos de desarrollo de aprendizaje profundo PyTorch y Tensorflow. Behavior Metrics es de código abierto y multiplataforma gracias a la tecnología Docker.

Para la ejecución de los experimentos, estos se definen en un archivo YAML en el que se define el escenario, la tarea, el controlador del vehículo... A partir de esta información, los experimentos se ejecutan y se generan métricas cuantitativas que son guardadas en ficheros para su posterior análisis. Además, se proporcionan análisis preliminares como mapas del recorrido de los vehículos, colisiones... para que el usuario pueda comprender el rendimiento de la solución de forma sencilla. En cuanto a los escenarios, se soportan aquellos proporcionados por el simulador y se permite incluir nuevos de forma sencilla. En este caso el simulador principal es CARLA, así que se incluye soporte para todos sus mapas, vehículos... En cuanto a las tareas, se soporta el seguimiento del carril, la conducción con tráfico y la navegación entre puntos. Cada una de estas tareas tiene sus propias métricas de evaluación especializadas (porcentaje de recorrido realizado, error espacial respecto a la trayectoria ideal, distancia con otros vehículos...), además de algunas comunes como la distancia recorrida o el número de colisiones.

En cuanto al controlador de los vehículos, se trata de la pieza principal que se encarga de manejar y controlar el comportamiento del vehículo en todo momento. Se basa

en las entradas sensoriales que el vehículo recibe, las procesa y a partir de ese procesamiento genera las decisiones de control finales que se traducen en comandos de aceleración o giro. Como entradas sensoriales, se pueden definir aquellos sensores que monta el vehículo en la configuración del experimento y se soportan todos los que el simulador proporciona, además de otros personalizados como la vista de pájaro. Estos sensores, como el resto del sistema, son fácilmente adaptables o extensibles, dada la naturaleza de código abierto del proyecto. El núcleo del controlador del vehículo puede ser un modelo de aprendizaje profundo, una política generada por aprendizaje por refuerzo o incluso un piloto programado de forma explícita.

En el modo de evaluación con interfaz gráfica, se genera una interfaz a partir de un archivo de configuración con la que el usuario interactúa para controlar el experimento y visualizar el estado de los sensores. Además, puede seguir con la vista de la simulación el desarrollo del experimento en tiempo real. Por otro lado, en el modo desatendido, toda la configuración del experimento se describe en un fichero de configuración, pudiendo evaluar una gran variedad de controladores en diferentes entornos de forma desatendida. La aplicación generará datos de evaluación tanto para cada uno de ellos como de forma agregada, brindando al investigador una visión integral del comportamiento de sus soluciones.

Esta herramienta ya ha sido utilizada para la realización de contribuciones en conducción autónoma, mostrando experimentalmente su utilidad y validez, como se desarrolla en la Subsección 10.4.4, Subsección 10.4.5 (artículo publicado en una revista científica [244]) y Subsección 10.4.6. Además de eso, se ha utilizado para la evaluación objetiva de soluciones en un proyecto de código abierto dentro de la iniciativa de Google Summer of Code [245] sobre navegación entre puntos. Para conocer todos los detalles sobre esta contribución, se puede revisar el Capítulo 5.

10.4.4. Mejora del control de extremo a extremo en conducción autónoma mediante entrada cinemática y arquitecturas basadas en memoria

En esta subsección se explica una de las contribuciones de la tesis [246], que está bajo revisión por pares en el momento de escribir este documento [247]. La base de esta contribución es la generación de varias aproximaciones que mejoran las capacidades de un sistema de conducción autónoma de extremo a extremo basado en aprendizaje por imitación, añadiendo memoria visual y entrada de datos cinemática (velocidad del vehículo). La comparación entre ellas y la aproximación base se basa en métricas fundamentales de error (MSE, MAE) y varias métricas de grano fino externas, complementarias, basadas en el comportamiento del vehículo controlado en varios escenarios de tests dentro del simulador CARLA e integradas en Behavior Metrics.

Esta investigación se centra en la aplicación de seguimiento de carril utilizando diferentes diseños urbanos y también se basa en el uso de la vista de pájaro como datos de entrada para el modelo que controla el vehículo. Las adiciones de memoria cubren tanto

modificaciones arquitectónicas como diferentes tipos de entrada sensorial compuesta. Dentro de la parte experimental, se muestra y valida la idea de que incorporar capacidades de memoria visual y entrada de datos cinemática hace que el sistema sea más robusto. Esto le permite hacer frente a una mayor variedad de situaciones desafiantes, teniendo en cuenta la reducción de colisiones experimentada y la autorregulación de la velocidad que lleva el vehículo. Como en las subsecciones anteriores, todo el trabajo presentado es código abierto, incluyendo la arquitectura de los modelos, los pesos generados a partir del entrenamiento de los modelos, la herramienta de comparación y el dataset.

El estudio que se presenta en esta contribución se basa en las implicaciones en el comportamiento final del comportamiento robótico de la adición de memoria visual y entrada de datos cinemáticos al modelo. La entrada de datos de vista de pájaro consiste en una imagen segmentada que elimina parte de la complejidad que el sistema debería procesar, como podrían ser las texturas, para que se enfoque en las partes relevantes para este estudio. Se presentan y prueban 4 modelos y una variante de cada uno de ellos, con la entrada de vista de pájaro ya mencionada y con una salida de comandos de control del vehículo que incluye la aceleración, giro del volante y el freno.

En concreto, se utiliza una arquitectura sin memoria que está basada en la arquitectura PilotNet [159], llamada PilotNet*, cambiando la salida a la que se acaba de mencionar con tres valores en lugar de la arquitectura base que cuenta con solo una salida. Por otro lado, se utilizan tres arquitecturas con memoria visual en su arquitectura, con la misma salida anterior. En primer lugar, DeepestLSTMTinyPilotNet*, que se basa en DeepestLSTMTinyPilotNet [213] que cuenta con capas ConvLSTM para procesar los datos y mantener una memoria de lo ya visto. En segundo lugar, PilotNet*x3(Conv3D), que es una variante de PilotNet, con módulos Conv3D que sustituyen a las capas Conv2D y que recibe tres imágenes de entrada temporalmente sucesivas, en lugar de una. En tercer lugar, el modelo PilotNet*x3(TimeDistributed) también recibe tres imágenes simultáneamente. Este modelo fusiona la información procesada de estas imágenes con las características extraídas, y luego utiliza módulos LSTMs para procesar la información temporal.

Además de estas cuatro arquitecturas, se presenta una variante de cada una de ellas incluye la entrada cinemática de la velocidad instantánea que lleva el vehículo en el momento dado. Esta entrada cinemática se incluye como un nuevo canal en las imágenes de entrada, siendo todos sus valores la velocidad normalizada. Para entrenar estos modelos, se utiliza un dataset sintético extraído a partir del comportamiento de un piloto experto en unos circuitos de entrenamiento del simulador CARLA. Este piloto experto conduce por los diferentes circuitos de entrenamiento mientras se graban sus acciones y el estado de los sensores. La validación experimental se realiza en otros circuitos que consideramos de test y que no han sido nunca vistos por los agentes. Durante el entrenamiento, se incluyen técnicas que buscan mejorarlo, como aumentado de datos (en donde se vio que el aumentado *Affine* es de gran importancia para la mejora de los resultados) y la sobrerrepresantación de los datos de mayor relevancia (datos de curvas, por ejemplo).

En cuando a la validación experimental realizada, para esta contribución se presentan seis experimentos diferentes, donde el primero se hace fuera de línea y el resto de ellos en línea, utilizando Behavior Metrics con el simulador de conducción autónoma CARLA. El primero de ellos compara los modelos utilizando las métricas típicas en aprendizaje automático, como MAE o MSE, donde se ve que los modelos con entrada cinemática tienen una reducción del 86% en MAE y 53% en MSE con respecto a la base. El segundo experimento busca hacer validación en un escenario de test, haciendo que los controladores de los vehículos no superen una velocidad máxima de 30km/h. Este límite se introduce porque es la velocidad máxima seleccionada para el piloto experto que se utiliza para extraer los conjuntos de datos. Las diferencias en este punto son pequeñas para la variedad de modelos, pudiendo completar todos ellos los experimentos a una velocidad media parecida al resto y sin colisiones. En el tercer experimento, esta limitación de la velocidad máxima se elimina, para probar si los modelos son capaces de regular la velocidad del vehículo de forma autónoma. Aquí se observa que solamente los modelos que reciben la entrada cinemática (velocidad a la que se desplaza el vehículo) son capaces de completar los experimentos de forma exitosa. En el cuarto experimento se evalúa el comportamiento del sistema al tomar el control de una situación de velocidad alta. Esto quiere decir que el vehículo se está conduciendo a 50km/h o 70km/h y se activa en ese momento el autopiloto con los diferentes modelos entrenados. De este modo se comprueba si los modelos son capaces de retomar el control satisfactoriamente en situaciones nunca vistas en entrenamiento. En este caso solo probamos los modelos que reciben la entrada cinemática, ya que son los que han generado buenos resultados en el experimento anterior. Teniendo en cuenta eso, los modelos que son capaces de retomar el control a 50km/h son aquellos que también cuentan con memoria visual, dejando fuera a PilotNet* con entrada cinemática, que ya no es capaz de tomar el control. Si se sube la velocidad a 70km/h solamente es capaz de completar los experimentos con éxito el modelo PilotNetx3*(Conv3D) con entrada cinemática, dejando constancia ya en este momento de que en determinados casos contar con la entrada cinemática y un modelo con memoria visual aporta ventajas en el control.

En el quinto experimento, se comprueba la robustez de los modelos a la manipulación sensorial, en este caso añadiendo ruido Gaussiano a las imágenes de entrada en un 50% o 90% de los píxeles De nuevo se extrae de este experimento que a la hora de aumentar mucho la manipulación sensorial, los únicos modelos que aún muestran resultados satisfactorios son aquellos que cuentan con entrada cinemática y memoria visual. En el caso de este experimento, el único modelo que es capaz de completar los experimentos satisfactoriamente con el 90% es PilotNetx3*(TimeDistributed). En el último experimento se estudia la densidad y la longitud de la memoria visual en cuanto al número de imágenes que recibe y la longitud temporal entre cada una de ellas. Con este experimento se busca estudiar cómo afecta la densidad de la cantidad de datos recibida y el espacio temporal entre cada uno de esos datos. La mejor combinación de número de imágenes son tres y la distancia de pasos de tiempo en la que marca el límite para que funcione el sistema es (t, $t-5$, y $t-10$).

