



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



ROS2 versus AUTOSAR: Automated Parking System case-study

Master Thesis

submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

by

Kenneth Casado Navarro

In partial fulfillment
of the requirements for the master in
ELECTRONIC ENGINEERING



Advisor: Dr.-Ing. Núria Mata and Dr.-Ing. Luis Javier de la Cruz Llopis
Munich, Date 25/11/2022

Revision history and approval record

Revision	Date	Purpose
0	05/09/2022	Document creation
1	04/10/2022	Document revision
2	10/11/2022	Document revision
3	16/11/2022	Document revision
4	22/11/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	
Kenneth Casado Navarro	
Núria Mata	
Luis Javier de la Cruz Llopis	

Written by:		Reviewed and approved by:	
Date	23/11/2022	Date	25/10/2022
Name	Kenneth Casado Navarro	Name	Núria Mata
Position	Project Author	Position	Project Supervisor

Abstract

Vehicles are complex systems as they combine several engineering disciplines, such as mechanical, electric, electronic, software and telecommunication. In the last decades, most innovations in the automotive domain have been achieved as a combination of electronics and software. Consequently, the software development and deployment has resulted a highly sophisticated engineering process to manage and to integrate. With the introduction of artificial intelligence, automated driving has become a reality. However it has additionally increased the requirements on the system design.

One widely accepted approach to manage complexity is to divide the system into subsystems through a well-defined architecture. The architecture of an autonomous system must be suitable to guarantee that the self-driving functionality remains safe in a broad range of operational domains. The challenge is how to design the architecture of the system to be reliable and resilient to changing context.

The automotive industry has well established standards and development practices, but it is open to explore and integrate solutions from other domains like Internet of Things and Robotics. In the area of autonomous systems, the capabilities of the robotics middleware ROS2 have been used for prototyping purposes. It is an open question whether ROS2 is suitable for automotive safety relevant applications.

This master thesis addresses this challenge through evaluating the possible application of ROS2 in the automotive domain. The development consists of implementing an architecture for an autonomous driving function case-study, an Automated Parking System, which adapts to its context by switching between different operational modes.

The Automated Parking System has been implemented and validated in a simulation environment. The experiment results show which benefits bring ROS2 compared with the automotive standardised architecture AUTOSAR.

Key words: ROS2, Autonomous Driving, Software Architecture, CARLA Simulator.

Acknowledgements

I would like to be extremely grateful to Dr.-Ing. Núria Mata for the trust and opportunity to participate in her team throughout the development of my thesis. Thank you for your inspiration and enthusiasm. The totally international and youthful teamwork is something to admire. Of course, I also want to express my warmest thanks to my colleagues and friends from the Fraunhofer IKS Joao-Vitor Zacchi, René Beck together with Roger Tremosa and Yassine El Kabdani as my predecessors of work. The special support I have received from Barcelona by Dr.-Ing. Luis Javier de la Cruz Llopis deserves mention and gratitude as well.

Fraunhofer Institute for Cognitive Systems IKS gave me the opportunity to write this master thesis in the Cognitive Systems Engineering Department. I want to thank the "Bayerisches Staatsministerium für Wirtschaft, Landesentwicklung und Energie" for the funding under the project "Unterstützung des thematischen Aufbaus des Instituts für Kognitive Systeme".

Another special mention belongs to my friends from the university. Albert, Marc and Ivan, among others who have always encouraged me to reach the maximum in my professional and personal decisions. It is impossible to name all of them but it is absolutely impossible to imagine myself achieving success without the experiences and adventures lived together.

I have absolute certainty I would not have written this thesis without Andrea, who made me embark on this adventure to Germany. She is indispensable in my good and bad times, and giving me unconditional support. Thank you is not enough for all that you contribute to my life.

Finally, I want to infinitely express my gratitude to my immediate family for being always there for me. Special mention should be made to Diego, my grandfather and one of my life's references. I would never have arrived here without the courage and values he had always inculcated in me. Definitely, I would like to dedicate the master's thesis to him.

Contents

List of Figures	7
List of Tables	8
Glossary	9
1 Preface	10
1.1 Motivation	10
2 Introduction	11
2.1 State of purpose	11
2.2 Requirements and specifications	12
2.3 Methodology and procedure	12
2.4 Workplan	12
3 Background	13
3.1 Systems engineering	13
3.2 Software architecture	13
3.2.1 AUTOSAR	14
3.2.2 ROS	15
3.2.2.1 ROS1	16
3.2.2.2 ROS2	16
3.2.2.3 Communication patterns	17
3.3 Safety in Automated Systems	19
3.3.1 Safety standards	19
3.3.2 Operational Design Domain	20
3.4 Simulation environment and virtual integration	20
3.4.1 Simulator bridge	21
4 Methodology	22
4.1 Overview and system under design	22
4.2 Software Architecture	23
4.2.1 Perception	24
4.2.1.1 Radar	25
4.2.1.2 Camera	25
4.2.1.3 Lidar	26
4.2.1.4 Semantic Lidar	27
4.2.1.5 GNSS	28
4.2.1.6 IMU	28
4.2.2 ODD Handling	28
4.2.3 AS Mode Manager	29
4.2.4 Localization	31
4.2.5 Drive Planning	31
4.2.6 Motion	33
4.3 ROS2 Design	33

4.3.1	CARLA-ROS-Bridge packages	35
4.3.2	CARLA World	35
4.4	APS Deployment in ROS2	35
4.4.1	Nodes	35
4.4.2	Topics and messages	36
4.4.3	Node - Topic Architecture	37
4.5	Implementation	41
4.5.1	Waypoints	41
4.5.2	External Cloud	41
4.5.3	Vehicle functions	42
4.5.4	Parking maneuver selection	43
5	Results	47
5.1	APS integration in ROS2	47
5.2	ROS2 versus AUTOSAR	51
6	Conclusions and Future Development	53
	Budget	54
	Environment Impact	55
	Bibliography	56
	Appendices	60
A	System uses cases and constraints	61
B	CARLA Simulator	63
B.1	Parking spots	63
C	ROS2 architecture topics	65
D	ROS2 architecture messages	67
E	ROS2 rqt-graph	68
F	ROS2 sequence diagram example	69

List of Figures

2.1	GAANT diagram of the project	12
3.1	ROS2 Interfaces	18
3.2	Prototype setup	19
3.3	ODD taxonomy domain	20
3.4	Carla-ros-bridge functionality	21
4.1	Automated Parking System (APS) and the External Cloud Service (ECS) .	22
4.2	ROS2 Software Architecture	24
4.3	Vehicle sensors	25
4.4	RVIZ automated parking system architecture	27
4.5	Mode Manager state machine	31
4.6	Parallel parking maneuver	31
4.7	Perpendicular parking maneuver	32
4.8	ROS2 Automated Parking System workspace structure	34
4.9	ROS2 nodes	36
4.10	ROS2 architecture	39
4.11	Parallel parking maneuver waypoints	42
4.12	Outdoor parking maneuver waypoints	43
5.1	ROS2 architecture flowchart	48
5.2	Sequence diagram example PM_Forward out of ODD situation	49
5.3	Obstacle detection example.	50
5.4	Spider diagram of AUTOSAR Classic Platform and ROS2 comparison . . .	52
B.1	CARLA World - Town 5	63
B.2	Town_5_Opt parking zones	64
B.3	Outdoor parking graph	64
E.1	ROS2 rqt-graph plot	68
F.1	Sequence diagram example PM_Forward in correct situation	69

List of Tables

3.1	AUTOSAR Classic vs AUTOSAR Adaptive	15
3.2	ROS1 vs ROS2	16
4.1	System requirements	23
4.2	Radar message	25
4.3	Camera message	26
4.4	Lidar variables	26
4.5	Semantic lidar message	27
4.6	GNSS message	28
4.7	IMU message	28
4.8	Predefined ODD	29
4.9	Mode manager module input signals	30
4.10	Mode manager module output signals	30
4.11	Drive Planning states	32
4.12	Topic example	36
4.13	CarlaEgoVehicleControl message	37
4.14	Topics used in the communication	40
4.15	External Cloud signals	42
5.1	APS most relevant signals	50
5.2	ROS2 vs AUTOSAR Classic	51
6.1	Equipment costs	54
6.2	Personal costs	54
A.1	Use Cases of the system	61
A.2	Constrains of the system	62
C.1	Topic list	65
D.1	CarlaWeatherParameters message	67
D.2	Pose message	67

Glossary

API: Application Programming Interface

APS: Automated Parking System

AUTOSAR: Automotive Open System Architecture

BMW: Bayerische Motoren Werke GmbH

CAN: Controller Area Network

CPU: Central Processing Unit

DDS: Digital Data Storage

ECU: Electronic Control Unit

ISO: International Organization for Standardization

NASA: National Aeronautics and Space Administration

ODD: Operational Design Domain

OEM: Original Equipment Manufacturer

OSEK: Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug

RAM: Random Access Memory

RCL: Robot Command Language

ROM: Read-Only Memory

ROS: Robotic Operational System

RSA: Reference Software Architecture

RTE: Real Time Environment

SAE: Society of Automotive Engineers

SOA: Service-Oriented Architecture

SWC: Software Component

POSIX: Portable Operating System Interface

VDO: Vereinigte Deutscher Ota

VFB: Virtual Functional Bus

Chapter 1

Preface

After many years of studying for a telecommunications bachelor's degree and a master's degree in electronics, I realized that the world is filled with uncertainties, in fact, engineering tries to give solutions to those uncertainties. One of the great mysteries of mankind has been knowing how the brain operates, which is the computational benchmark that scientists are constantly trying to reach, since cerebrum is the most developed intelligent structure known so far. As a cognitive system, it is capable of receiving an infinite amount of data, processing these data, and reacting with the best behaviour. But the most interesting capability is that the brain always processes an adequate safety response.

This thesis has been developed at one of Europe's leading research centers, the Fraunhofer Institute for Cognitive Systems IKS in Munich, Germany. The Fraunhofer Institute for Cognitive Systems IKS investigates technologies for smart and adaptive systems to respond reliably and safely to unexpected or previously unknown situations. This center acts as a bridge between science and industry to bring innovative concepts of cognitive systems into the tangible application.

1.1 Motivation

During my master's degree I already got in contact with the automotive industry, concretely in the Nextium hardware department, at IDNEO group. However, the lack of knowledge in automotive software led me to enroll for a master thesis which would enhance my knowledge in automotive software architecture and the design of vehicle functions.

Additionally, this master thesis gives me the chance to learn not only about automotive software, but also about how to develop software in robotics. The reason for this is that the project is focused on developing an architecture for the automotive industry using a robotic operating system. The comparison of the AUTOSAR vs ROS II architectures scopes the target of this project.

Chapter 2

Introduction

Mobility has become a fundamental requirement in day-to-day life. People need to move constantly, whether for work or pleasure. Globalization and new IoT technologies have transformed the entire world in only decades. In this revolution, the automotive sector has been immersed in the adaptation of innovative functionalities and technology.

The automotive industry has reached a large consensus on standardization. This sector covers many disciplines, as for example mechanical, electronic, software, telecommunication, etc., which causes the deployment of a vehicle to become a highly complex engineering project to manage and to integrate. For this reason, the major companies in the automotive industry have reached an agreement to combine efforts in the creation of standards. The main success of this strategy has been the AUTOSAR partnership, the first architecture standardization in the field. Since the creation of AUTOSAR in 2003, the standard has become the most widely adopted automotive software architecture worldwide.

With the boom of artificial intelligence, autonomous driving has emerged as a reality. However, this involves many challenges for machine learning because the designer cannot take an infinite range of scenarios into account during implementation. Architecture must adapt to changing situations autonomously and safely. At the moment, safety is the most critical element in the development of these technologies. The architecture of an autonomous driving system must be suitable to guarantee that the automated driving functionality is safe in a wide range of operational domains. This is in fact a requirement for any autonomous system applicable to robots, drones and trains.