En esta contribución se han presentado una serie de experimentos que han validado la hipótesis inicial de que añadir memoria visual y entrada cinemática mejora la calidad final del control del vehículo autónomo. Para conocer todos los detalles sobre esta contribución, se puede revisar el Capítulo 6.

10.4.5. Optimización del control extremo a extremo en conducción autónoma

Esta subsección muestra otra de las contribuciones de esta tesis doctoral, donde se exploran y comparan una variedad de alternativas para la optimización de modelos que resuelven la aplicación de seguir el carril de forma visual en escenarios urbanos con una aproximación de aprendizaje por imitación. Esta contribución es una publicación en una revista científica y en una conferencia [244] y su material, incluyendo arquitecturas, pesos de los modelos, conjunto de datos y herramienta de comparación, es de código abierto [249]. Las técnicas de optimización incluyen la cuantización, recorte, ajuste fino (reentrenamiento) y agrupación, cubriendo todas las opciones disponibles en los entornos más comunes de aprendizaje profundo. También se exploran optimizaciones dadas por TensorRT, que son específicas al hardware específico de Nvidia utilizado. De nuevo se utiliza Behavior Metrics para los experimentos en línea con el simulador CARLA.

En la conducción autónoma, las soluciones basadas en visión suelen ser generadas utilizando modelos de aprendizaje profundo, las cuales son soluciones computacionalmente muy exigentes. Un componente muy importante dentro de estos sistemas es el hardware disponible, ya que de él depende el rendimiento de la aplicación robótica. Además de la calidad que puedan tener sus decisiones de control, el sistema debe tener una alta frecuencia de respuesta para ser utilizable. Algunos vehículos autónomos cuentan con este hardware de altas prestaciones, pero no siempre es así y actualizarlo no es siempre una opción. Por eso, se puede optar por investigar las técnicas de optimización de los modelos propios de aprendizaje profundo utilizados para incorporarlas dentro del sistema final.

En esta contribución exploramos las diferentes técnicas de optimización disponibles en los entornos de aprendizaje profundo (TensorFlow y PyTorch) y en TensorRT y comparamos su rendimiento en un sistema extremo a extremo de control para un vehículo autónomo entrenado por aprendizaje por imitación.

Para la base de exploración, se toma el modelo neuronal de PilotNet*, ya introducido en la Subsección 10.4.4, que tiene una entrada visual de vista de pájaro y genera comandos de control que son la aceleración, el giro y el freno del vehículo. Este modelo se entrena con el dataset previamente presentado siguiendo el mismo modo de trabajo. Las optimizaciones que se aplican son aquellas disponibles en los diferentes entornos de trabajo y son en concreto: cuantización (aproximación de los valores internos de la red neuronal a ancho de bit más bajo), recorte (eliminar partes y conexiones innecesarias de la red), ajuste fino (algunas técnicas incluyen reentrenamiento para generar el modelo optimizado) y agrupamiento (agrupar zonas de la red con pesos similares que puedan combinarse). Además de estas optimizaciones, se aplican también las variantes y com-

binaciones de todas ellas. Esto da lugar a 10 modelos optimizados diferentes utilizando TensorFlow, seis utilizando PyTorch y seis utilizando TensorRT.

En la parte experimental se generan 3 experimentos. En el primero se hace una evaluación fuera de línea de cada una de las técnicas disponibles, variaciones y combinaciones, evaluando el tamaño de los modelos, su MSE en un conjunto de evaluación de los datos y su frecuencia de inferencia utilizando una GPU. Lo que se extrae de este experimento es que las optimizaciones generan modelos más eficientes y que la mejor forma de sacarle partido a estas ideas es con la combinación de las diferentes técnicas en un mismo modelo. Esa combinación genera los mejores resultados en cuanto a tamaño del modelo, MSE y frecuencia de inferencia en GPU. Estos resultados son similares para TensorFlow y PyTorch y también para TensorRT, aunque los mejores resultados se observan en esta última aproximación. La explicación para este hecho se puede achacar al hecho de que las optimizaciones proporcionadas por TensorRT son específicas para el *hardware* utilizado de Nvidia en comparación con las optimizaciones proporcionadas por los *frameworks* que son más genéricas independientemente del lugar de despliegue del modelo.

En el siguiente experimento, se prueban en línea los modelos, en un circuito de test de CARLA y obteniendo las métricas con Behavior Metrics. Lo que se observa, de nuevo, es que aquellos modelos que combinan las diferentes optimizaciones generan los mejores modelos. Se observa una ganancia de 47 veces en cuanto a la frecuencia de inferencia y de 5 veces en la frecuencia de control en comparación con la base sin optimizar.

Para el último experimento, se espera que un vehículo siga el carril a partir de los datos de entrada visuales y se realiza un análisis de su comportamiento en cuanto a frecuencia de inferencia y de decisiones de control. En este experimento se observa que para que el vehículo conduzca de forma satisfactoria, su controlador tiene que generar al menos 10 decisiones por segundo (10Hz), generando un límite inferior. Subiendo este número de inferencias, el control se mantiene igual en cuanto a calidad de las decisiones. A la hora de utilizar un modelo optimizado, es mucho más sencillo superar ese número de 10Hz e incluso se observa que el modelo que mejor se comporta entre los desarrollados previamente genera 1014Hz. Esto quiere decir que incluso con un hardware mucho más limitado, en el que no se alcanzasen esos 1014Hz, este modelo aún podría generar resultados satisfactorios, mientras que otros lo tendrían más complicado o sería imposible para ellos.

Con esta validación experimental, hemos probado la importancia de incluir técnicas de optimización en el desarrollo de sistemas de conducción autónoma y hemos hecho un estudio de qué técnicas concretas generan los mejores resultados. Para conocer todos los detalles sobre esta contribución, se puede revisar el Capítulo 7.

10.4.6. Conducción autónoma extremo a extremo en tráfico

Esta subsección presenta otra contribución de esta tesis doctoral, que está bajo revisión por pares en una revista científica en el momento de escribir este documento [251]. En esta contribución se busca generar un modelo de seguimiento del carril que sea más o menos sencillo, que siga el carril de forma exitosa y que mantenga una distancia segura respecto a otros vehículos que se pueda encontrar a lo largo del experimento. En esta aproximación se vuelve a utilizar una aproximación de aprendizaje por imitación, creando una serie de conjuntos de datos a partir del comportamiento de un agente experto conduciendo por el escenario. En este caso se demuestra que los modelos, pese a su simplicidad e incluir solo unos pocos cambios en su arquitectura, pueden ampliar su área de aplicación en un margen amplio con respecto al punto de partida para que el que fueron creados. En este caso se utiliza de nuevo una variante de PilotNet, en cuya forma base solamente controlaba en giro del volante y en esta contribución, incluyendo unos pocos cambios, es capaz de controlar la aceleración, giro y freno, además de mantener una distancia de seguridad con respecto al vehículo delantero que cumple con los límites de seguridad. Además de mantener esta distancia de seguridad, es capaz de detenerse si el vehículo delantero lo hace y reanudar la marcha una vez el vehículo delantero vuelve a moverse. Para la evaluación experimental, se utilizan de nuevo el simulador de conducción autónoma CARLA y el software de evaluación objetiva y masiva Behavior Metrics, esta vez con otra nueva métrica específica que se incluye en la herramienta para medir la distancia con otros vehículos.

Para el desarrollo de esta contribución, se generan tres versiones del dataset supervisado que son complementarios que se obtienen grabando los datos sensoriales y las salidas de control generadas por el piloto experto mientras conduce por un entorno de entrenamiento. En el primero, no se incluyen otros vehículos. En el segundo, se complementa con grabaciones del piloto experto conduciendo con un único tipo de vehículo delantero como tráfico. En la tercera versión, se complementa lo anterior con una mayor variedad de vehículos delanteros, en este caso seis, diferentes entre sí. En este caso, se utiliza una cámara puesta en la parte frontal del vehículo como entrada de datos.

En cuanto a los modelos utilizados, se prueban dos variantes del modelo PilotNet (PilotNet* y PilotNet**). En la primera de ellas (PilotNet*) se le añade una nueva salida que genera tanto la aceleración como el freno, ya que son complementarias, y se incluyen capas de dropout. En la segunda variante (PilotNet**), además de lo anterior, se incluye la entrada cinemática del modelo como un nuevo canal, siendo todos sus valores la velocidad normalizada del modelo. Para el entrenamiento de los modelos, de nuevo se utilizan técnicas de aumentado de datos y sobrerrepresentación de los datos que tienen pocas apariciones.

Para la validación experimental, se utilizan tres experimentos. En el primero se realiza un experimento de una ejecución típica sin tráfico para las dos arquitecturas presentadas. En este experimento se utilizan los diferentes modelos entrenados con las variantes 2 y 3 del dataset. Aquí se observa que los diferentes modelos son capaces de completar

los experimentos de forma satisfactoria. El que mejor resultados arroja es la variante PilotNet** con el dataset que incluye varios vehículos, la versión más avanzada. En concreto, se observa una mejora una mejora en la desviación con respecto al centro del carril a la hora de conducir y en el número de invasiones de carril.

En el segundo experimento se prueba el rendimiento de los modelos en situaciones con tráfico. En este experimento, se ve que el modelo PilotNet** entrenado con el conjunto de datos con varios vehículos genera de nuevo los mejores resultados en un escenario experimental, manteniendo una distancia de seguridad adecuada con otros vehículos. Concretamente, genera un 81% de éxito en los experimentos en el circuito de prueba en comparación con el 16% del segundo modelo mejor. Además supera sustancialmente en las métricas de desviación con respecto a la posición ideal en el centro del carril y en el número de invasiones de carril. Además el modelo PilotNet** más avanzado es capaz de mantener una distancia de seguridad adecuada con respecto al resto de vehículos en un 86% de las ocasiones.

Para el último experimento, se realiza la prueba con diferentes tipos de vehículos delanteros, para comprobar las capacidades de generalización del modelo. En este caso, de nuevo se ve que el modelo PilotNet** con el dataset con varios vehículos es capaz de generalizar a muchos más vehículos, incluso a algunos que nunca ha visto. En los resultados se observa una subida en cuanto al éxito en los experimentos hasta en el 100% de ocasiones en las que se encuentra con la mayoría de vehículos evaluados, incluso con vehículos que morfológicamente distan mucho de los utilizados para entrenar.