The development of autonomous robots began before vehicle manufacturers introduced the first automated functionalities in the car. In 2007, activities on a modular architecture for robots started with the Robotic Operational System (ROS). ROS is an open-source framework supported by a variety of tools which help developers to record data, visualize data and integrate a number of algorithms to develop autonomous functions using existing open-source libraries. The automotive industry is taking advantage of ROS2 for building prototypical autonomous functions, however it is still an open issue how to certify safety using the ROS2 framework. This master thesis evaluates the possible applications of ROS2 in the automotive domain. The results are presented through an automated parking system case study.

2.1 State of purpose

The purpose of the dissertation is to implement an architecture in ROS2 which ensures that automated driving functions are safe for well-defined operational domain. This architecture will be exemplary for an automated parking system. The final purpose is to

compare the performance of this architecture with a previous architecture implemented in AUTOSAR [1] [2].

2.2 Requirements and specifications

The most relevant requirements of the system are taken from the ISO/TR 4804 (2020) for automated driving systems. Two previous master thesis [1] [2] already implemented a simple AUTOSAR Classic Platform architecture derived from the ISO/TR 4804 (2020). An automated parking system study case was specified, designed and implemented. The Cognitive Systems Engineering department at Fraunhofer Institute IKS proposed me to study the advantages of using ROS2 for the architecture and validate the results for the same automated parking system.

2.3 Methodology and procedure

The project methodology includes a modelling part and an evaluation part. The modelling part consist on the ROS2 architecture with special focus on the automated system mode management. The evaluation part concentrates on the validation of this architecture in a simulated environment supporting ROS2, i.e. the CARLA Simulator. The crucial steps of this development are the Mode Manager and the Drive Planning modules. The results are compared with the ones in [1] [2].

2.4 Workplan

This master thesis started with the state-of-the-art on systems engineering and software architecture. The requirements defined in [1] [2] were enhanced to provide a more realistic design for the safe state. The AUTOSAR related requirements were adapted for the ROS2 framework based on a service-oriented architecture. The resulting architecture was deployed in the CARLA Simulator for the automated driving system and integrated in ROS2. The thesis ends with a comparison between AUTOSAR and ROS2 for a safety relevant automated functions. The block diagram, in Figure 2.1 shows clearly the stages mentioned.

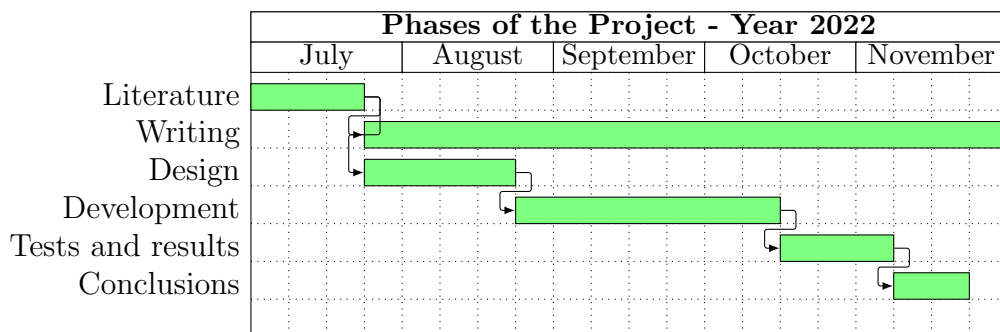


Figure 2.1: GAANT diagram of the project.

Chapter 3

Background

An overview of the industry standards for autonomous systems combined with the definition of software architectures is required to understand this dissertation. This state of the art contains a brief overview of systems engineering, an introduction to the principles of software architecture and a comparison between two standardized software architectures: Automation Software Architecture (AUTOSAR) Classic Platform and Robot Operating System 2 (ROS2).

3.1 Systems engineering

The systems engineering is a multidisciplinary approach towards the design, construction, management, performance and decommissioning of a system [3]. A method usually applied in all fields involved in the design process of a product or application, being hardware or software processes.

System engineering integrates different disciplines into a common aim, efforts and forms an appropriate structured development process [4]. The decomposed subsystems can be classified by priority, functionality and safety-level. This type of work reduces the costs of the project and at the same time improves the quality of the product [5]. System engineering uses an iterative approach, beginning by defining the system architecture and continuing by defining more accurately the requirements, communication paths and implementation.

The software architecture can be described as a hierarchical structure with different levels of abstraction. System blocks are connected to the others and the developer focuses on what, where, and how to develop the tasks [6].

3.2 Software architecture

When an engineering project such as automated vehicle parking is developed, it is essential to structure the complex system into subsystems. Well-defined and clean architecture gives their advantages, their applicability, etc. All this information is required to choose which architecture fits better the system under design.

Software architecture is the high-level structure of the system which defines the sub-modules that will implement the functions. Software architecture is the structure of the system explained at high-level behavior which provides a standardized framework to develop the software of the system and give quality, maintainability, and correct performance to the system [7][8]. Like a set of structures needed for the system, which comprises software elements, relations among them, and properties of both [9]. The software architecture

is the abstraction where the architecture defines the components and the interconnection between components without specifying internal details [10].

There exist different architectural views which can be combined to provide the best description for all types of system complexities. There is not only one correct solution for the architectural design of a system. The solution is open to what the developer wants [11]. Software architecture provides a pattern of communication between stakeholders and indicates the most important decisions [12]. At the same time, allowing the transfer and reuse of the system abstraction.

The logical architecture abstracts the components of the system in a modular way and captures their dependencies, without any implementation details. Only the interaction between the components is taken into account. The logical architecture is technology agnostic. The application software will be later developed according to a specific domain on a Reference Software Architecture [13].

Reference Software Architecture contains the elements and relations provide templates for concrete architectures in a particular domain. This architecture allows the application software to specify and implement independently where the software is going to be executed, providing a list of functions and their interfaces (APIs) on the one hand but on the other hand, the interactions with each of the functions that are outside the scope of the reference architecture [14] [15].

In the automotive domain, two Reference Software Architecture have been standardized by the AUTOSAR consortium, namely the AUTOSAR Classic Platform and the AUTOSAR Adaptive Platform. [16]. Whereas, in robotics domain, ROS is a well established RSA through the name of Robotic Operational System which involves this project development and is presented during the next sections.

3.2.1 AUTOSAR

The AUTomotive Open System Architecture (AUTOSAR) development partnership was founded in July 2003 by BMW, Bosch, Continental, Daimler Chrysler, Siemens, VDO, and Volkswagen to develop and establish an open industry standard for automotive E/E architecture. Later, Ford Motor Company, Peugeot, Citroën Automobiles S.A., Toyota Motor Corporation, and General Motors added to the partnership [17]. This framework has become a global established developer partnership between vehicle manufacturers, suppliers service providers, and automotive electronic, semiconductor, and software companies.

Today vehicles integrate different Electronic Control Units (ECUs) which act as small computers programmed with embedded software. These electronic components sometimes are not prepared for the increasing requirements and complexity of this software systems due to computational limitations [18].

The objective of this standards is to manage complexity of highly integrated E/E architectures through increased reuse and exchange of software modules between OEMs and suppliers. AUTOSAR defines exchange formats for the configuration process for the basic

software stack and the integration of the application software into the ECU. AUTOSAR enables open E/E system architectures for future intelligent mobility, supporting high levels of dependability, especially safety and security [19]. The aim is to define a common specification language to describe the interfaces between components and communication mechanisms to guarantee integration of the software in a vehicle [20]. Vehicle manufacturers and subsystem’s suppliers are able to cooperate in projects, while keeping and protecting their know-how (business) at implementation level.

The partnership started with the AUTOSAR Classic Platform to support the development of deeply embedded ECU, suitable for functions with high real-time demands and strict safety requirements. Then these devices needed low-power computing of around 1000 DMIPs (Dhrystone Million Instructions per Second) whereas today the system requires more computation [21]. In 2016, the partnership started the AUTOSAR Adaptive Platform to support the development and integration of application software in high-performance computers of around 20.000 DMIPs [22][23]. The AUTOSAR Adaptive Platform focuses on supporting the growing trend of the connected and autonomous vehicle. The comparison between AUTOSAR platforms is shown in Table 3.1.

Table 3.1: AUTOSAR Classic vs AUTOSAR Adaptive [22][23].

AUTOSAR Classic Platform	AUTOSAR Adaptive Platform
Operating system based on OSEK	Operating system based on POSIX (PSE51)
Code execution into ROM	Code Execution into RAM
All applications have the same addresses	Each application has its own address
Optimized for CAN signal-based communication	Designed for signal-oriented communication
Fixed task configuration	Support of multiple dynamic scheduling strategies
Specification	Specification and code
C Programming	C/C++ Programming

The Adaptive Platform (AP) supports complex applications, with maximum flexibility and scalability in processing and computing, this platform follows service-oriented architecture (SOA) [21]. The SOA makes each component more independent and free of interference which guarantees safety and security. This last version aims to be adaptive to different product development processes, more concretely in agile-based processes.

The AP stands planned dynamics in the sense of enabling the incremental deployment of applications in short iterations cycles. Thus, it allows continuous development and integration processes for the complete product life cycle of the vehicle. The communication patterns of AUTOSAR are similar to the ROS communication patterns. In the next chapters the robotic middleware ROS is explained in detail.

3.2.2 ROS

In robotic systems, the history of robot software is long and storied, going back more

than 50 years. The appearance of robots like Shakey [24] and the deployment of robotics frameworks which provide architectural methods to decompose complex software into more manageable pieces represented a milestone in the robotic field.

What eventually would become ROS, started to gestate at Stanford University, 2007, through two PhD students. The founders Eric Berger, Keenan WYROBECK and Scott Hassan created the Willow Garage which was a technology incubator for the first commit of ROS code. The concept evolved until today that it has become the most standardized middleware and reference software architecture for robotics. Some years ago, in 2014, NASA announced the first robot to run ROS in space, the Robotnaut 2 [25].

3.2.2.1 ROS1

The Robot Operating System is an open-source robotic middleware which contains lots of software libraries and tools for easily building robot applications. It is a set of software frameworks for robotic software development produced by a global community of developers who implements and improve constantly the software. The middleframework contribute with the tools, libraries and capabilities to create a robotic application through ROS1. Nevertheless, this platform can be used in multiple domains beyond robotics, for example automotive, space, industrial, aeronautic, and others [26]. ROS1 concerns architectural and engineering limitations have been address by the ROS2 initiative.

3.2.2.2 ROS2

The ROS2 platform was created to support real-time systems, because of the limited support in ROS1. This new version has the advantage of modern libraries, the revision of ROS API and giving real-time system code [25].

The new platform design has better distributions of systems approach where requirements are separated into independent components [27]. The second principle is the abstraction by getting the benefits of exposing the details of a component making easy the substitution of an alternative. This new framework is asynchronous, working across multiple time domains [29]. The last principle is modularity because the ecosystem is organized into many federated packages, as opposed to a single code base [30]. The main reasons to use one platform or another is depicted in Table 3.2.

Table 3.2: ROS1 vs ROS2 [28].

Category	ROS1	ROS2
Network transport	Bespoke protocol built on TCP/UDP	Existing standard (DDS)
Network architecture	Central name server	Peer-to-peer discovery
Platform support	Linux	Linux, Windows and macOS
Client libraries	Independently in each language	Common underlying C library
Node vs process	Single node per process	Multiple nodes per process
Threading model	Callback queues and handlers	Swappable executor

Category	ROS1	ROS2
Node state management	None	Lifecycle nodes
Embedded systems	Minimal experimental support	Commercial support
Parameter access	Auxiliary protocol in XMLRPC	Implemented using service calls
Parameter types	Type interfered when assigned	Type declared and enforced
Security	Suffers malicious misuse	Secure against types of misuse
Cryptography	None	Encryption and access control
Computing	Normal computing	Real-time computing
Systems	Research	Commercial critical safety

In the ROS2, the processes are represented as nodes interconnected by edges known as topics, where messages are sent and received, as bus functionality. Making possible the node-to-node communication through the topics [31]. Objects in the nodes have to be statically allocated for the correct behaviour in real-time system using a communication within the ROS2 network realised by following the Data Distribution Service (DDS) [32] standard defined by the Object Management Group (OMG) [32]. Chapter 3.2.2.3 helps to understand the communication concepts of this middleware.