En conclusión, se presenta una propuesta para la conducción autónoma segura en escenarios con tráfico a partir de una aproximación entrenada por aprendizaje por imitación de extremo a extremo. Se prueba que con modelos que en principio podríamos considerar no muy profundos, aplicando pequeños cambios, el rango a aplicación de los mismos es mucho mayor, sin necesidad de tener que utilizar modelos mucho más complejos. Este modelo solamente utiliza una cámara como entrada sensorial y con un modelo aparentemente simple es capaz de regularse en situaciones de tráfico. Para conocer todos los detalles sobre esta contribución, se puede revisar el Capítulo 8.

10.5. Conclusiones

En esta tesis doctoral se han propuesto una serie de objetivos de investigación en el campo de la conducción autónoma y la monitorización del tráfico rodado. Se han abordado y en consecuencia se han generado una serie de contribuciones. Se ha comenzado con una *revisión integral de la literatura* en los campos que se ven involucrados en cuanto a la conducción autónoma y la monitorización del tráfico (visión por computador, robótica e inteligencia artificial). En particular, se ha revisado la detección de objetos basada en aprendizaje profundo. Después se ha revisado el campo de la conducción autónoma y en especial las soluciones extremo a extremo. Esto ha incluido una revisión de la posibilidad

de añadir memoria a los sistemas de conducción u otros tipos de entrada sensorial. También se ha revisado el aprendizaje por imitación, la evaluación de soluciones de conducción autónoma, los simuladores y las técnicas de optimización en aprendizaje profundo para modelos de conducción autónoma.

Después de esta revisión, se ha presentado la primera contribución, TrafficSensor. Este sistema es una *solución para monitorización del tráfico rodado utilizando aprendizaje profundo*. Con los conocimientos extraídos de esta contribución, se ha visto la robustez que las soluciones basadas en aprendizaje profundo aportan a la percepción de los sistemas, tanto en monitorización del tráfico como en conducción autónoma. Además, se ha comprendido la importancia de la generación de evaluaciones eficientes para comparar soluciones.

Abordando estas cuestiones, se ha desarrollado Detection Metrics, una contribución original de la tesis. Este software multiplataforma de código abierto está orientado hacia la *evaluación automática, cuantitativa y masiva de modelos de detección de objetos visual*. Esta contribución se ha utilizado para la primera, TrafficSensor, en su parte experimental, probando su validez. Además se ha validado con otro experimento.

Siguiendo con las contribuciones y gracias al aprendizaje de las previas, se ha presentado Behavior Metrics, *una herramienta software de código abierto para la evaluación en línea de sistemas de conducción autónoma*. De nuevo se ha tomado como punto de partida la importancia de la generación de métricas cuantitativas y cualitativas relevantes para los investigadores para poder avanzar en su investigación. Esta herramienta soporta diferentes tareas de conducción como el seguimiento de carril, la conducción en tráfico y la navegación entre puntos, aportando métricas específicas para cada una de ellas que complementan a las dadas por los simuladores. Esta herramienta se ha utilizado en el resto de contribuciones, probando su validez de forma experimental.

Se ha continuado la exploración en el campo de la conducción autónoma a partir de esto con un nuevo estudio que es otra de las contribuciones. Más allá de *conseguir el modelo básico visual que es capaz de seguir el carril de forma satisfactoria con aprendizaje por imitación*, este estudio ha explorado *las implicaciones de añadir memoria visual y entrada de datos cinemática* a unas arquitecturas de aprendizaje profundo aparentemente sencillas y se ha explorado cómo estos cambios afectan al comportamiento del robot. Se ha demostrado que añadir ambos cambios hace que el comportamiento mejore en determinadas situaciones.

Continuando con la investigación en los modelos de conducción autónoma de extremo a extremo, se ha presentado un estudio de las posibilidades que la *optimización de las redes neuronales añade al rendimiento de estos modelos y cómo afecta esto en el comportamiento final del mismo*. Estos modelos han demostrado experimentalmente que añadir optimizaciones mejora el sistema final gracias a hacer el controlador más rápido y pequeño sin perder calidad en la decisión de control. Dentro de este estudio, se han explorado tanto optimizaciones en los diversos *frameworks* de desarollo de modelos de

aprendizaje profundo y optimizaciones a nivel de hardware con TensoRT.

Como última contribución, se ha presentado una propuesta para *conducción segura en tráfico* siguiendo una aproximación de extremo a extremo basada en aprendizaje por imitación y aprendizaje profundo. El modelo generado, a pesar de su aparente simplicidad y modificaciones de poco calado respecto al modelo base del que surge, consigue ajustar su velocidad con respecto a otros vehículos, parando cuando los otros lo hacen y continuando su marcha en el momento adecuado. Además de esto, este modelo es capaz de generalizar a una gran variedad de diferentes tipos de vehículos delante del propio.

10.5.1. Contribuciones de investigación

Algunos de los resultados presentados en esta tesis han sido publicados y compartidos con la comunidad científica a través de artículos colaborativos con otros investigadores:

Artículos en revistas y conferencias:

- S. Paniego, R. Calvo-Palomino, and J. Cañas, "Behavior Metrics: An Open-Source Assessment Tool for Autonomous Driving Tasks," *Software X*, vol. 26, pp. 101702, 2024 doi: 10.1016/j.softx.2024.101702. [Online].
  Disponible: https://doi.org/10.1016/j.softx.2024.101702 [239]

- S. Paniego, N. Paliwal, and J. Cañas, "Model optimization in deep learning based robot control for autonomous driving," *IEEE Robotics and Automation Letters and IEEE International Conference on Robotics and Automation (ICRA)*, vol. 9, no. 1, pp. 715–722, 2024. doi: 10.1109/LRA.2023.3336244. [Online]. Disponible: https://doi.org/10.1109/LRA.2023.3336244 [244]

- S. Paniego, V. Sharma, and J. M. Cañas, "Open source assessment of deep learning visual object detection," *Sensors*, vol. 22, no. 12, 2022. doi: 10.3390/s22124575. [Online]. Available: https://www.mdpi.com/1424-8220/22/12/4575 [234]

- J. Fernández, J. M. Cañas, V. Fernández, and S. Paniego, "Robust real-time traffic surveillance with deep learning," Computational Intelligence and Neuroscience, vol. 2021, p. 4 632 353, Dec. 2021. doi: 10.1155/2021/4632353. [Online]. Disponible: https://doi.org/10.1155/2021/4632353 [224]

Manuscritos bajo revisión por pares (revisado en marzo de 2024)

- Enhancing End-to-End Control in Autonomous Driving through Kinematic-Infused and Visual Memory Imitation Learning. Sergio Paniego, Roberto Calvo-Palomino, and José María Cañas

- Autonomous Driving in Traffic with End-to-End Vision-based Deep Learning. Sergio Paniego, Enrique Sinohara, and José María Cañas

Preprints:

- S. P. Blanco, S. Mahna, U. A. Mishra, and J. Canas, Memory based neural networks for end-to-end autonomous driving, 2022. arXiv: 2205.12124 [cs.RO]. [256]

También he formado parte de otros proyectos de investigación durante el desarrollo de esta tesis doctoral, aunque no contribuyen a esta:

Artículos en workshops

- P. F. de Cabo, R. Lucas, I. Arranz, S. Paniego, and J. M. Cañas, "RL-studio: A tool for reinforcement learning methods in robotics," in ROBOT2022: Fifth Iberian Robotics Conference, Springer International Publishing, Nov. 2022, pp. 502–513.doi: 10.1007/978-3-031-21062-4_41. [Online]. Available: https://doi.org/10.1007%2F978-3-031-21062-4_41. [257]

10.5.2. Trabajo futuro

Las contribuciones presentadas en esta tesis doctoral abren nuevos caminos de exploración basados en la misma. En concreto, algunos de los que abre y en los que trabajamos de cara al futuro son:

- **Navegación extremo a extremo entre puntos en escenarios utilizando comandos de entrada:** nos gustaría ampliar las capacidades del sistema mientras se mantiene su simplicidad. En este caso, se busca que el sistema comprenda *comandos de entrada* que le indiquen cómo se debe comportar [172]. Esta idea ya se ha explorado como un proyecto de Google Summer of Code [245] y se sigue con su desarrollo.
- **Transfiriendo las soluciones actuales de extremo a extremo a un vehículo real:** estamos actualmente trabajando con el laboratorio de la Universidad Carlos III de Madrid, Autonomous Mobility and Perception Lab (AMPL) en la transferencia de los modelos de extremo a extremo presentados en esta tesis a un vehículo real [258] [259].
- **Conducción extremo a extremo de un vehículo autónomo modulada con instrucciones basadas en texto:** dado el gran avance de los modelos grandes del lenguaje en los últimos años (BERT [261], GPT-3 [262], Llama 2 [263] o Mistral 7B [264]), en este trabajo futuro buscaríamos introducir instrucciones al vehículo autónomo que estén basadas en lenguaje natural [265] [266] [267] [268]. Para ello, se espera abordar como un proyecto de Google Summer of Code 2024 [17].

[17] https://jderobot.github.io/activities/gsoc/2024

- **Exploración de la conducción autónoma en vehículos aéreos:** las soluciones actualmente presentadas están enfocadas en vehículos terrestres. Otra posible línea de investigación futura es trasladar este conocimiento a vehículos aéreos como podrían ser los drones [269] [270]. Se está explorando esta idea actualmente como parte de un trabajo fin de grado [18].

- **Exploración de la conducción autónoma en entornos no estructurados:** los terrenos no estructurados presentan unas particularidades en la percepción y la navegación de los vehículos que son bastante desafiantes. En esta línea futura, exploraremos esta línea. Existen conjuntos de datos enfocados en este tipo de problemas de percepción, como GOOSE [271] o Rellis 3D [272]. Actualmente, se está trabajando en ella con un proyecto de la Agencia Estatal de Investigación de España (*(GAIA) Gestión integral para la prevención, extinción y reforestación debido a incendios forestales, Proyectos de I+D en líneas estratégicas en colaboración entre organismos de investigación y difusión de conocimientos TRANSMISIONES 2023. Ref PLEC2023-010303 (2024-2026)*).

- **Exploración del aprendizaje por refuerzo basado en aproximaciones de conducción autónoma extremo a extremo:** como última línea futura, exploraremos las aproximaciones de aprendizaje por refuerzo y su combinación con el aprendizaje profundo para generar sistemas de conducción autónoma [179] [273]. Esto lo abordaremos sobre la base de lo que ya hemos explorado previamente en aprendizaje por refuerzo [257].