Self-driving vehicles must perform decisions based on real-time sensors, thus ensuring high reliability on functional safety. Theoretically, the only difference between self-driving vehicles and robots is the final application. ROS1 cannot fulfill high reliability and real-time performance [33].

ROS2 is considered the solution for automated driving systems. Some studies are researching on the implementation of ROS2 architecture for autonomous driving cars maintaining the advantages such as a distributed architecture and standardized message types. It provides the necessary reliability and real-time performance to allow the deployment of automated vehicle functions, such as the automated parking function used in this master thesis. Recent works [34] [35] show a good evaluation in different scenarios, verifying the ROS2 architecture usability and variability.

3.2.2.3 Communication patterns

This Robotic Operational System is fundamented on communication patterns, more specifically, topics, services and actions under the concept of a node. Moreover, the robotic software also provides patterns for parameters, timers, launch, and other auxiliary tools that can be used to design a robotic system. The environment is composed by nodes, topics, messages, services, actions and discoveries.

The node serve for a single and modular purpose in the robotic system, it communicates with other nodes through the topics and each of them has a unique name [36]. Each node contains programming code for specific functions of each node and can receive (subscribe) and share (publish) information to other nodes. The information transmitted is called message. The topic is similar to a bus which send and receive messages as for example

actuator commands, data sensor, etc. and only it allows to transfer of specific data. The message has a concrete structure within its own parameters.

The service is a type of communication mode to achieve a result or advertisement, as well as a request/response pattern where the client can request to generate a response. The action is a goal-oriented and asynchronous communication interface, like service but with a request, response, periodic feedback, and the ability to be canceled [36].

The discovery is an automatic process through which nodes determine how to talk to each other. Nodes periodically advertise their presence and when they go offline. When a node is started, it advertises its presence to other nodes on the network with the same ROS domain [37]. Nodes respond to this threat with information about themselves so that the appropriate connections can be made, and the nodes can communicate correctly. All communication connections are shown more in detail in Figure 3.1.

The nodes are able to publish or subscribe multiple topics. The ROS client libraries allow nodes to be written in different programming languages to communicate with each other. A core ROS client library (RCL) implements common functionality needed for the ROS APIs in different languages. The following client libraries maintained by the ROS II are rclcpp (C++ client library) and rclpy (Python client library) [38]. The general visualization of the Robotic Operational System structure is shown in Figure 3.2.

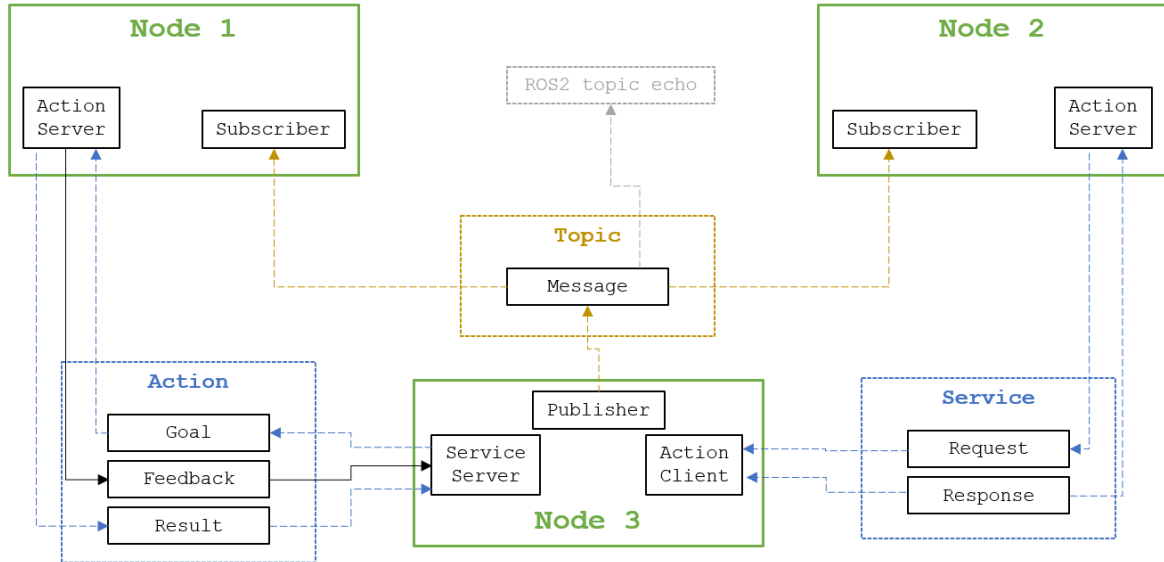


Figure 3.1: ROS2 node interfaces: Topics, services, and actions [28].

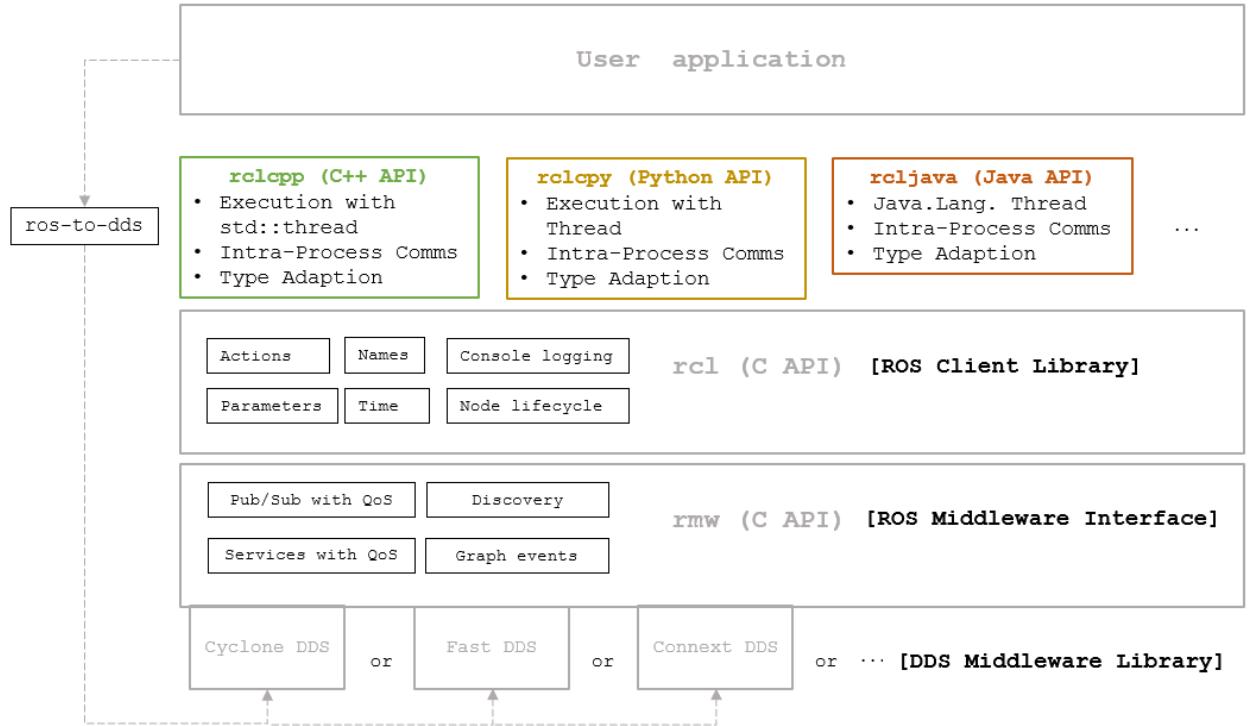


Figure 3.2: ROS2 Client Library API Stack [28].

3.3 Safety in Automated Systems

3.3.1 Safety standards

In the automotive sector, there are different safety standards related to automated systems and these standards must be fulfilled to guarantee safety. Some of the most relevant standards of this dissertation are the next ones:

1. SAE Automated Driving Levels
2. ISO-26262
3. ISO/TR-4804
4. ASAM OpenODD Concept Paper

The SAE (J3016) describes various driving automation levels but without strict requirements. The ISO 26262-1:2018 [39] is specific for automotive systems but doesn't consider yet The requirements on automated systems.

The ISO/TR-4804 "Road vehicles - Safety and cybersecurity - Design, verification, and validation" [40] does focus on safety and cybersecurity in automated driving systems, outlining a framework to develop, validate, fabricate, and operate an automated driving system. This standard includes some technical verification and validation methods from SAE J3016, in most cases only for levels 3 and 4 [41].

The Association for Standardization of Autonomous and Measuring Systems (ASAM) is an international association of car manufacturers, suppliers and engineering providers [42]. ASAM is currently leading the Operational Design Domain (ODD) technical standardization.

3.3.2 Operational Design Domain

The Operational Design Domains are “Operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics” [43]. The ASAM OpenODD introduces important use-cases and requirements that automated driving systems have to take into account to guarantee safety and to define the taxonomy.

The ODD should be valid for the whole operational life of an automobile and takes part in the operational and security aspects of connected automated vehicles, specifying which environment parameters is able to manage. The ODD has a huge impact on the design of the functions and capabilities since it conditions the utility of the vehicle function [44]. Some domain taxonomies of an automated driving system are exemplary mentioned in Figure 3.3.

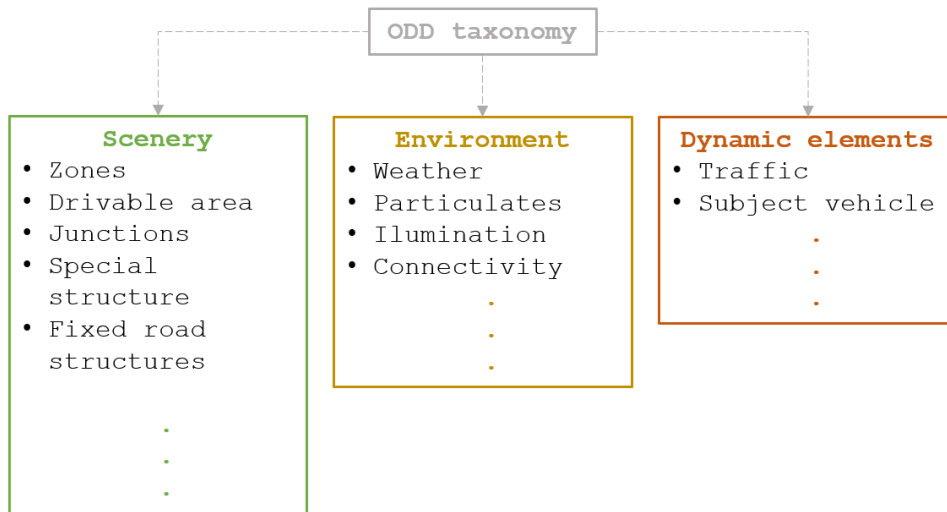


Figure 3.3: ODD taxonomy domain.

3.4 Simulation environment and virtual integration

In the last years, different tools to an autonomous system in a simulated environment have been developed. The simulator has to test the system in the most possible realistic situations. The simulator software is capturing data through sensors to perception data and environmental data for path planning.

CARLA is a software to support the development, training, and validation of the automated vehicle systems. This software is recognized by many leading companies in the industry such as Intel, Toyota, Samsung Europe, CVC, Valeo/KI Delta, Baselabsa and various Fraunhofer institutes. This simulator allows the developer to modify cities, roads, traffic, weather, etc., and to adapt the simulation to the real situation where the vehicle can be involved [45][46]. This master thesis decides to simulate the system in CARLA Simulator because it is open-source software and it is the most accurate equilibrium one for good graphics, good vehicle dynamics, a good number of sensors, good driver models, and good traffic flow simulations. But one of the most remarkable specifications of this simulator is the ground truth.