[18] https://www.youtube.com/watch?v=jJ4Xdin1gg4&ab_channel=JdeRobot

Appendix A

Replicability and software, data, and models availability

This appendix is intended to ensure the accessibility and reproducibility of the research presented in this PhD thesis. All the software, deep learning models' weights, and data that we have presented in this document are open source, ensuring their replicability and extensibility for future research. We also keep a website for summarizing the content and providing access to the information. The PhD material website is hosted in https://sergiopaniego.github.io/phd_thesis/.

A.1. Software availability

Detection Metrics, *Behavior Metrics*, and *TrafficSensor* are open source, along with all the data, models' weights, and training code used during this research. They are hosted on *GitHub* in the links provided below:

- **TrafficSensor**: https://github.com/JdeRobot/smart-traffic-sensor
- **Detection Metrics**:https://github.com/JdeRobot/DetectionMetrics
- **Behavior Metrics**:https://github.com/JdeRobot/BehaviorMetrics

A.2. Training code availability

The code for training the visual end-to-end deep learning imitation learning for driving autonomously models presented along the thesis and to modify them is also open source:

- **DeepLearningStudio**:
  https://github.com/JdeRobot/DeepLearningStudio

A.3. Models' weights availability

The models' weights used during the thesis are open source to increase the reproducibility of the presented work. They are hosted in *HuggihgFace* to facilite the access. In the first link, we include the models' weights for the contribution presented in Chapter 6. The second one hosts the models' weights for the optimized deep learning models presented in Chapter 7. In the third link, the models used in Chapter 8 are presented:

- **MemoryPilotNet (Chapter 6)**:
  https://huggingface.co/sergiopaniego/MemoryPilotNet
- **OptimizedPilotNet (Chapter 7)**:
  https://huggingface.co/sergiopaniego/OptimizedPilotNet
- **subjective_vision_pilotnet (Chapter 8)**:
  https://huggingface.co/YujiroS/subjective_vision_pilotnet

A.4. Datasets availability

The datasets used for training the models are also open source to enhance research progress and reproducibility. In the first link, the dataset used during Chapter 6 and Chapter 7 is hosted. The other link hosts the datasets used for Chapter 8:

- **CarlaFollowLanePreviousV (Chapter 6 and Chapter 7)**:
  https://huggingface.co/datasets/sergiopaniego/CarlaFollowLanePreviousV
- **traffic-6 (Chapter 8)**:
  https://huggingface.co/datasets/YujiroS/traffic-6

Bibliography

[1] T. Litman, "Autonomous Vehicle Implementation Predictions Implications for Transport Planning," Victoria Transport Policy Institute, 2022.

[2] N. H. T. S. Administration, "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey," U.S. Department of Transportation, 2015.

[3] W. H. Organization, "Global status report on road safety 2018," World Health Organization, Jun. 2018.

[4] L. D. Lillo, T. Gode, X. Zhou, M. Atzei, R. Chen, and T. Victor, *Comparative Safety Performance of Autonomous- and Human Drivers : A Real-World Case Study of the Waymo One Service*, 2023. arXiv: 2309.01206 [cs.RO].

[5] L. Zhang, "Human Ridehail Crash Rate Benchmark," *Cruise*, 2023, https://www.getcruise.com/news/blog/2023/human-ridehail-crash-rate-benchmark/ [Online; accessed 31-Mar-2024].

[6] J. Vargas, S. Alsweiss, O. Toker, R. Razdan, and J. Santos, "An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions," *Sensors*, vol. 21, no. 16, 2021. DOI: 10.3390/s21165397. [Online]. Available: https://www.mdpi.com/1424-8220/21/16/5397.

[7] SAE International, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, https://www.sae.org/standards/content/j3016, [Online; accessed 31-Mar-2024], 2023.

[8] J. Fingas, "Waymo trials fully driverless rides in San Francisco," *Engadget*, 2022, https://www.engadget.com/waymo-fully-driverless-rides-san-francisco-183703989.html [Online; accessed 31-Mar-2024].

[9] Xiaomi, "Xiaomi Unveils Five Core Automotive Technologies and Debuts Xiaomi SU7, Completing the Human x Car x Home Smart Ecosystem," *Xiaomi*, 2024, https://www.mi.com/global/discover/article?id=3095 [Online; accessed 31-Mar-2024].

[10] Tesla, "Autopilot and Full Self-Driving Capability," *Tesla*, 2023, https://www.tesla.com/support/autopilot [Online; accessed 31-Mar-2024].

[11] openpilot contributors, *Openpilot*, https://github.com/commaai/openpilot, [Online; accessed 31-Mar-2024], 2024.

[12] R. Akhtar, "Tesla FSD on its way to Europe as Test Operators hiring spree begins," *The Driven*, 2023, https://thedriven.io/2023/06/07/tesla-fsd-on-its-way-to-europe-as-test-operators-hiring-spree-begins/ [Online; accessed 31-Mar-2024].

[13] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1, Morgan-Kaufmann, 1988.

[14] DARPA, "The Grand Challenge," *DARPA*, 2004, https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles [Online; accessed 31-Mar-2024].

[15] S. Racing, "Stanley website," *Stanford Racing*, 2006, https://cs.stanford.edu/group/roadrunner//old/index.html [Online; accessed 31-Mar-2024].

[16] R. K. Nagpal and E. Cohen, "Automotive electronics revolution requires faster, smarter interfaces," *Embedded*, 2022, https://www.embedded.com/automotive-electronics-revolution-requires-faster-smarter-interfaces/ [Online; accessed 31-Mar-2024].

[17] Waabi, "Introducing the Waabi Driver," *Waabi*, 2023, https://waabi.ai/introducing-the-waabi-driver/ [Online; accessed 31-Mar-2024].

[18] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. [Online]. Available: https://github.com/ros-planning/navigation2.

[19] L. Paull *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1497–1504. DOI: 10.1109/ICRA.2017.7989179.

[20] B. Balaji *et al.*, "DeepRacer: Educational Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning," *arXiv preprint arXiv:1911.01562*, 2019. arXiv: 1911.01562 [cs.RO].

[21] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning," in *Post Proceedings of the NeurIPS 2019 Demonstration and Competition Track*, H. J. Escalante and R. Hadsell, Eds., ser. Proceedings of Machine Learning Research, PMLR, 2020.

[22] T. Thadani, "Cruise recalls all its driverless cars after pedestrian hit and dragged," *The Washington Post*, 2023, https://www.washingtonpost.com/technology/2023/11/08/cruise-crash-driverless-recall/ [Online; accessed 31-Mar-2024].

[23] A. Greenberg, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," *Wired*, 2023, https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ [Online; accessed 31-Mar-2024].

[24] C. M. Tripp Mickle and Y. Lu, "G.M.'s Cruise Moved Fast in the Driverless Race. It Got Ugly.," *The New York Times*, 2023, https://www.nytimes.com/2023/11/03/technology/cruise-general-motors-self-driving-cars.html [Online; accessed 31-Mar-2024].

[25] C. Gómez-Huélamo *et al.*, "How to build and validate a safe and reliable Autonomous Driving stack? A ROS based software modular architecture baseline," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1282–1289. DOI: 10.1109/IV51971.2022.9827271.

[26] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end Autonomous Driving: Challenges and Frontiers," *arXiv preprint arXiv:2306.16927*, 2023. arXiv: 2306.16927 [cs.RO].

[27] Jin-Cyuan Lai and Shih-Shinh Huang and Chien-Cheng Tseng, "Image-Based Vehicle Tracking and Classification on the Highway," *Green Circuits and Systems (ICGCS), 2010 International Conference on*, pp. 666–670, Jun. 2010.

[28] J.M Blosseville, C. Krafft, F. Lenior, V. Motyka, S. Beucher, "New Traffic Measurement By Image Processing," *IFAC Control, Computers, Communications in Transportation*, pp. 35–42, Jan. 1989.

[29] Tomás Rodríguez, Narciso García, "An adaptive, real-time, traffic monitoring system," *Machine Vision and Applications*, pp. 555–567, Jan. 2009.

[30] Li-Chih Chen a, Jun-Wei Hsieh b,n, Yilin Yan b, Duan-Yu Chen, "Vehicle make and model recognition using sparse representation and symmetrical SURFs," *Elsevier*, Jan. 2015.

[31] C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, "An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840–1852, 2021. DOI: 10.1109/TITS.2020.3025687.

[32] Michael Hodlmoser, Branislav Micusik, Ming-Yu Liu, Marc Pollefeys, Martin Kampel, "Classification and Pose Estimation of Vehicles in Videos by 3D Modeling within Discrete-Continuous Optimization," *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference*, pp. 198–205, Oct. 2012.

[33] Niluthpol Chowdhury Mithun, Nafi Ur Rashid, and S. M. Mahbubur Rahman, "Detection and Classification of Vehicles From Video Using Multiple Time-Spatial Images," *IEEE Transactions on intelligent transportation systems*, vol. 13, 3 Sep. 2012.

[34] Ninad S. Thakoor, Member, IEEE, and Bir Bhanu, "Structural Signatures for Passenger Vehicle Classification in Video," *IEEE Transactions on intelligent transportation systems*, vol. 14, 4 Dec. 2013.

[35] C. N. T. Tao Wang Zhigang Zhu, "A multimodal temporal panorama approach for moving vehicle detection, reconstruction and classification," vol. Computer Vision and Image Understanding, pp. 1724–1735, 117 Dec. 2013.

[36] J. G. a. Q. Wei Wang Yulong Shang, "Real-time Vehicle Classification Based on Eigenface," vol. Consumer Electronics, Communications and Networks (CECNet), pp. 4292–4295, Apr. 2011.

[37] Y. Yang, Y. Ming, Y. Gang, and Z. Yandong, "Length-Based Vehicle Classification in Multi-lane Traffic Flow," vol. Transactions of Tianjin University, Vol 17, pp. 362–368, 5 Oct. 2011.

[38] N. Buch, S.A. Velastin and J. Orwell, "A Review of Computer Vision Techniques for the Analysis of Urban Traffic," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 920–239, 3 Mar. 2011.

[39] Neeraj K. Kanhere; Stanley T. Birchfield, "A Taxonomy and Analysis of Camera Calibration Methods for Traffic Monitoring Applications," *Intelligent Transportation Systems, IEEE Transactions*, vol. 11, pp. 441–452, 2 Jun. 2010.