The CARLA Simulator provides interesting packages for the purpose of this project, to connect the simulator with the ROS2 environment. These packages fit in carla-ros-bridge which considers all the communication information between ROS2 architecture and functions and the CARLA client.

3.4.1 Simulator bridge

The CARLA developers provide an open-source bridge, known as carla-ros-bridge [47]. The bridge contains the tooling required to test the ROS2 architecture in the simulator. These packages enable the acquisition of data from the CARLA Simulator providing many tools to communicate the robotic environment and the simulator environment, as describe the diagram in Figure 3.4.

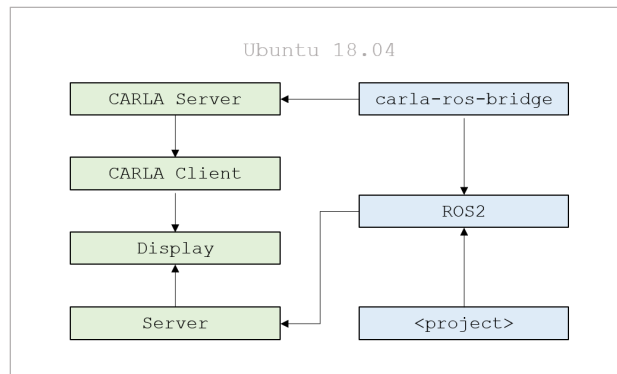


Figure 3.4: Carla-ros-bridge functionality.

Chapter 4

Methodology

4.1 Overview and system under design

The development of this project takes into account systems engineering and the industry development processes such as the V-Model, as well as concepts derived from the safety standards. A modular and scalable software architecture has been designed.

The architecture is generic and supports future developments for other vehicle functions only adapting the requirements to the system. This research continues the work of two previous projects based on a safe architecture in AUTOSAR [1][2].

The same as in [1][2], the architecture has been implemented for an Automated Parking System (APS). The car parks autonomously once the driver has selected a free parking slot via a mobile application. The APS assumes an External Cloud System (ECS) provider which manages parking slots, Figure 4.2.

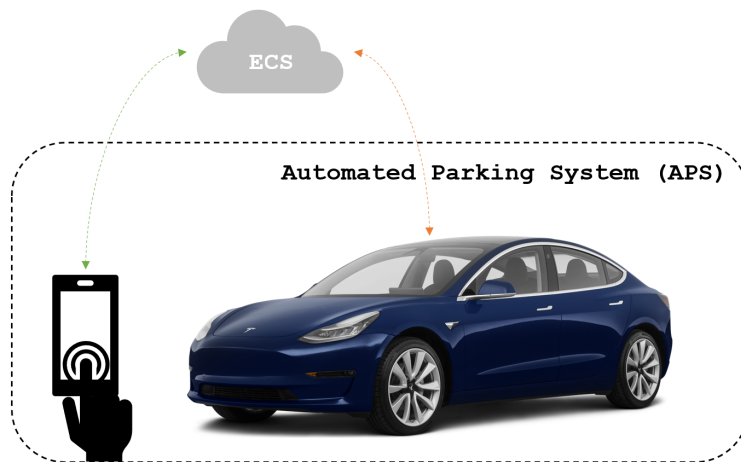


Figure 4.1: Automated Parking System (APS) and the External Cloud Service (ECS).

The customer gets out of the vehicle and activates the APS functionality through the mobile application. Once the parking system is activated and the parking slot is selected, the vehicle's Mode Manager takes control of the situation and starts the automated parking maneuver. The system must check with the different sensors that the maneuver can be executed safely. If the system detects unsafe environmental conditions, the Mode Manager has to activate the safe mode maneuver.

The system requirements of the APS are based on the use cases, constraints and requirements in [1] [2] and have been extended for a more realistic behaviour of the safe mode. In

the Table 4.1, the system requirements that are been reused are indicated with a citation.

Table 4.1: System requirements.

Requirements	Definition
SYS-1	APS follows up the state of the parking maneuver in real-time.
SYS-2	APS supports parallel parking mode and perpendicular parking mode.
SYS-3	APS drives to the location provided by the ECS.
SYS-4 [1]	The system calculates all possible routes in real-time and manages the best fitting one, it can abort the maneuver if safety is not guaranteed.
SYS-5 [1]	Real-time status information through the APS App.
SYS-6 [1]	When APS is active, only the system can modify the route of the parking maneuver.
SYS-7 [1]	APS notifies the driver when the vehicle is parked.
SYS-8	APS detects any obstacle or ODD exit while maneuvering.
SYS-9 [1]	APS avoids obstacle collision by stopping the car during the maneuver.
SYS-10	APS drives the car to a safe parking area when it detects an ODD exit.
SYS-11	ECS provides the localization of the spot, park mode and type of parking mode.
SYS-12	Parking speed is limited to 20 km/h forward and backwards but decreased to 5-10 km/h in the last meters of the maneuver.
SYS-13 [1]	APS aborts if any collision occurs or miss-recognize the scenario.
SYS-14 [1]	APS aborts if the user stops the function via APS App and follow the new route provided by the APS App.
SYS-15 [1]	APS locks the vehicle whenever the system is active or the vehicle is parked.
SYS-16	APS manages the parking location modes as parallel or perpendicular.

4.2 Software Architecture

The aim of the project is to develop a software architectural that implements a minimal automated driving and parking system. The architecture model consists on a "Sense - Plan - Act" abstract architecture which is based on perception, data process and action. Figure 4.2 shows a high-level component diagram.

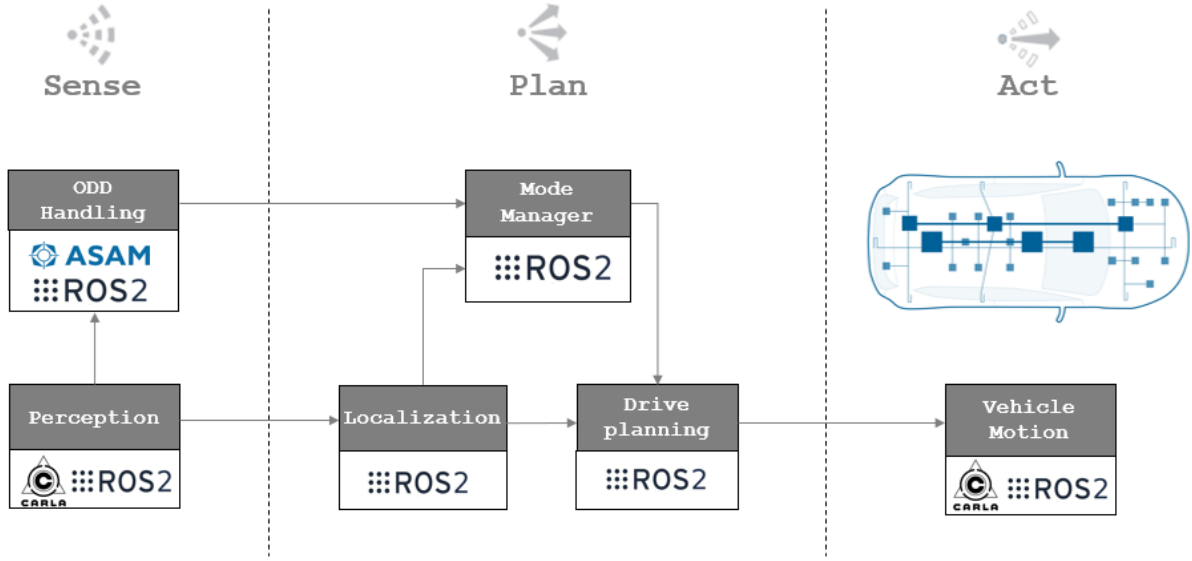


Figure 4.2: APS Safety Architecture Model.

The perception collects the information of the different sensors, i.e. radar or lidar. This information is processed by the system to recognize the vehicle’s surroundings and to be able to anticipate obstacles and safety hazards. The ODD Handling module identifies the operational context. The Mode Manager switches between operational modes to ensure safety. The Drive Planning decides, depending on the mode switches, the correct maneuver that Vehicle Motion is going to implement. All the modules are implemented in the ROS2 environment through nodes, topics, and messages [34][48]. The following chapters consist of a concise description of the features of each component of the system.

4.2.1 Perception

The perception is responsible for taking data from the different sensors installed in the vehicle and identifying important information [48]. In this project, supported sensors are radar, lidar, camera, GNSS, and IMU. Figure 4.3 shows the localization of the sensors on the vehicle. To create a reliable and robust system, it is necessary to combine the information of several sources. All the information provided by the sensors is explained in the next sub-sections.

When designing the system, it is essential to model each sensor component and define where the component will be deployed in the vehicle. In this project, the Automated Parking System is validated in the CARLA Simulator. CARLA has modelled radar, camera, Lidar, GNSS, IMU, and semantic Lidar sensors [49]. The following chapters define the type of data computed by each sensor.

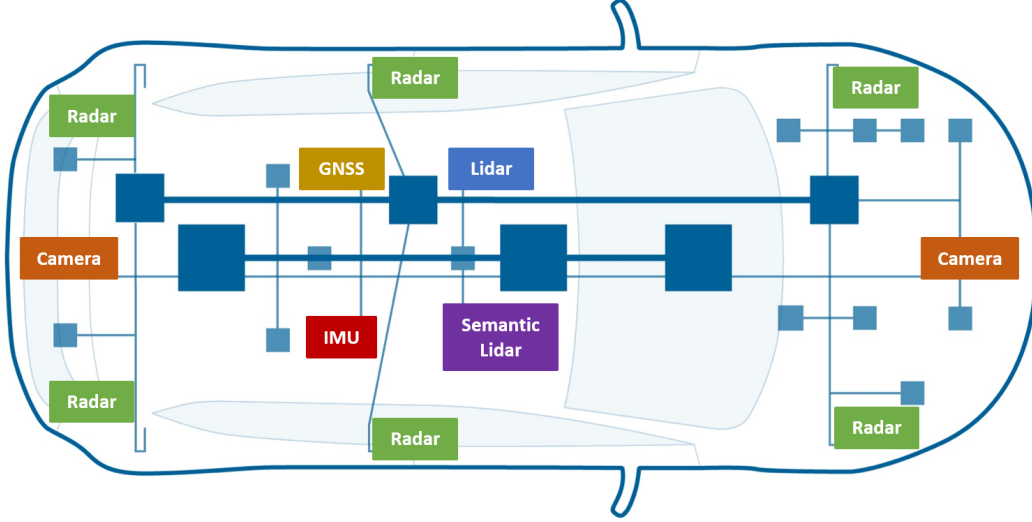


Figure 4.3: Vehicle sensors.

4.2.1.1 Radar

The Radar component converts the microwaves echo signals into electrical signals which can be processed into digital signals. A radar can detect solid obstructions from cement, metals, glass and other wall material like people, animals, and trees. The advantage of the radar is that weather conditions and light incidence don't affect its capability to recognize objects. Furthermore, the radar detects the velocity and motion of objects that could appear in its surroundings [50].

To have relevant information of the surroundings of the car, it is recommended to install four radar sensors, at the front, laterals and back, as shown in Figure 4.3. The radar sensor delivers the message through the carla-ros-bridge described in the next Table 4.2.

Table 4.2: Radar message.

Variable	Units	Type	Description
Altitude angle	rad	Float	Altitude angle signal
Azimuth angle	rad	Float	Azimuth angle signal
Depth	mts	Float	Depth of the signall
Velocity	m/s	Float	Velocity signal

The radar message is published in the topic `/carla/ego_vehicle/radar`.

4.2.1.2 Camera

The camera provides high-resolution images of the surroundings of the vehicle and allows to detect objects. Recognized obstacles can be compared with other sensor information to

verify if there is or not an object. The main disadvantage is that weather conditions affect the camera detection [50]. Image processing has high computational demands. Cameras provide 120 degrees of vision, it is enough to install one camera at the front and another at the back, as demonstrated in Figure 4.3. The camera message has four variables, Table 4.3.

Table 4.3: Camera message.