[40] Y.-S. C. Shih-Hao Yu Jun-Wei Hsieh and W.-F. Hu, "An Automatic Traffic Surveillance System for Vehicle Tracking and Classification," vol. Lecture Notes in Computer Science Volume 2749, pp. 379–386, Jun. 2003.

[41] C. Pang, W. Lam, and N. Yung, "A novel method for resolving vehicle occlusion in a monocular traffic-image sequence," *Intelligent Transportation Systems*, vol. 5, pp. 129–141, 3 Sep. 2004.

[42] C. Pang, W. Lam, and N. Yung, "A Method for Vehicle Count in the Presence of Multiple-Vehicle Occlusions in Traffic Images," *Intelligent Transportation Systems*, vol. 8, pp. 441–459, 3 Sep. 2007.

[43] Koller, D., Weber, J., Malik, J., "Robust multiple car tracking with occlusion reasoning," *European Conference on Computer Vision*, vol. 1, pp. 189–196, May 1994.

[44] A. J. Kanwal Yousaf Arta Iftikhar, "Comparative Analysis of Automatic Vehicle Classification Techniques: A Survey," *International Journal of Image, Graphics and Signal Processing*, vol. 4, no. 9, pp. 52–59, Sep. 2007.

[45] Y. W. Yang Lv Benjamin Yao and S.-C. Zhu, "Reconfigurable Templates for Robust Vehicle Detection and Classification," vol. Applications of Computer Vision (WACV), pp. 321–328, Jan. 2012.

[46] Zezhi Chen, Tim Ellis, Sergio A Velastin, "Vehicle Type Categorization: A comparison of classification schemes," *2011 14th International IEEE Conference on Intelligent Transportation Systems*, pp. 74–79, Oct. 2011.

[47] haoxiang Zhang, Tieniu Tan, Kaiqi Huang, and Yunhong Wan, "Three-Dimensional Deformable-Model-Based Localization and Recognition of Road Vehicles," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 21, pp. 1–13, 1 Jan. 2012.

[48] Zezhi Chen, Tim Ellis, "Multi-shape Descriptor Vehicle Classification for Urban Traffic," *International Conference on Digital Image Computing: Techniques and Applications*, pp. 465–461, Dec. 2011.

[49] Andrea Vedaldi, Varun Gulshan, Manik Varma and Andrew Zisserman, "Multiple Kernels for Object Detection," *IEEE 12th International Conference on Computer Vision (ICCV)*, 2009.

[50] B. Scholkopf and A. Smola, "Learning with Kernels," *MIT Press*, 2002.

[51] K. Kim, T. Chalidabhongse, D. Harwood, and L. Davis, "Background modeling and subtraction by codebook construction," *Proc. ICIP*, vol. 5, pp. 3061–3064, Oct. 2004.

[52] K. Kim, T. H. Chalidabhongse, D. Harwood, L. Davis, "Real-time foreground–background segmentation using codebook model," *Elseiver, Real time imaging*, pp. 172–185, Mar. 2005.

[53] Luis Unzueta– Marcos Nieto– Andoni Cortés– Javier Barandiaran– Oihana Otaegui– and Pedro Sánchez, "Adaptive Multicue Background Subtraction for Robust Vehicle Counting and Classification," vol. 13, pp. 527–540, 2 Jun. 2012.

[54] T. Chalidabhongse, K. Kim, D. Harwood, and L. Davis, "A perturbation method for evaluating background subtraction algorithms," *In Proc. IEEE Joint Int. Workshop VS-PETS*, pp. 1–7, Jan. 2003.

[55] Z. Zivkovic and F. van der Heijden, "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction," *Pattern Recognition Letters*, vol. 27, pp. 773–780, 7 2006.

[56] Lili Huang, Student Member, IEEE, "Real-Time Multi-Vehicle Detection and Sub-Feature Based Tracking for Traffic Surveillance Systems," *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics.*, 2010.

[57] K. Robert, "Night-Time Traffic Surveillance: A Robust Framework for Multi-Vehicle Detection, Classification and Tracking," vol. Advanced Video and Signal Based Surveillance, pp. 1–6, Sep. 2009.

[58] Zoran Zivkovic, "Improved Adaptive Gaussian Mixture Model for Background Subtraction," *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2, pp. 28–31, Aug. 2004.

[59] Pulli Harsha Samhitha, Allu Naga Jyothi, Ramana Vesapogu, Manasa Mannem and S.Sri Harsha, "Vehicle Detection, Tracking and Speed Measurement for Traffic Regulation," *IRJET.*, vol. 04, 04 Apr. 2017.

[60] Kenan Mu, Fei Hui, and Xiangmo Zhao, "Multiple Vehicle Detection and Tracking in Highway Traffic Surveillance Video Based on SIFT Feature Matching," *Journal of Information Processing Systems.*, 2016.

[61] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[62] Marcos Nieto, Luis Unzueta, Javier Barandiaran, Andoni Cortés, Oihana Otaegui and Pedro Sánchez, "Vehicle tracking and classification in challenging scenarios via slice sampling," *EURASIP Journal on Advances in Signal Processing*, Oct. 2011.

[63] Michael Hodlmoser, Branislav Micusik, Marc Pollefeys, Ming-Yu Liu, Martin Kampel, "Model-Based Vehicle Pose Estimation and Tracking in Videos Using Random Forests," *International Conference on 3D Vision*, pp. 430–437, Jul. 2013.

[64] Zezhi Chen, Tim Ellis, Sergio A Velastin, "Vehicle Detection, Tracking and Classification in Urban Traffic," *15th International IEEE Conference on Intelligent Transportation Systems*, Sep. 2012.

[65] Zhou Zhu and Xiaobo Lu, "An Accurate Shadow Removal Method For Vehicle Tracking," *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 2, pp. 59–62, Oct. 2010.

[66] Liang Wang, Fangliang Chen, Huiming Yin, "Detecting and tracking vehicles in traffic by unmanned aerial vehicles," *Department of Civil Engineering and Engineering Mechanics, Columbia University.*, 2016.

[67] M. Lili Huang Barth, "Real-time multi-vehicle tracking based on feature detection and color probability model," *Intelligent Vehicles Symposium (IV)*, pp. 981–986, Jun. 2010.

[68] Bjorn Johansson, Johan Wiklund, Per-Erik Forssén, Gösta Granlund, "Combining shadow detection and simulation for estimation of vehicle size and position," *Pattern Recognition Letters*, Jan. 2009.

[69] Matthew J. Leotta and Joseph L. Mundy, "Vehicle Surveillance with a Generic, Adaptive, 3D Vehicle Model," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 33, pp. 1457–1469, 7 Jul. 2011.

[70] Baker, K.D Sullivan, G.D., "Performance Assessment of Model-based nacking," *Applications of Computer Vision, Proceedings, 1992., IEEE Workshop on*, pp. 28–35, Dec. 1992.

[71] Guerrero-Gomez-Olmedo, R. and Lopez-Sastre, R. J. and Maldonado-Bascon, S. and Fernandez-Caballero, A., "Vehicle Tracking by Simultaneous Detection and Viewpoint Estimation," *IWINAC 2013, Part II, LNCS 7931*, pp. 306–316, Jan. 2013.

[72] G.Welch and G.Bishop, "An introduction to the kalman filter," *Technical Report TR 95-041. University of North Carolina at Chapel Hill*, 2006.

[73] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[74] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[75] C. Papageorgiou and T. Poggio, "A Trainable System for Object Detection," *Int. J. Comput. Vision*, vol. 38, no. 1, pp. 15–33, Jun. 2000.

[76] C. P. Papageorgiou, M. Oren, and T. Poggio, "A General Framework for Object Detection," in *Proceedings of the Sixth International Conference on Computer Vision*, ser. ICCV '98, Jan. 1998, pp. 555–.

[77] C.Migel Bautista, C.Austin Dy, M.Iñigo Mañalac, R.Angelo Orbe and M.Cordel, "Convolutional neural network for vehicle detection in low resolution traffic videos," *Center for Automation Research College of Computer Studies, De La Salle University*, 2016.

[78] Jason Kurniawan, Sensa G.S. Syahra , Chandra K. Dewa and Afiahayati, "Traffic Congestion Detection: Learning from CCTV Monitoring Images using Convolutional Neural Network," *INNS Conference on Big Data and Deep Learning*, pp. 291–297, 2018.

[79] Wankou Yang, Ziyu Li , Chao Wang and Jun Li, "A multi-task Faster R-CNN method for 3D vehicle detection based on a single image," *Applied Soft Computing Journal*, 2020.

[80] Ji-qing Luo, Hu-sheng Fang, Fa-ming Shao, Yue Zhong and Xia Hua, "Multi-scale traffic vehicle detection based on faster ReCNN with NAS optimization and feature enrichment," *Defence Technology*, 2020.

[81] Jean-Francois Rajotte , Martin Sotir, Cedric Noiseux , Louis-Philippe Noel and Thomas Bertiere, "Object Counting on Low Quality Images: A Case Study of Near Real-Time Traffic Monitorin," *2018 17th IEEE International Conference on Machine Learning and Applications*, 2018.

[82] Chiman Kwan, David Gribben, Bryan Chou, Bence Budavari, Jude Larkin,Akshay Rangamani, Trac Tran, Jack Zhang and Ralph Etienne-Cummings, "Real-Time and Deep Learning Based Vehicle Detection and Classification Using Pixel-Wise Code Exposure Measurements," *Electronics*, 2020.

[83] Pooja Mahto, Priyamm Garg, Pranav Seth and J Panda, "Refining YOLOV4 for Vehicle Detection," *International Journal of Advanced Research in Engineering and Technology (IJARET)*, vol. 11, 2020.

[84] Luyang Zhang, HaitaoWang, XinyaoWang, Shuai Chen, Huaibin Wang and Kai Zheng, "Vehicle Object Detection Based on Improved RetinaNet," *Journal of Physics: Conference Series*, 2021.

[85] Christian Szegedy, Alexander Toshev and Dumitru Erhan, "Deep Neural Networks for Object Detection," *NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systems*, vol. 2, pp. 2553–2561, 2013.

[86] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton, "Imagenet classification with deep convolutional neural networks," *In Advances in Neural Information Processing Systems 25*, 2012.

[87] L. Liu *et al.*, "Deep learning for generic object detection: A survey," *arXiv preprint arXiv:1809.02165*, 2018. arXiv: 1809.02165 [cs.RO].

[88] S. Agarwal, J. O. du Terrail, and F. Jurie, "Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks," *arXiv preprint arXiv:1809.03193*, vol. abs/1809.03193, 2018. arXiv: 1809.03193 [cs.RO]. [Online]. Available: http://arxiv.org/abs/1809.03193.

[89] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *arXiv preprint arXiv:1807.05511*, 2018. arXiv: 1807.05511 [cs.RO].

[90] L. Jiao *et al.*, "A Survey of Deep Learning-Based Object Detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019. DOI: 10.1109/ACCESS.2019.2939201.

[91] Zhengxia Zou and Z. Shi and Yuhong Guo and Jieping Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, vol. abs/1905.05055, 2019. arXiv: 1905.05055 [cs.RO].

[92] X. Wu, D. Sahoo, and S. C. H. Hoi, "Recent Advances in Deep Learning for Object Detection," *arXiv preprint arXiv:1908.03673*, vol. abs/1908.03673, 2019. arXiv: 1908.03673 [cs.RO]. [Online]. Available: http://arxiv.org/abs/1908.03673.

[93] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, *A Survey of Modern Deep Learning based Object Detection Models*, 2021. DOI: 10.48550/ARXIV.2104.11892. arXiv: 2104.11892 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2104.11892.

[94] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103 812, 2023. DOI: https://doi.org/10.1016/j.dsp.2022.103812. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200422004298.

[95] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," Dec. 2015, pp. 2722–2730. DOI: 10.1109/ICCV.2015.312.

[96] Z. Yang and R. Nevatia, "A multi-scale cascade fully convolutional network face detector," Dec. 2016, pp. 633–638. DOI: 10.1109/ICPR.2016.7899705.

[97] Y. Cai, L. Wen, L. Zhang, D. Du, and W. Wang, *Rethinking Object Detection in Retail Stores*, 2020. DOI: 10.48550/ARXIV.2003.08230. arXiv: 2003.08230 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2003.08230.

[98] R. Kachach and J. M. Cañas, "Hybrid three-dimensional and support vector machine approach for automatic vehicle tracking and classification using a single camera," *Journal of Electronic Imaging*, vol. 25, no. 3, p. 033 021, 2016.

[99] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[100] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[101] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[102] S. Song and J. Xiao, "Tracking Revisited Using RGBD Camera: Unified Benchmark and Baselines," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 233–240.

[103] L. Spinello and K. O. Arras, "People detection in RGB-D data," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 3838–3843. DOI: 10.1109/IROS.2011.6095074.

[104] T. D. B. Alexe *et al.*, *The Open Images Dataset V4*, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11263-020-01316-z.

[105] K. Tong, Y. Wu, and F. Zhou, "Recent advances in small object detection based on deep learning: A review," *Image and Vision Computing*, vol. 97, p. 103 910, 2020. DOI: https://doi.org/10.1016/j.imavis.2020.103910. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885620300421.

[106] C. Chen, M.-Y. Liu, O. Tuzel, and J. Xiao, "R-CNN for Small Object Detection," in *Computer Vision – ACCV 2016*, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds., Cham: Springer International Publishing, 2017, pp. 214–230.

[107] S. Shao *et al.*, "CrowdHuman: A Benchmark for Detecting Human in a Crowd," *arXiv preprint arXiv:1805.00123*, 2018. arXiv: 1805.00123 [cs.RO].

[108] Á. Arcos-García, J. A. Álvarez-García, and L. M. Soria-Morillo, "Evaluation of deep neural networks for traffic sign detection systems," *Neurocomputing*, vol. 316, pp. 332–344, 2018. DOI: https://doi.org/10.1016/j.neucom.2018.08.009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121830924X.

[109] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, "Traffic-Sign Detection and Classification in the Wild," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[110] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "LabelMe: a database and web-based tool for image annotation," *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[111] B. Kang, Z. Liu, X. Wang, F. Yu, J. Feng, and T. Darrell, *Few-shot Object Detection via Feature Reweighting*, 2018. DOI: 10.48550/ARXIV.1812.01866. arXiv: 1812.01866 [cs.RO]. [Online]. Available: https://arxiv.org/abs/1812.01866.

[112] P. L. Jeune and A. Mokraoui, *A Unified Framework for Attention-Based Few-Shot Object Detection*, 2022. DOI: 10.48550/ARXIV.2201.02052. arXiv: 2201.02052 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2201.02052.

[113] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of Image Classification Algorithms Based on Convolutional Neural Networks," *Remote Sensing*, vol. 13, no. 22, 2021. DOI: 10.3390/rs13224712. [Online]. Available: https://www.mdpi.com/2072-4292/13/22/4712.

[114] Z. Liu *et al.*, *Swin Transformer V2: Scaling Up Capacity and Resolution*, 2022. arXiv: 2111.09883 [cs.CV].

[115] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, *Image Segmentation Using Deep Learning: A Survey*, 2020. DOI: 10.48550/ARXIV.2001.05566. arXiv: 2001.05566 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2001.05566.

[116] G. Weng, B. Dong, and Y. Lei, "A level set method based on additive bias correction for image segmentation," *Expert Systems with Applications*, vol. 185, p. 115 633, 2021. DOI: https://doi.org/10.1016/j.eswa.2021.115633. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421010277.

[117] P. Ge, Y. Chen, G. Wang, and G. Weng, "A hybrid active contour model based on pre-fitting energy and adaptive functions for fast image segmentation," *Pattern Recognition Letters*, vol. 158, pp. 71–79, 2022. DOI: https://doi.org/10.1016/j.patrec.2022.04.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865522001234.

[118] A. Kirillov *et al.*, *Segment Anything*, 2023. arXiv: 2304.02643 [cs.CV].

[119] A. Shatnawi, G. Al-Bdour, R. Al-Qurran, and M. Al-Ayyoub, "A comparative study of open source deep learning frameworks," in *Information and Communication Systems (ICICS), 2018 9th International Conference on*, IEEE, 2018, pp. 72–77.

[120] M. Abadi *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[121] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[122] J. Redmon, *Darknet: Open Source Neural Networks in C*, https://pjreddie.com/darknet/, Accessed: 2022-04-27, 2013–2016.

[123] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[124] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[125] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv preprint arXiv:1506.01497*, Jun. 2015. arXiv: 1506.01497 [cs.RO].

[126] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," *CoRR*, vol. abs/1512.02325, 2015. arXiv: 1512.02325.

[127] *SSD Mobilenet V2 COCO Config*, https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_mobilenet_v2_coco.config, Accessed: 2022-04-27.

[128] *SSD: Single-Shot Multibox Detector Keras implementation*, https://github.com/pierluigiferrari/ssd_keras, Accessed: 2022-04-27.

[129] 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.

[130] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2017. arXiv: 1612.08242 [cs.RO].

[131] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, vol. abs/1804.02767, 2018. arXiv: 1804.02767 [cs.RO].

[132] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*, 2020. arXiv: 1804.02767 [cs.RO].

[133] G. Jocher *et al.*, *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*, version v6.1, Feb. 2022. DOI: 10.5281/zenodo.6222936. [Online]. Available: https://doi.org/10.5281/zenodo.6222936.

[134] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information*, 2024. arXiv: 2402.13616 [cs.CV].

[135] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-End Object Detection with Transformers*, 2020. DOI: 10.48550/ARXIV.2005.12872. arXiv: 2005.12872 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2005.12872.

[136] Z. Liu *et al.*, *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, 2021. DOI: 10.48550/ARXIV.2103.14030. arXiv: 2103.14030 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2103.14030.

[137] H. Zhang *et al.*, *DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection*, 2022. DOI: 10.48550/ARXIV.2203.03605. arXiv: 2203.03605 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2203.03605.

[138] Z. Zong, G. Song, and Y. Liu, *DETRs with Collaborative Hybrid Assignments Training*, 2022. arXiv: 2211.12860 [cs.CV].

[139] L. Liu *et al.*, "Computing Systems for Autonomous Driving: State-of-the-Art and Challenges," *arXiv preprint arXiv:2009.14349*, 2020. arXiv: 2009.14349 [cs.RO].

[140] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2759–2764. DOI: 10.1109/IROS.2016.7759428.

[141] E. Shinohara, " Autonomous driving in traffic using end-to-end deep learning," M.S. thesis, Universidad Rey Juan Carlos, 2023.

[142] C. Urmson *et al.*, "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," in *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, M. Buehler, K. Iagnemma, and S. Singh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–59. DOI: 10.1007/978-3-642-03991-1_1. [Online]. Available: https://doi.org/10.1007/978-3-642-03991-1_1.

[143] J. Levinson *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168. DOI: 10.1109/IVS.2011.5940562.

[144] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *arXiv preprint arXiv:1906.05113*, 2019. arXiv: 1906.05113 [cs.RO].

[145] Autoware contributors, *Autoware*, https://github.com/autowarefoundation/autoware, [Online; accessed 31-Mar-2024], 2022.

[146] P. Wu, Y. Cao, Y. He, and D. Li, "Vision-Based Robot Path Planning with Deep Learning," in *Computer Vision Systems*, M. Liu, H. Chen, and M. Vincze, Eds., Cham: Springer International Publishing, 2017, pp. 101–111.

[147] M. Bojarski *et al.*, "Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car," *arXiv preprint arXiv:1704.07911*, 2017. arXiv: 1704.07911 [cs.RO].

[148] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[149] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-End Race Driving with Deep Reinforcement Learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia: IEEE Press, 2018, pp. 2070–2075. DOI: 10.1109/ICRA.2018.8460934. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460934.

[150] W. Zeng, S. Wang, R. Liao, Y. Chen, B. Yang, and R. Urtasun, "DSDNet: Deep Structured Self-driving Network," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 156–172. DOI: 10.1007/978-3-030-58589-1_10.

[151] P. Karkus, B. Ivanovic, S. Mannor, and M. Pavone, "DiffStack: A Differentiable and Modular Control Stack for Autonomous Vehicles," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=teEnA3L4aRe.

[152] Y. Hu *et al.*, "Planning-oriented Autonomous Driving," *arXiv preprint arXiv:2212.10156*, 2023. arXiv: 2212.10156 [cs.RO].

[153] S. Casas, A. Sadat, and R. Urtasun, "MP3: A Unified Model to Map, Perceive, Predict and Plan," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14 398–14 407. DOI: 10.1109/CVPR46437.2021.01417.

[154] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the Limitations of Behavior Cloning for Autonomous Driving," *arXiv preprint arXiv:1904.08980*, 2019. arXiv: 1904.08980 [cs.RO].

[155] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end Learning of Driving Models from Large-scale Video Datasets," *arXiv preprint arXiv:1612.01079*, 2016. arXiv: 1612.01079 [cs.RO].