Variable	Units	Type	Description
Fov	degrees	Float	Horizontal field
Height	pixels	Int	Image height
Width	pixels	Int	Image width
Raw	-	Float	Array of RGB pixels

The camera message is published into `/carla/ego_vehicle/camera`.

4.2.1.3 Lidar

Light Imaging Detection and Ranging (LIDAR) identifies objects in the surroundings, measuring shape, size, and distance. This component uses laser light pulses to scan the surroundings shooting millions of laser signals, which are reflected on the surfaces of objects around and returned to the receiver incorporated into the module. The LIDAR creates a 3D model of the vehicle’s surroundings [50].

The advantage of the lidar is the identification of elements with higher resolution than radar, but the disadvantages are the high price of these component sensors and the affection of weather conditions to the measurements. This is the main reason to locate only one in the vehicle, as indicated in Figure 4.3. Currently, vehicles that contain autonomous driving systems are equipped with a single large 360-degree LIDAR sensor on the roof that provides a complete view of the surroundings. The message variables are disclosed in Table 4.4.

Table 4.4: Lidar variables.

Variable	Units	Type	Description
x	meters	Float32	Position x
y	meters	Float32	Position y
z	meters	Float32	Position z
CosAngle	degrees	Float32	Orientation angle
ObjIdx	-	UInt32	Type of the object detected
ObjTag	-	UInt32	Tag of the object detected

The lidar message is published into `/carla/ego_vehicle/lidar`. The next Figure 4.4 gives an idea of how RVIZ visualizer plot the points received from the topic.

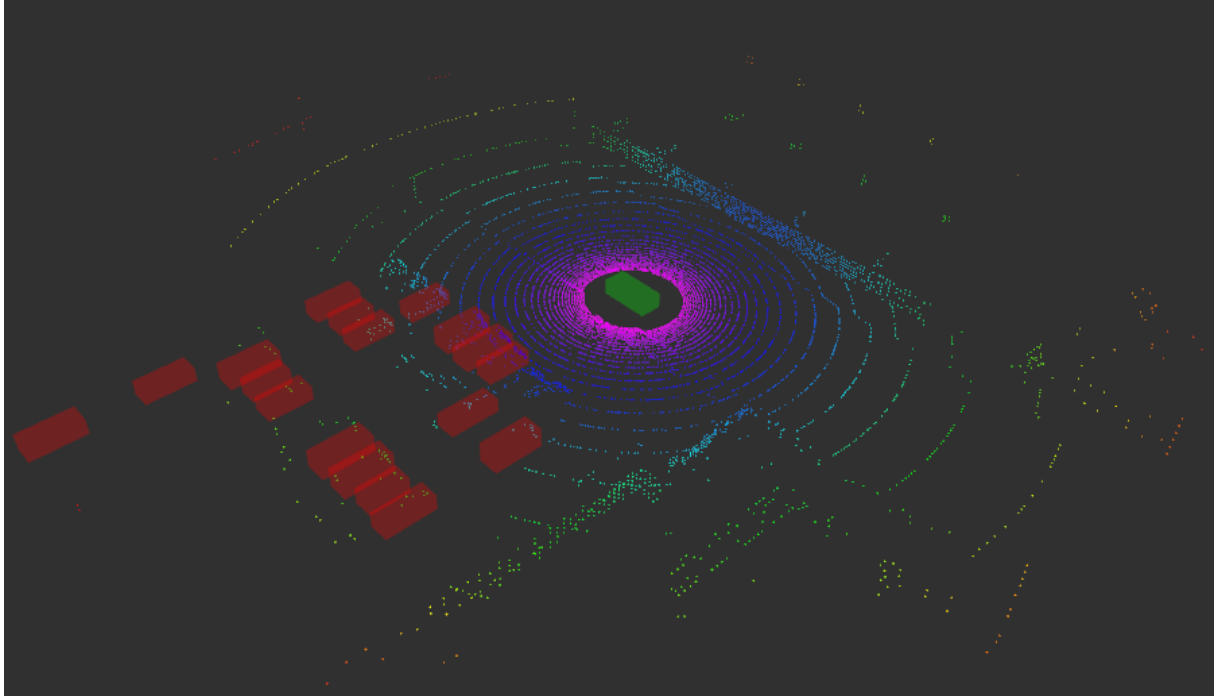


Figure 4.4: RVIZ automated parking system architecture.

4.2.1.4 Semantic Lidar

Semantic Light Imaging Detection and Ranging are considered rotating LIDAR using ray-casting. The differences within the Lidar sensor are that the semantic lidar includes more data for each point [50]. For more information about message variables look at Table 4.5.

Table 4.5: Semantic lidar message.

Variable	Units	Type	Description
x	meters	Float32	Position x
y	meters	Float32	Position y
z	meters	Float32	Position z
CosAngle	degrees	Float32	Orientation angle
ObjIdx	-	UInt32	Type of the object detected
ObjTag	-	UInt32	Tag of the object detected

The semantic lidar message is published into `/carla/ego_vehicle/semantic_lidar`.

4.2.1.5 GNSS

Global Navigation Satellite System (GNSS) provides geospatial global positioning coverage autonomously, using triangulation to determine the position of a receiver in three-dimensional space by calculating the distance between the vehicle and several satellites [50].

The installation of one sensor is sufficient to achieve accurate information. One advantage is that weather conditions have minimal impact on these elements, due to their operational frequency at around 1.575 GHz is insensitive to weather conditions [50]. The message data is shown in Table 4.6.

Table 4.6: GNSS message.

Variable	Units	Type	Description
Altitude	meters	Float32	Altitude signal
Latitude	degrees	Float32	Latitude signal
Longitude	degrees	Float32	Longitude signal

The GNSS sensor message is published into `/carla/ego_vehicle/gnss`.

4.2.1.6 IMU

Inertial Measurement Unit (IMU) consists of an accelerometer and a gyroscope. The accelerometer measures a vehicle's three linear acceleration components whereas the gyroscope measures a vehicle's three rotational rate components. With the information from the GNSS sensor, which provides the initial location of the vehicle, the IMU can provide current information on the current vehicle location and orientation [50]. This sensor is insensitive to weather conditions, due to independence from the motion of the vehicle. More characteristics are presented in Table 4.7.

Table 4.7: IMU message.

Variable	Units	Type	Description
Accelerometer	m/s^2	Vector3D	Acceleration value
Compass	radians	Float	Compass orientation
Gyroscope	rad/s	Vector3D	Gyroscope orientation

The imu message is published into `/carla/ego_vehicle/imu`.

4.2.2 ODD Handling

The ODD Handling component processes the information provided by the sensors to

recognized context information. Context is designed according to the attributes of the operational design domains using the standard TR/PAS 1883:2020.

The ODD Handling component considers the scenarios and attributes listed in Table 4.8. With the peculiarity of the climatic phenomena scenarios, exceeding the 70% range in some attributes constitutes exiting the ODD. When the system exits the designed ODD, the safe mode is activated.

Table 4.8: Predefined ODD based CARLA simulator support.

ODD taxonomy		Attribute	Sub-Attribute	Threshold
Scenario	Derivable area	Type	Urban roads	100
		Geometry	Up-slope	-
			Down-slope	-
			Level plane	-
	Structures	Buildings	-	-
		Street lights	-	-
	Special structures	Pedestrians crossing	-	-
Environment	Weather	Water retention		70
		Wind	-	70
		Rainfall	-	70
		Fog	-	70
		Sunny	-	-
	Illumination	Day	-	-
		Night/Low light	-	-
		Cloudiness	Clear	-
			Partly cloudy	-
			Overcast	-
		Artificial illumination	-	-
Dynamic elements	Traffic	Parked vehicles	-	-
		On road vehicles	-	-

4.2.3 AS Mode Manager

The AS Mode Manager is the essential component of the architecture to guarantee safety. It receives information about the context from the ODD Handling component. This signals received are explained in Table 4.9. If the system exits the operational design domain (out_of_ODD), then the AS Mode Manager will enable the M_Safe mode. In the case an obstacle is detected (hazard_detection), then the Drive Planning will be notified. The Drive Planning use the three output signals of the AS Mode Manager to choose the motion function.

The component receives inputs signals from the ECS required to know the properties of the parking slot (APS_activation, location_selection, parking_selection). The As Mode Manager is modelled with a state machine. Depending on the input signals in Table 4.9, the component assigns one or another value to the output signals shown in Table 4.10.

Table 4.9: Mode manager module input signals.

Topic	Units	Type	Description	Origin module
APS_activation	-	Int8	Activation of the system	External cloud
location_selection	-	Int8	Type of spot selected	External cloud
parking_selection	-	Int8	Park or unpark	External cloud
hazard_detection	-	Bool	Hazard detection	ODD Handling
out_of_ODD	-	Bool	Out of ODD detection	ODD Handling
APS_done	-	Bool	Parking maneuver finished	Drive Planning

Table 4.10: Mode manager module output signals.

Topic	Value name	Value	Description
APS_vehicle_mode	VEH_DRIVING	0	When vehicle is driving
	VEH_PARKING	1	When vehicle is parking
	VEH_UNPARKING	2	When vehicle is unparking
	VEH_STOPPED	3	When vehicle is stopped
APS_state_mode	APS_OFF	0	APS deactivated
	APS_ON	1	APS activated
	APS_SAFEMODE	2	APS safe mode activated
APS_maneuver_mode	PARALLEL_PARKING	0	Parallel parking
	PERPENDICULAR_PARKING	1	Perpendicular parking

The AS Mode Manager state machine has three different states. The OFF_state indicates when APS is not activated, when APS maneuver is completed and when the vehicle doesn't detect more hazard or out of ODD situation after entering the EMERGENCY_state. EMERGENCY_state is compulsory to create when hazard_detection, out_of_ODD and APS_activation are activated. The rest of the cases while APS_activation is activated, the system is going to be in PARKING_MANEUVER_state. The states and transitions of the state machine are depicted in Figure 4.5.

In PARKING_MANEUVER_state, depending on the input signal values, the outputs have one or another value. These logic is explained in the next implementation chapters.

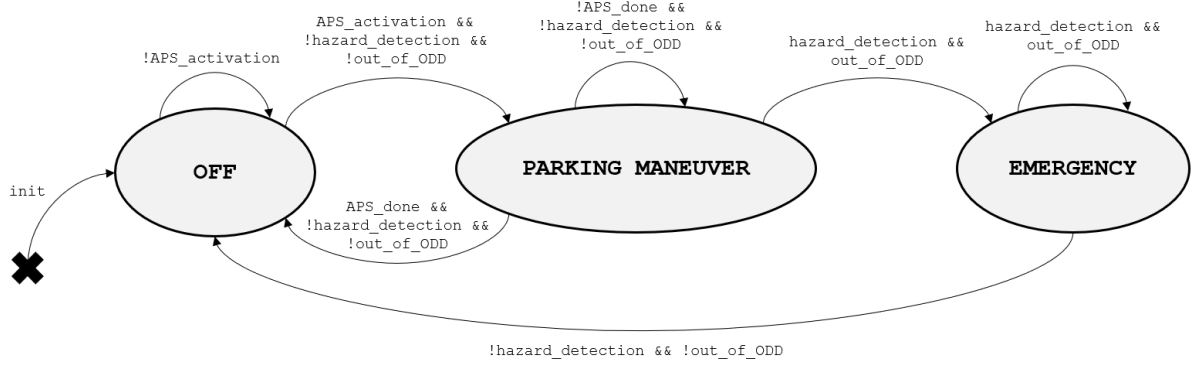


Figure 4.5: Mode Manager state machine.

4.2.4 Localization

The localization module is responsible for identifying the vehicle's surroundings by analyzing and combining sensors' information. This unit delivers the surrounding info to the Drive Planning and AS Mode Manager components.

In this project, the current position of the vehicle and the surrounding data is provided by some topics where sensors real-time information is published. As an example, the current location of the vehicle can be collected through some ROS2 topics defined in the implementation part. The combination of this data with the other sensors data allows the vehicle to have a clear situation about the surrounding of the vehicle. This component emulates the behaviour of a complex sensor structure.