[156] J. Kocić, N. Jovičić, and V. Drndarević, "An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms," *Sensors*, vol. 19, no. 9, 2019. DOI: 10.3390/s19092064.

[157] A. Riboni, N. Ghioldi, A. Candelieri, and M. Borrotti, "Bayesian Optimization and Deep Learning forsteering wheel angle prediction," *arXiv preprint arXiv:2110.13629*, 2021. arXiv: 2110.13629 [cs.RO].

[158] E. Santana and G. Hotz, "Learning a Driving Simulator," *arXiv preprint arXiv:1608.01230*, 2016. arXiv: 1608.01230 [cs.RO].

[159] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016. arXiv: 1604.07316 [cs.RO].

[160] Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989. DOI: 10.1162/neco.1989.1.4.541.

[161] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *2017 American Control Conference (ACC)*, 2017, pp. 4914–4919. DOI: 10.23919/ACC.2017.7963716.

[162] S. Yang, W. Wang, C. Liu, W. Deng, and J. K. Hedrick, "Feature analysis and selection for training an end-to-end autonomous vehicle controller using deep learning approach," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 1033–1038.

[163] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline," *arXiv preprint arXiv:2206.08129*, 2022. DOI: 10.48550/ARXIV.2206.08129. arXiv: 2206.08129 [cs.CV].

[164] H. Shao, L. Wang, R. Chen, S. L. Waslander, H. Li, and Y. Liu, "ReasonNet: End-to-End Driving With Temporal and Global Reasoning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 13 723–13 733.

[165] K. Chitta, A. Prakash, and A. Geiger, *NEAT: Neural Attention Fields for End-to-End Autonomous Driving*, 2021. arXiv: 2109.04456 [cs.CV].

[166] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "TransFuser: Imitation With Transformer-Based Sensor Fusion for Autonomous Driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, 12878–12 895, 2023. DOI: 10.1109/TPAMI.2022.3200245.

[167] D. Chen and P. Krähenbühl, *Learning from All Vehicles*, 2022. arXiv: 2203.11934 [cs.RO].

[168] A. O. Ly and M. Akhloufi, "Learning to Drive by Imitation: An Overview of Deep Behavior Cloning Methods," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 195–209, 2021. DOI: 10.1109/TIV.2020.3002505.

[169] S. Schaal, "Learning from demonstration," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, ser. NIPS'96, Denver, Colorado: MIT Press, 1996, pp. 1040–1046.

[170] F. Torabi, G. Warnell, and P. Stone, "Behavioral Cloning from Observation," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 4950–4957. DOI: 10.24963/ijcai.2018/687. [Online]. Available: https://doi.org/10.24963/ijcai.2018/687.

[171] C. Diehl, J. Adamek, M. Krüger, F. Hoffmann, and T. Bertram, "Differentiable Constrained Imitation Learning for Robot Motion Planning and Control," *arXiv preprint arXiv:2210.11796*, 2023. arXiv: 2210.11796 [cs.RO].

[172] F. Codevilla, M. Müller, A. Dosovitskiy, A. M. López, and V. Koltun, "End-to-End Driving Via Conditional Imitation Learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.

[173] T. Pearce and J. Zhu, "Counter-Strike Deathmatch with Large-Scale Behavioural Cloning," in *2022 IEEE Conference on Games (CoG)*, IEEE, 2022, pp. 104–111.

[174] S. Ross, G. J. Gordon, and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," *arXiv preprint arXiv:1011.0686*, 2011. arXiv: 1011.0686 [cs.RO].

[175] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst," *arXiv preprint arXiv:1812.03079*, 2018. arXiv: 1812.03079 [cs.RO].

[176] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, p. 125, Feb. 2020. DOI: 10.3390/info11020125. [Online]. Available: https://doi.org/10.3390%2Finfo11020125.

[177] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," *Advances in neural information processing systems*, vol. 29, 2016.

[178] D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon, "IQ-Learn: Inverse soft-Q Learning for Imitation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4028–4039, 2021.

[179] A. Amini *et al.*, "Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020. DOI: 10.1109/LRA.2020.2966414.

[180] R. Gutiérrez-Moreno, R. Barea, E. López-Guillén, J. Araluce, and L. M. Bergasa, "Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator," *Sensors*, vol. 22, no. 21, p. 8373, 2022.

[181] Y. Lu *et al.*, *Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios*, 2023. arXiv: 2212.11419 [cs.AI].

[182] P. A. Lopez *et al.*, "Microscopic Traffic Simulation using SUMO," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/.

[183] E. Espié, C. Guionneau, B. Wymann, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, The Open Racing Car Simulator," 2005.

[184] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[185] V. Costa, R. J. Rossetti, and A. Sousa, "Autonomous driving simulator for educational purposes," in *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, 2016, pp. 1–5. DOI: 10.1109/CISTI.2016.7521461.

[186] 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.

[187] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. V. Gool, *End-to-End Urban Driving by Imitating a Reinforcement Learning Coach*, 2021. arXiv: 2108.08265 [cs.CV].

[188] DeepDrive contributors, *Deepdrive*, https://github.com/deepdrive/deepdrive-sim, [Online; accessed 31-Mar-2024], 2022.

[189] H. Fan *et al.*, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018. DOI: 10.48550/ARXIV.1807.08048.

[190] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *arXiv preprint arXiv:1705.05065*, 2017. DOI: 10.48550/ARXIV.1705.05065. arXiv: 1705.05065 [cs.RO].

[191] Udacity's Self-Driving Car Simulator contributors, *Udacity's self-driving car simulator*, https://github.com/udacity/self-driving-car-sim, [Online; accessed 31-Mar-2024], 2022.

[192] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for Data: Ground Truth from Computer Games," *arXiv preprint arXiv:1608.02192*, 2016. DOI: 10.48550/ARXIV.1608.02192. arXiv: 1608.02192 [cs.RO].

[193] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for Benchmarks," *arXiv preprint arXiv:1604.07316*, 2017. DOI: 10.48550/ARXIV.1709.07322. arXiv: 1709.07322 [cs.RO].

[194] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," vol. 3, Jan. 2009.

[195] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022.

[196] H. Caesar *et al.*, "nuScenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[197] F. Yu *et al.*, "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning," *arXiv preprint arXiv:1805.04687*, 2018. arXiv: 1805.04687 [cs.RO].

[198] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[199] Y. Cabon, N. Murray, and M. Humenberger, "Virtual KITTI 2," *arXiv preprint arXiv:2001.10773*, 2020. arXiv: 2001.10773 [cs.RO].

[200] M. Cordts *et al.*, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *arXiv preprint arXiv:1604.01685*, 2016. DOI: 10.48550/ARXIV.1604.01685. arXiv: 1604.01685 [cs.CV].

[201] H. Caesar *et al.*, "NuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles," *arXiv preprint arXiv:2106.11810*, 2022. arXiv: 2106.11810 [cs.RO].

[202] Udacity's Self-Driving Dataset contributors, *Udacity's Self-Driving Dataset*, https://github.com/udacity/self-driving-car, [Online; accessed 31-Mar-2024], 2023.

[203] S. Ettinger *et al.*, "Large Scale Interactive Motion Forecasting for Autonomous Driving : The Waymo Open Motion Dataset," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9690–9699. DOI: 10.1109/ICCV48922.2021.00957.

[204] M.-F. Chang *et al.*, "Argoverse: 3D Tracking and Forecasting With Rich Maps," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8740–8749. DOI: 10.1109/CVPR.2019.00895.

[205] W. G. Najm, J. D. Smith, and M. Yanagisawa, "Pre-crash scenario typology for crash avoidance research," John A. Volpe National Transportation Systems Center (U.S.), Tech. Rep. DOT-VNTSC-NHTSA-06-02; DOT HS 810 767, Apr. 2007. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/6281.

[206] M. N. Sharath and B. Mehran, "A Literature Review of Performance Metrics of Automated Driving Systems for On-Road Vehicles," *Frontiers in Future Transportation*, vol. 2, 2021. DOI: 10.3389/ffutr.2021.759125. [Online]. Available: https://www.frontiersin.org/articles/10.3389/ffutr.2021.759125.

[207] L. Westhofen *et al.*, "Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art," *Archives of Computational Methods in Engineering*, vol. 30, no. 1, pp. 1–35, Jan. 2023. DOI: 10.1007/s11831-022-09788-7. [Online]. Available: https://doi.org/10.1007/s11831-022-09788-7.

[208] D. Paz, P.-j. Lai, N. Chan, Y. Jiang, and H. I. Christensen, *Autonomous Vehicle Benchmarking using Unbiased Metrics*, 2020. arXiv: 2006.02518 [cs.RO].

[209] P. S. Chib and P. Singh, *Recent Advancements in End-to-End Autonomous Driving using Deep Learning: A Survey*, 2023. arXiv: 2307.04370 [cs.RO].

[210] J. de la Peña, L. M. Bergasa, M. Antunes, F. Arango, C. Gómez-Huélamo, and E. López-Guillén, "AD PerDevKit: An Autonomous Driving Perception Development Kit using CARLA simulator and ROS," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, 2022, pp. 4095–4100. DOI: 10.1109/ITSC55140.2022.9922369.

[211] L. Chi and Y. Mu, "Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues," *arXiv preprint arXiv:1708.03798*, 2017. arXiv: 1708.03798 [cs.RO].

[212] R. Zhao, Y. Zhang, Z. Huang, and C. Yin, "End-to-end Spatiotemporal Attention Model for Autonomous Driving," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, 2020, pp. 2649–2653. DOI: 10.1109/ITNEC48623.2020.9085185.

[213] J. del Egio, L. M. Bergasa, E. Romera, C. Gómez Huélamo, J. Araluce, and R. Barea, "Self-driving a Car in Simulation Through a CNN," in *Advances in Physical Agents*, R. Fuentetaja Pizán, Á. García Olaya, M. P. Sesmero Lorente, J. A. Iglesias Martínez, and A. Ledezma Espino, Eds., Cham: Springer International Publishing, 2019, pp. 31–43.

[214] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies," *arXiv preprint arXiv:1710.03804*, 2017. arXiv: 1710.03804 [cs.RO].

[215] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[216] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 802–810.

[217] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023. DOI: 10.1145/3578938. [Online]. Available: https://doi.org/10.1145/3578938.

[218] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541.

[219] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," *arXiv preprint arXiv:2106.08295*, 2021. arXiv: 2106.08295 [cs.RO].