4.2.5 Drive Planning

The Drive Planning component defines the function maneuver without colliding with any obstacle. This component needs Localization and As Mode Manager output signals. The component also requires the coordinates, provided by the ECS, where the vehicle has to park. It obeys traffic rules, but to avoid a crash situation, traffic laws can be overridden by collision avoidance maneuvers.

The Localization provides the position of the surrounding objects. With this information, the parking mode and the location of the parking slot, the Drive Planning component starts the parking maneuver. The Drive Planning implements four different maneuver functions: Park and unpark, shown in Table 4.11, using parallel and perpendicular maneuvers, shown in Figures 4.6 and 4.7.

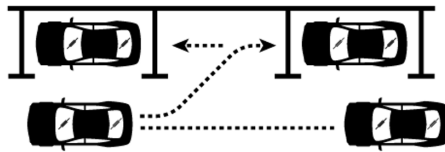


Figure 4.6: Parallel parking maneuver.

Apart from these four functions, there is a safe maneuver M_Safe if something unexpected happens.

M_Safe function stops the car if there is a threat as for example a collisions or the system gets out of the defined ODD. If the system is out of the Operational Design Domains (ODD), the vehicle drives autonomously to the initial position whereas if the system is going to collide, the vehicle effectuates an emergency stop. This project does address in the perpendicular backwards parking since this is not considered a required maneuver.

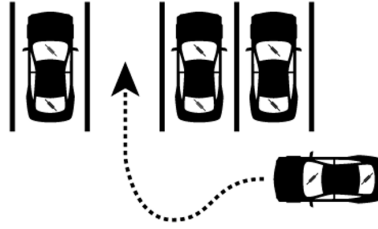


Figure 4.7: Perpendicular parking maneuver.

The Drive Planning receives the three AS Mode Manager signals to choose which maneuver to implement. APS_state_mode defines whether the automated parking system has to be activated or deactivated. As long as the signal is disabled, the drive planning remains on standby with no movement of the vehicle. When this signal activates, the component proceeds to engage the parking maneuvers (PM_ForwardBackwards, UM_BackwardForwards, PM_Forward, UM_Backward).

Table 4.11: Drive Planning states when APS_state_mode is ON.

		APS_vehicle_mode			
		VEH_DRIVING	VEH_PARKING	VEH_UNPARKING	VEH_STOPPED
APS_maneuver_mode	PARALLEL	-	<i>PM_ForwardBackwards</i>	<i>UM_BackwardForwards</i>	-
	PERPENDICULAR	-	<i>PM_Forward</i>	<i>UM_Backward</i>	-

If the system is activated, APS_vehicle_mode and APS_maneuver_mode signals get into the scene. The APS_vehicle_mode indicates the status of the vehicle. If this signal reports

VEH_PARKING or VEH_UNPARKING states, then the module verifies in which type of parking slot the vehicle is going to maneuver. This verification is done by means of the signal APS_maneuver_mode. Otherwise, if APS_vehicle_mode displays VEH_DRIVING or VEH_STOPPED the drive planning stays deactivated. The Algorithm 1 of the implementation section shows more in detail how the Drive Planning decides the correct maneuver.

4.2.6 Motion

The Vehicle Motion component executes the different maneuvers as well as autopilot, PM_ForwardBackwards, UM_BackwardForwards, PM_Forward and UM_Backward functions. These functions contains the algorithms and the actuator's commands needed to move the vehicle. The design of a motion controller is required to calculate the correct force needed in the actuator to avoid any collision or to move at a concrete speed.

The component function is to execute the movements indicated by the Drive Planning, by translating the actions into a throttle, steer, brake, etc. strengths messages. These messages depends on the final application, if it is tested over a simulator or over physical actuators.

4.3 ROS2 Design

Once architecture has been defined, the structure of the ROS2 software must be described. The implementation of ROS2 allows the execution of independent computing processes. These processes are known as nodes, which provide robustness against fault isolation, code reusability, simplicity, and faster development and modularity.

Communication between these nodes takes place by sending messages through topics, which must be structured according to the messages. When defining the nodes, the content of a message and the topic has to be described. The node publishes or subscribes these topics what means inserting a message or extracting a message. The publishing/-subscription model is intended to be modular in scale and is adequate for distributed systems.

This robotic environment embraces DDS as the communication protocol, nevertheless, as an outlier, internal process communication is run outside of DDS. For the time being, the design is focused on the node structure executing within ROS2 in Python 3.8. A software system in ROS2 is created as a workspace with a standardized package structure.

The workspace is composed by four folders: source (src), build, install and log, see Figure 4.8. The other folders are built for the correct function of the software design over the middleware. The source (src) folder includes packages as well as the carla-ros-bridge, the automated driving system and the vehicle functions.

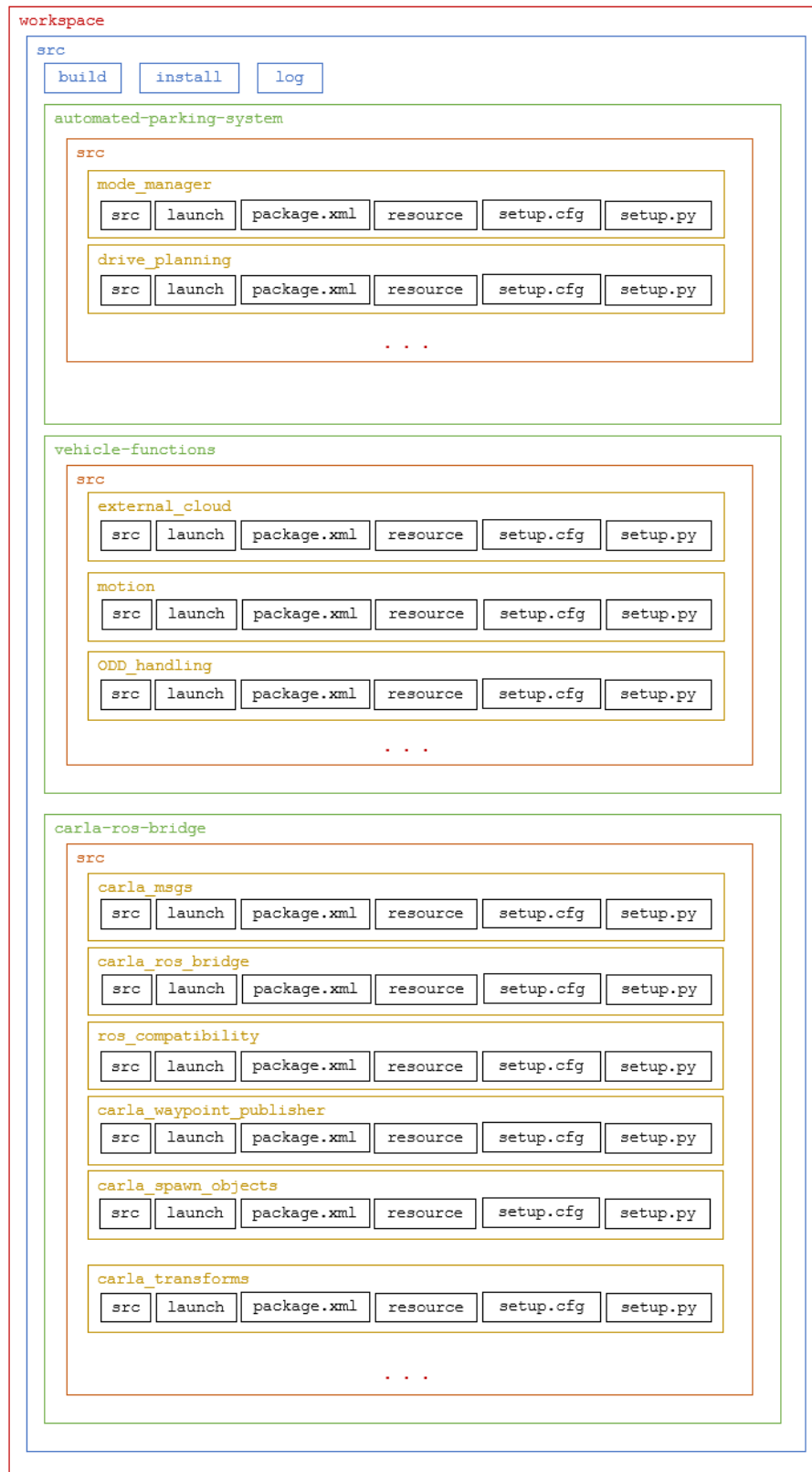


Figure 4.8: ROS2 Automated Parking System workspace structure.

4.3.1 CARLA-ROS-Bridge packages

The bridge has multiple factors to highlight, however, the most remarkable ones are the compatibility between CARLA Simulator and ROS2 middleware, the perfect communication between both worlds, and the achievement of a standardized structure of this robotic environment projects developed for this simulator, which allows the reuse and modulation of the developed projects. Of the many utility packages provided by this tool, only a few have been used for this project. Nonetheless, the fact of having more utilities just leaves the door open for future releases of this project with more functionalities.

These packages allow architectures to be launched on ROS2 environment over CARLA Simulator, visualizing the results on the CARLA Simulator or on RVIZ. Basically, it is common for programs running in this promising robotic environment to be launched with *.launch.py*, known as launch files. The final implementation section goes into detail on the structure of this launch and the configuration file.

4.3.2 CARLA World

The project implements CARLA classes to emulate the behavior wanted for the APS. The system uses the data from the GNSS sensor to know the odometry and the ground truth data proportioned by the simulator for the object detection.

The Global Navigation Satellite System (GNSS) sensor is considered an actor from the simulation world, the simulation receives a simulation of the GPS signal of the vehicle in real-time. The position displays the geographical reference given by the map, being the parameters as longitude, latitude, and altitude of the vehicle. The odometry topic has a processed information of this GNSS sensor, the reason why the system directly uses this topic to know the correct position of the car.

The obstacle detection is a function developed in this project to calculate the distance of the vehicle from a possible obstacle by using a norm algorithms. The simulator and carla-ros-bridge gives the possibility to use services which gives the position of the objects.

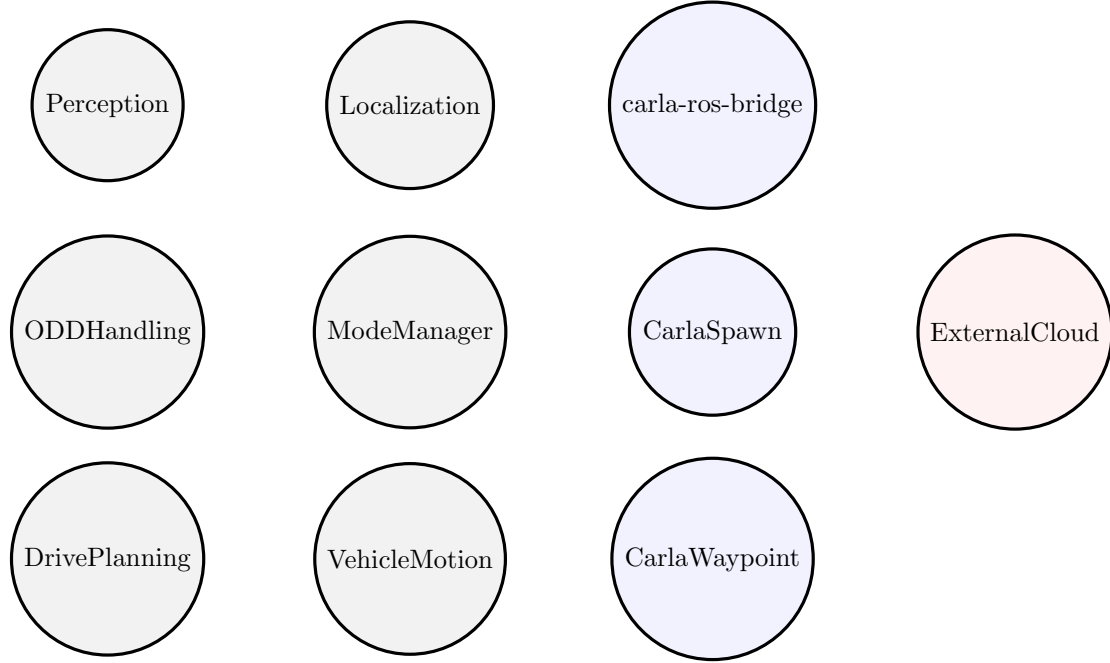
4.4 APS Deployment in ROS2

After presenting the top-level architecture, the software components and describe the functionality to be implemented, now it is time to start with the detailed design of the software in ROS2. The implementation includes the development of the nodes and topics to implement the APS and the requirements described in Section 4.1 and Section 4.3.

4.4.1 Nodes

It is best practice to create a ROS2 node for each component of the top-level architecture described in Figure 4.2 of Section 4.1. The nodes defined specifically for the Autonomous Parking System are launch after the carla-ros-bridge node is running over the ROS2 environment. The nodes created are shown in Figure 4.9 and their implementation are explained in Section 4.5.

Figure 4.9: ROS2 nodes. Three different function nodes: APS components (■), CARLA components (■) and External components (■).



Each node has a very specific function to execute. However, the execution cannot occur randomly, rather it has to remain strictly predetermined by the designers in order to fulfill the run-time requirements. A connection between two nodes determines a communication path. This communication takes place through the topics where these nodes are published or subscribed.

4.4.2 Topics and messages

The system consists of predefined topics on carla-ros-bridge to communicate automatically with CARLA and obtain feedback or perform actions on the simulator. Additional topics shall be created to communicate the nodes of the Automated Parking System. More information about topics and their respective messages are described in the Table C.1 of Appendix C. An example of this table is a topic and the message type shown in Table 4.12. It is a predefined topic from carla-ros-bridge which is responsible of the actuators located in the vehicle. The topic `vehicle_control_cmd` is published by the Vehicle Motion node.

Table 4.12: Topic `/carla/ego_vehicle/vehicle_control_cmd` example.

Topic	Message
<code>/carla/ego_vehicle/vehicle_control_cmd</code>	<code>CarlaEgoVehicleControl</code>

The topics contain a message structure, for example, there are message types as carla-msgs, std-msgs, nav-msgs, etc. Inside the types exist different messages with different formats. In this case, the message *CarlaEgoVehicleControl* is declared in carla-msgs types.

Messages contain the data formats required to get into a topic, if this format is not followed, errors might occur producing the consequent non-communication between nodes. The carla-msgs package contains some of the messages utilized for the topics available in carla-ros-bridge. The *CarlaEgoVehicleControl* contains all the parameters to apply into actuators control as shown in Table 4.13. A detailed list of the messages used in this project can be found in Appendix D.

Table 4.13: CarlaEgoVehicleControl message.

Variable	Units	Type	Description
throttle	-	Float32	Motive force
steer	-	Float32	Wheel deviation
brake	-	Float32	Brake force
hand_brake	-	Bool	Hand brake activation
reverse	-	Bool	Reverse activation
gear	-	Int32	Current gear of the vehicle
manual_gear_shift	-	Bool	Manual gear shift activation

When a node publishes the topic, all nodes subscribed to this topic receive the message and execute functions according to the received data. Using this publish/subscribe mechanism, it is possible to develop practically infinite and complex network systems. The implementation of the node functionality is described briefly in the upcoming section to have an idea of how to program code which can be ran in ROS2.

4.4.3 Node - Topic Architecture

The most critical and important step of the implementation is to define the low-level architecture. Perception, Location, Driving Planning, ODD Handling, Vehicle Motion and AS Mode Manager components have to be implemented as nodes, topics, and messages through the aforementioned bridge. The challenge is to run nodes over ROS2, which is not designed for automotive applications.

In the system, the topics can be classified according to what object they act on, basically, some of them simply act on general actors as weather and simulator state. Others are totally focused on the vehicle and others on receiving data from the /external-cloud such as traffic information, traffic lights, weather, parking spot and maneuver estimated.

The most relevant topics are described in the Figure 4.10 and Table 4.14. The Figure 4.10 shows three different function nodes: CARLA components (■), External components (■) and APS components (■). A complete architecture extracted from ROS2 through

rqt_graph is on Figure E.1 of Appendix E. Carla_ros_bridge publishes and subscribes all the information from ROS2 to CARLA and vice versa. Each node is represented with circle boxes whereas the topics are the directed edges of the graph.

Three nodes provided by CARLA are used in the APS. Carla Spawn generates the objects and vehicle into the simulation map. SetInitialPose contains the initial vehicle position when requests the APS service. CarlaWaypoint generates the waypoints for automated operation to be able to follow a path of points on the map. The VehicleMotion traces the movements following the instructions of the DrivePlanning. The Drive Planning node processes the waypoints coming from the CarlaWaypoint node and publishes the command needed by VehicleMotion to produce motion.

For the integration of the APS in CARLA, another node has been created for simulate the real External Cloud Service (ECS), by the name of ExternalCloud. Thus, creating a node that publishes the parking location selected by the driver.

Finally, two more nodes are created for the parking maneuver of the vehicle. The first one is the ODDHandling which implements the algorithms to perform the detection of hazards for the safety. The second node is the Localization which manages the surrounding of the car.

When the system is implemented and operating, apart from being able to visualize the simulation, the ROS2 middleware provides a graph of the complete state of the system and the interconnections. Fortunately, the ROS2 framework enables to visualize either through CARLA Simulator, through a PyGame window or through RVIZ, which is typically used by Robotic Operational System.

Figure 4.10: ROS2 architecture. Three different function nodes: APS components (■), CARLA components (■) and External components (■).

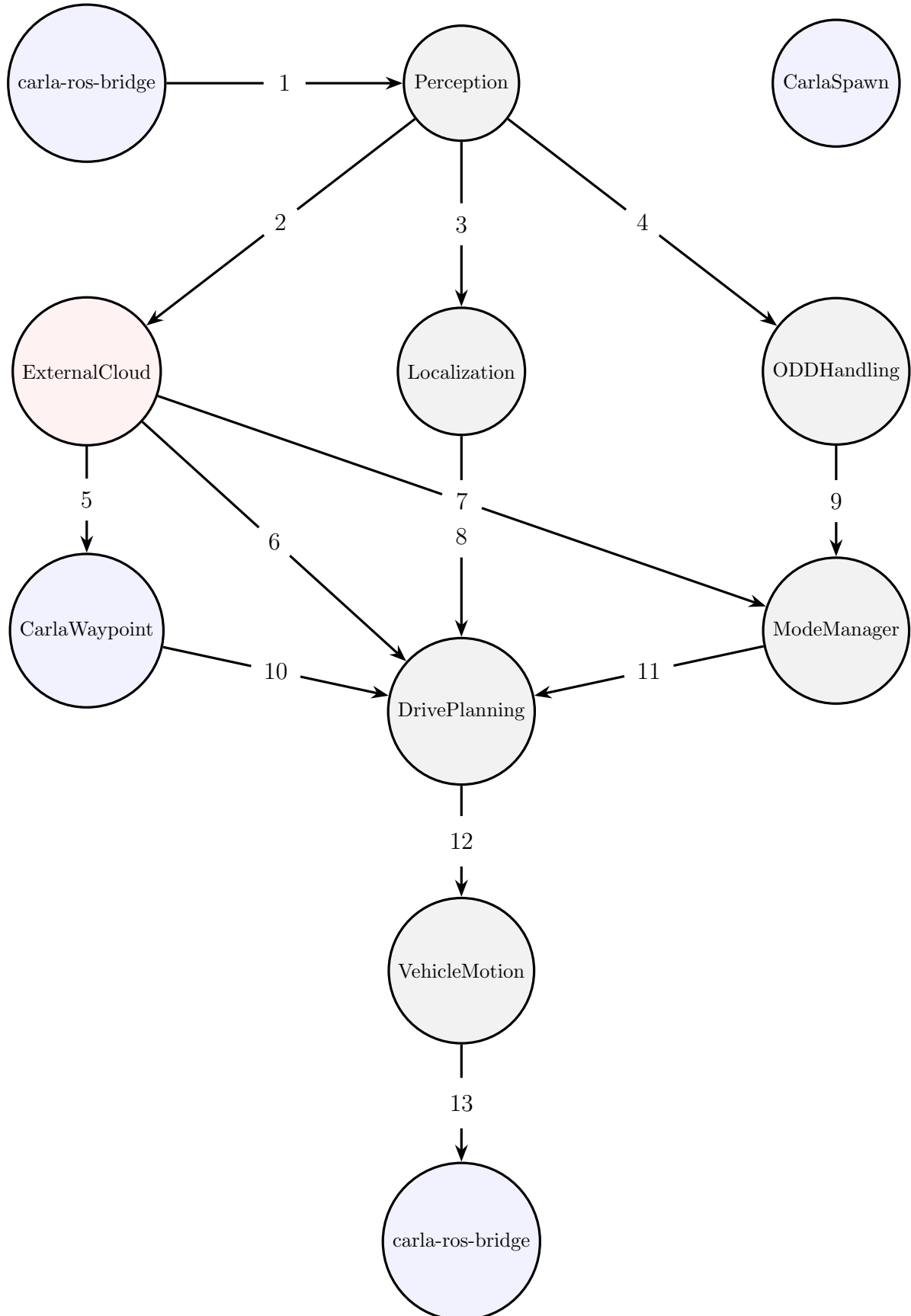


Table 4.14: Topics used in the communication.

Branch	Topics
1	/carla/ego_vehicle/control/set_transform /carla/ego_vehicle/odometry /carla/ego_vehicle/radar_front /carla/ego_vehicle/rgb_front/image /carla/ego_vehicle/semantic_lidar /carla/ego_vehicle/semantic_segmentation_front/image /carla/ego_vehicle/gnss /carla/ego_vehicle/imu /carla/ego_vehicle/lidar
2	/carla/weather_control
3	/carla/ego_vehicle/odometry /carla/ego_vehicle/radar_front /carla/ego_vehicle/rgb_front/image /carla/ego_vehicle/semantic_lidar /carla/ego_vehicle/semantic_segmentation_front/image /carla/ego_vehicle/gnss /carla/ego_vehicle/imu /carla/ego_vehicle/lidar
4	/carla/ego_vehicle/odometry /carla/ego_vehicle/objects /carla/traffic_lights/info /carla/traffic_lights/status /carla/weather_control
5	/initial_maneuver_goal_pose
6	/initial_parking_maneuver_goal_pose /final_parking_maneuver_goal_pose
7	/carla/ego_vehicle/APS_activation /carla/ego_vehicle/location_selection /carla/ego_vehicle/parking_selection
8	/carla/ego_vehicle/localization/info
9	/carla/ego_vehicle/out_of_ODD /carla/ego_vehicle/hazard_detection
10	/carla/ego_vehicle/waypoints
11	/carla/ego_vehicle/APS_done /carla/ego_vehicle/speed_command /carla/ego_vehicle/APS_state_mode /carla/ego_vehicle/APS_maneuver_mode /carla/ego_vehicle/APS_vehicle_mode
12	/carla/ego_vehicle/motion/info
13	/carla/ego_vehicle/vehicle_control_cmd

4.5 Implementation

The implementation of most of the items of this project has already been described previously. Nevertheless, how to generate the node and make it publish or subscribe hasn't been stated yet. For this reason, this section provides a demonstration of construction of a node class and how publications and subscriptions are defined in the initialization definition. The main function generates the node through Threads with an specific function during a time lap and next the system spins this node.

Additionally, it is explained how the system manages path planning through waypoints. In the parking, it is necessary to generate waypoints for planning the trajectory to the parking slot. The location is given by an external node to the system, known as the External Cloud (ECS). This section ends up with the description of the algorithms describing the autonomous vehicle functionality.

4.5.1 Waypoints

The waypoint node generates a route between the position of the vehicle and the selected destination. Multiple pieces of information are provided by the simulator, such as the lines of the road and the direction of each street, the node generates points to trace a trajectory towards the target. The map allows to generate waypoints only in the road ways.