[220] S. Vadera and S. Ameen, "Methods for Pruning Deep Neural Networks," *arXiv preprint arXiv:2011.00241*, 2020. arXiv: 2011.00241 [cs.RO].

[221] Tensorflow Authors, *TensorFlow Model Optimization*, https://www.tensorflow.org/model_optimization, [Online; accessed 31-Mar-2024], 2023.

[222] NVIDIA, *TensorRT NVIDIA*, https://developer.nvidia.com/tensorrt, [Online; accessed 31-Mar-2024], 2023.

[223] Y. Bai, L. Li, Z. Wang, X. Wang, and J. Wang, "Performance Optimization of Autonomous Driving Control under End-to-End Deadlines," *Real-Time Syst.*, vol. 58, no. 4, pp. 509–547, Dec. 2022. DOI: 10.1007/s11241-022-09379-6. [Online]. Available: https://doi.org/10.1007/s11241-022-09379-6.

[224] J. Fernández, J. M. Cañas, V. Fernández, and S. Paniego, "Robust Real-Time Traffic Surveillance with Deep Learning," *Computational Intelligence and Neuroscience*, vol. 2021, p. 4 632 353, Dec. 2021. DOI: 10.1155/2021/4632353. [Online]. Available: https://doi.org/10.1155/2021/4632353.

[225] Smart Traffic Sensor contributors, *Smart Traffic Sensor*, https://github.com/JdeRobot/smart-traffic-sensor, [Online; accessed 31-Mar-2024], 2022.

[226] TrafficMonitor contributors, *TrafficMonitor*, https://github.com/JdeRobot/traffic-monitor, [Online; accessed 31-Mar-2024], 2019.

[227] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*, 2019. arXiv: 1911.11929 [cs.CV].

[228] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[229] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, *Path Aggregation Network ffor Instance Segmentation*, 2018. arXiv: 1803.01534 [cs.CV].

[230] J.-y. Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker," *Intel Corporation, Microprocessor Research Labs*, pp. 2480–2487, Jan. 2000.

[231] R. Kachach, "Monitorización visual automática de tráfico rodado," Ph.D. dissertation, Universidad de Alicante, 2016.

[232] N. Wojke, A. Bewley, and D. Paulus, *Simple Online and Realtime Tracking with a Deep Association Metric*, 2017. arXiv: 1703.07402 [cs.CV].

[233] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, Sep. 2016. DOI: 10.1109/icip.2016.7533003. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[234] S. Paniego, V. Sharma, and J. M. Cañas, "Open Source Assessment of Deep Learning Visual Object Detection," *Sensors*, vol. 22, no. 12, 2022. DOI: 10.3390/s22124575. [Online]. Available: https://www.mdpi.com/1424-8220/22/12/4575.

[235] DetectionMetrics contributors, *DetectionMetrics*, https://github.com/JdeRobot/DetectionMetrics, [Online; accessed 31-Mar-2024], 2023.

[236] *Tensorflow detection model zoo*, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md, Accessed: 2022-04-27.

[237] *Pytorch torchvision models*, https://pytorch.org/vision/stable/models.html, Accessed: 2022-04-27.

[238] Behavior Metrics contributors, *Behavior Metrics*, https://github.com/JdeRobot/BehaviorMetrics, [Online; accessed 31-Mar-2024], 2022.

[239] S. Paniego, R. Calvo-Palomino, and J. Cañas, "Behavior Metrics: An Open-source Assessment Tool for Autonomous Driving Tasks," *SoftwareX*, vol. 26, p. 101 702, 2024. DOI: https://doi.org/10.1016/j.softx.2024.101702. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711024000736.

[240] Open-source paper resources contributors, *Open-source paper resources*, https://roboticslaburjc.github.io/publications/2024/behavior_metrics_an_open_source_assessment_tool_for_autonomous_driving_tasks, [Online; accessed 31-Mar-2024], 2024.

[241] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[242] Behavior Metrics contributors, *Behavior Metrics GUI mode video example*, https://www.youtube.com/watch?v=ze_LDkmCymk, [Online; accessed 31-Mar-2024], 2024.

[243] Behavior Metrics contributors, *Behavior Metrics headless mode video example*, https://www.youtube.com/watch?v=rcrOF5t3MC4, [Online; accessed 31-Mar-2024], 2024.

[244] S. Paniego, N. Paliwal, and J. Cañas, "Model Optimization in Deep Learning Based Robot Control for Autonomous Driving," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 715–722, 2024. DOI: 10.1109/LRA.2023.3336244.

[245] M. Zhao. "Obstacle Avoidance for Autonomous Driving in CARLA Using Segmentation Deep Learning Models." (2023), [Online]. Available: https://theroboticsclub.github.io/gsoc2023-Meiqi_Zhao/blog/.

[246] Open-source paper resources contributors, *Open-source paper resources*, https://roboticslaburjc.github.io/publications/2023/enhancing_end_to_end_control_in_autonomous_driving_through_kinematic_infused_and_visual_memory_imitation_learning, [Online; accessed 31-Mar-2024], 2023.

[247] S. Paniego, R. Calvo-Palomino, and J. M. Cañas, "Enhancing End-to-End Control in Autonomous Driving through Kinematic-Infused and Visual Memory Imitation Learning," *Manuscript submitted for publication*, 2024.

[248] DeepLearningStudio contributors, *Deeplearningstudio*, https://www.github.com/JdeRobot/DeepLearningStudio, [Online; accessed 31-Mar-2024], 2022.

[249] Open-source paper resources contributors, *Open-source paper resources*, https://roboticslaburjc.github.io/publications/2023/model_optimization_in_deep_learning_based_robot_control_for_autonomous_driving, [Online; accessed 31-Mar-2024], 2023.

[250] Open-source paper resources contributors, *Improved Imitation learning with Bird-eye view for follow-lane autonomous driving in CARLA simulator*, https://www.youtube.com/watch?v=3KflagFjR8Q, [Online; accessed 31-Mar-2024], 2023.

[251] S. Paniego, E. Sinohara, and J. M. Cañas, "Autonomous Driving in Traffic with End-to-End Vision-based Deep Learning," *Manuscript submitted for publication*, 2024.

[252] Open-source paper resources contributors, *Open-source paper resources*, https://roboticslaburjc.github.io/publications/2023/autonomous_driving_in_traffic_with_end_to_end_vision_based_deep_learning, [Online; accessed 31-Mar-2024], 2022.

[253] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[254] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[255] JdeRobot. "Showcasing the social driving capacity with different front vehicles." [Online; accessed 31-Mar-2024], Youtube. (2023), [Online]. Available: https://www.youtube.com/watch?v=mVSfxQeWwrQ.

[256] S. P. Blanco, S. Mahna, U. A. Mishra, and J. Canas, *Memory based neural networks for end-to-end autonomous driving*, 2022. arXiv: 2205.12124 [cs.RO].

[257] P. F. de Cabo, R. Lucas, I. Arranz, S. Paniego, and J. M. Cañas, "RL-Studio: A Tool for Reinforcement Learning Methods in Robotics," in *ROBOT2022: Fifth Iberian Robotics Conference*, Springer International Publishing, Nov. 2022, 502–513. DOI: 10.1007/978-3-031-21062-4_41. [Online]. Available: https://doi.org/10.1007%2F978-3-031-21062-4_41.

[258] M. Á. de Miguel *et al.*, "A Research Platform for Autonomous Vehicles Technologies Research in the Insurance Sector," *Applied Sciences*, vol. 10, no. 16, 2020. DOI: 10.3390/app10165655. [Online]. Available: https://www.mdpi.com/2076-3417/10/16/5655.

[259] Á. Madridano, A. Al-Kaff, D. Martín, and A. de la Escalera, "Trajectory planning for multi-robot systems: Methods and applications," *Expert Systems with Applications*, vol. 173, p. 114 660, 2021. DOI: https://doi.org/10.1016/j.eswa.2021.114660. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421001019.

[260] A. Vaswani *et al.*, *Attention Is All You Need*, 2023. arXiv: 1706.03762 [cs.CL].

[261] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[262] T. B. Brown *et al.*, *Language Models are Few-Shot Learners*, 2020. arXiv: 2005.14165 [cs.CL].

[263] H. Touvron *et al.*, *Llama 2: Open Foundation and Fine-Tuned Chat Models*, 2023. arXiv: 2307.09288 [cs.CL].

[264] A. Q. Jiang *et al.*, *Mistral 7B*, 2023. arXiv: 2310.06825 [cs.CL].

[265] H. Shao, Y. Hu, L. Wang, S. L. Waslander, Y. Liu, and H. Li, *LMDrive: Closed-Loop End-to-End Driving with Large Language Models*, 2023. arXiv: 2312.07488 [cs.CV].

[266] J. Huang, P. Jiang, A. Gautam, and S. Saripalli, *GPT-4V Takes the Wheel: Promises and Challenges for Pedestrian Behavior Prediction*, 2024. arXiv: 2311.14786 [cs.CV].

[267] L. Chen *et al.*, "Driving with LLMs: Fusing Object-Level Vector Modality for Explainable Autonomous Driving," *arXiv preprint arXiv:2310.01957*, 2023. arXiv: 2310.01957 [cs.RO].

[268] A.-M. Marcu *et al.*, "LingoQA: Video Question Answering for Autonomous Driving," *arXiv preprint arXiv:2312.14115*, 2023. arXiv: 2312.14115 [cs.RO].

[269] R. Polvara *et al.*, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 115–123. DOI: 10.1109/ICUAS.2018.8453449.

[270] L. O. Rojas-Perez and J. Martinez-Carranza, "DeepPilot: A CNN for Autonomous Drone Racing," *Sensors*, vol. 20, no. 16, p. 4524, 2020.

[271] P. Mortimer, R. Hagmanns, M. Granero, T. Luettel, J. Petereit, and H.-J. Wuensche, "The GOOSE Dataset for Perception in Unstructured Environments," *arXiv preprint arXiv:2310.16788*, 2023. arXiv: 2310.16788 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2310.16788.

[272] P. Jiang, P. Osteen, M. Wigness, and S. Saripalli, *RELLIS-3D Dataset: Data, Benchmarks and Analysis*, 2020. arXiv: 2011.12954 [cs.CV].

[273] R. Trauth, M. Kaufeld, M. Geisslinger, and J. Betz, "Learning and Adapting Behavior of Autonomous Vehicles through Inverse Reinforcement Learning," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8. DOI: 10.1109/IV55152.2023.10186668.