An algorithm provided by the CARLA's developers provides transformation functions from Carla.Transform to Ros.Transform. This occurs since the bridge and the simulator communicate with Carla.Transform. However, between the nodes and the bridge, the communication must consist on messages supported in ROS2, therefore, the communication must be performed on Ros.Transform. These messages contain position and orientation and are the basis of the waypoints.

```
#CARLA Transform
Transform( Location(x,y,z) , Rotation( pitch , yaw , roll ) )

#ROS2 Transform
geometry_msgs.msg.Pose( position=geometry_msgs.msg.Point(x,y,z) , orientation=
    geometry_msgs.msg.Quaternion(x,y,z,w) )
```

Otherwise, to park in the outdoor parking requires different implementation due to waypoints cannot be located inside the outdoor parking. For this reason, the route chart within the outdoor parking, which was already implemented supported in [2], has been re-adapted to the robotic environment.

4.5.2 External Cloud

The External Cloud node emulates a real ECS which verifies if the maneuver selected by the driver is possible. Then, the component sends three essential signals to AS Mode Manager and the parking spot position to the Drive Planning. The description is placed in Figure 4.15.

Table 4.15: External Cloud signals.

Topic	Type	Description
APS_activation	Boolean	True if the client activates the APS
parking_selection	Int8	Different integer for park, unpark and stop
location_selection	Int8	Different integer for parallel and perpendicular
parking_maneuver_goal_pose	Pose	3D vector containing the parking spot location

4.5.3 Vehicle functions

The autopilot function is considered the first phase of the autonomous driving and indicated as green line. The autopilot involves the usage of waypoints combined with controller and misc classes developed by CARLA developers. The autopilot executes the appropriate movement on the actuators to reach the next waypoint at the set speed. The visualization of the autopilot phase (green + white arrows) is available in Figures 4.11 and 4.12. The white arrows indicate forward motion while the black ones describe reverse motion.

The second phase in the parallel parking includes a backward maneuver (orange + black arrows) and in perpendicular parking includes forward driving (orange + white arrows) through the parking zone. The third and last phase (red + white arrows) consists of a short forward movement in the parallel parking and forward curved movement, using an open-source quintic polynomial algorithm [51], in the perpendicular parking.

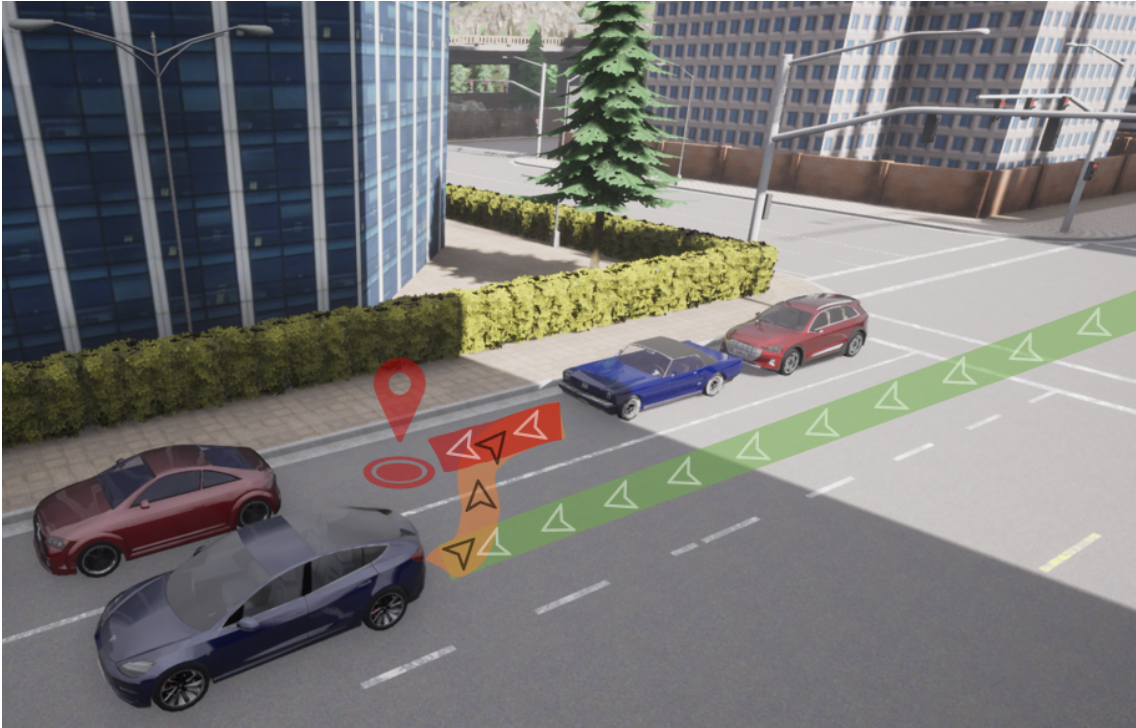


Figure 4.11: Parallel parking maneuver waypoints.



Figure 4.12: Outdoor parking maneuver waypoints.

4.5.4 Parking maneuver selection

Once the AS Mode Manager guarantees the parking execution ensuring safety, the Drive Planning component processes the data received using the following Algorithm 1.

Algorithm 1 Drive Planning Algorithm to select the parking maneuver

```

1: procedure SELECT_MANEUVER( $M\_Sta, M\_Veh, M\_Man$ )
2:   if  $M\_Sta == aps\_on$  then
3:     if  $M\_Veh == veh\_parking + M\_Man == parallel\_parking$  then
4:        $maneuver\_setected \leftarrow PM\_ForwardBackwards$ 
5:     else if  $M\_Veh == veh\_parking + M\_Man == perpendicular\_parking$  then
6:        $maneuver\_setected \leftarrow PM\_Forwards$ 
7:     else if  $M\_Veh == veh\_unparking + M\_Man == parallel\_parking$  then
8:        $maneuver\_setected \leftarrow UM\_BackwardForwards$ 
9:     else if  $M\_Veh == veh\_parking + M\_Man == parallel\_parking$  then
10:       $maneuver\_setected \leftarrow UM\_Backward$ 
11:   else
12:      $maneuver\_setected \leftarrow None$ 
13:   else if  $M\_Sta == aps\_safemode$  then
14:      $maneuver\_setected \leftarrow M\_Safe$ 
15:   else
16:      $maneuver\_setected \leftarrow None$ 
17:   return  $maneuver\_setected$ 

```

▷ The output is the maneuver function

To appreciate a more technical description about how to develop node architectures based in carla-ros-bridge usage, some important commands are listed in the next lines. For example, the concrete way to initialize a publisher or a subscriber is described into the `__init__`. Then, in the `run_step` function the publishers are sending messages to the topics following the next command which allows to publish the message passed as a parameter. Finally, in the main function, the node is executed through the initialization and the spinning.

```
# Mode Manager Node Code
import copy
import sys
import time
import threading

import ros_compatibility as roscmp
from ros_compatibility.exceptions import *
from ros_compatibility.qos import QoSProfile, DurabilityPolicy

import mode_manager.settings as s
from mode_manager.agent import Agent, AgentState

import carla_common.transforms as trans

from carla_msgs.msg import CarlaTrafficLightStatusList
from derived_object_msgs.msg import ObjectArray
from nav_msgs.msg import Odometry
from std_msgs.msg import Float64, Bool, Int8
from statemachine import StateMachine, State

class APS_state_machine(StateMachine):
    ...
    off = State('STATE_OFF', initial=True)
    on = State('STATE_PARKING_MANEUVER')
    emergency = State('STATE_EMERGENCY')
    ...

class ModeManagerAgent(Agent):
    def __init__(self):
        ...
        # Publishers
        APS_state_mode_publisher = self.new_publisher(Int8, "/carla/
            ego_vehicle/APS_state_mode", qos_profile = 10)
        ...
        # Subscribers
        hazard_detection_subscriber = self.new_subscription(Bool, "/carla/
            ego_vehicle/hazard_detection", hazard_detection, qos_profile =
            10)
        ...
        # Other algorithmic functions...
        def run_step(self):
            # Run steps declared in the node loop
            ...
            # Example of how to publish
            APS_state_mode_publisher.publish(new_state[1])
```

```

...
def main(args=None):
    # Node initialization and launching
    roscmp.init("mode_manager", args=args)
    controller = None
    try:
        executor = roscmp.executors.MultiThreadedExecutor()
        controller = ModeManagerAgent()
        executor.add_node(controller)
        roscmp.on_shutdown(controller.emergency_stop)
        update_timer = controller.new_timer(0.05, lambda timer_event=None:
            controller.run_step())
        controller.spin()

    except (ROSInterruptException, ROSException) as e:
        if roscmp.ok():
            roscmp.logwarn("ROS_error_during_exection:{ {}".format(e))
    except KeyboardInterrupt:
        roscmp.loginfo("User_requested_shut_down.")
    finally:
        roscmp.shutdown()

if __name__ == "__main__":
    main()

```

The last point to be highlighted is known as the launch file. These files are essential for launching any minimal sophisticated ROS2 project. The next code explains the methodology to wake up a node.

```

import os
import sys
import launch
import launch_ros.actions

def generate_launch_description():
    ld = launch.LaunchDescription([
        launch.actions.DeclareLaunchArgument(
            name='role_name',
            default_value='ego_vehicle'
        ),
        launch_ros.actions.Node(
            package='mode_manager',
            executable='mode_manager',
            name=['mode_manager_', launch.substitutions.LaunchConfiguration(
                'role_name')],
            output='screen',
            emulate_tty=True,
            parameters=[
                {
                    'role_name': launch.substitutions.LaunchConfiguration(
                        'role_name')
                }
            ]
        )
    ])

```

```
    ])  
    return ld  
  
if __name__ == '__main__':  
    generate_launch_description()
```

In a complex architecture, the recommendation is not to launch nodes one by one because some information could be lost if some node wakes up and publishing while the subscribers aren't still operative. That is the reason why a customized launch file is implemented in this project for synchronizing all the nodes launch files at the same time.

This technical overview of the ROS2 system implementation concludes the development section. Subsequently, in the results chapter, the analysis of the architecture and a comparison with the AUTOSAR Classic Platform is made.

Chapter 5

Results

In this chapter, the results of the ROS2 architecture are presented, described and evaluated. The architecture designed in ROS2 is compared with the architecture designed in AUTOSAR Classic [1]. The comparison provides knowledge about how to migrate from one to another architecture avoiding rewriting the entire system in future works.

5.1 APS integration in ROS2

The interest of the automotive industry to use ROS2 has motivated the development of a communication bridge for ROS2 nodes to interact with CARLA. A careful analysis of the functionality implemented in the carla-ros-bridge was done before start of development.

The communication among nodes is extremely fast and efficient, enabling the realization of sophisticated architectural structures. Thus providing a quick and a safe response for the safety of the vehicle. The operation of the system, when running, is shown clearly in the following flowchart in Figure 5.1, fulfilling the expectations of having an autonomous system in ROS2 architecture which ensures safety.

An easy method to understand the ultimate functionality of the system in various situations is through the use of sequential diagrams. In Figure 5.2, the meteorological conditions become severe exceeding the ODD maximum levels in cloudiness and precipitation. When these parameters are exceeded, the AS Mode Manager switches to M_Safe mode, thereby preserving safety of the vehicle and the surrounding. The Figure F.1, Appendix F, illustrates the operation of the parking system when the APS has been activated by the customer and the parking maneuver is safely executed.

The workflow of the safe parking system is depicted in Figure 5.1. The workflow is initiated when the vehicle is engaged. The nodes are automatically initialized and launched. The first one to be launched is carla-ros-bridge and afterward the rest. The system enters the OFF state if no issues occur. When the Automated Parking System is switched on by the driver and no danger is detected, the system enters the ON or PARKING_MANEUVER state. The ON state develops the maneuvers and the system detects any hazard or the exit of the ODD, the flow switches to the SAFE or EMERGENCY state.

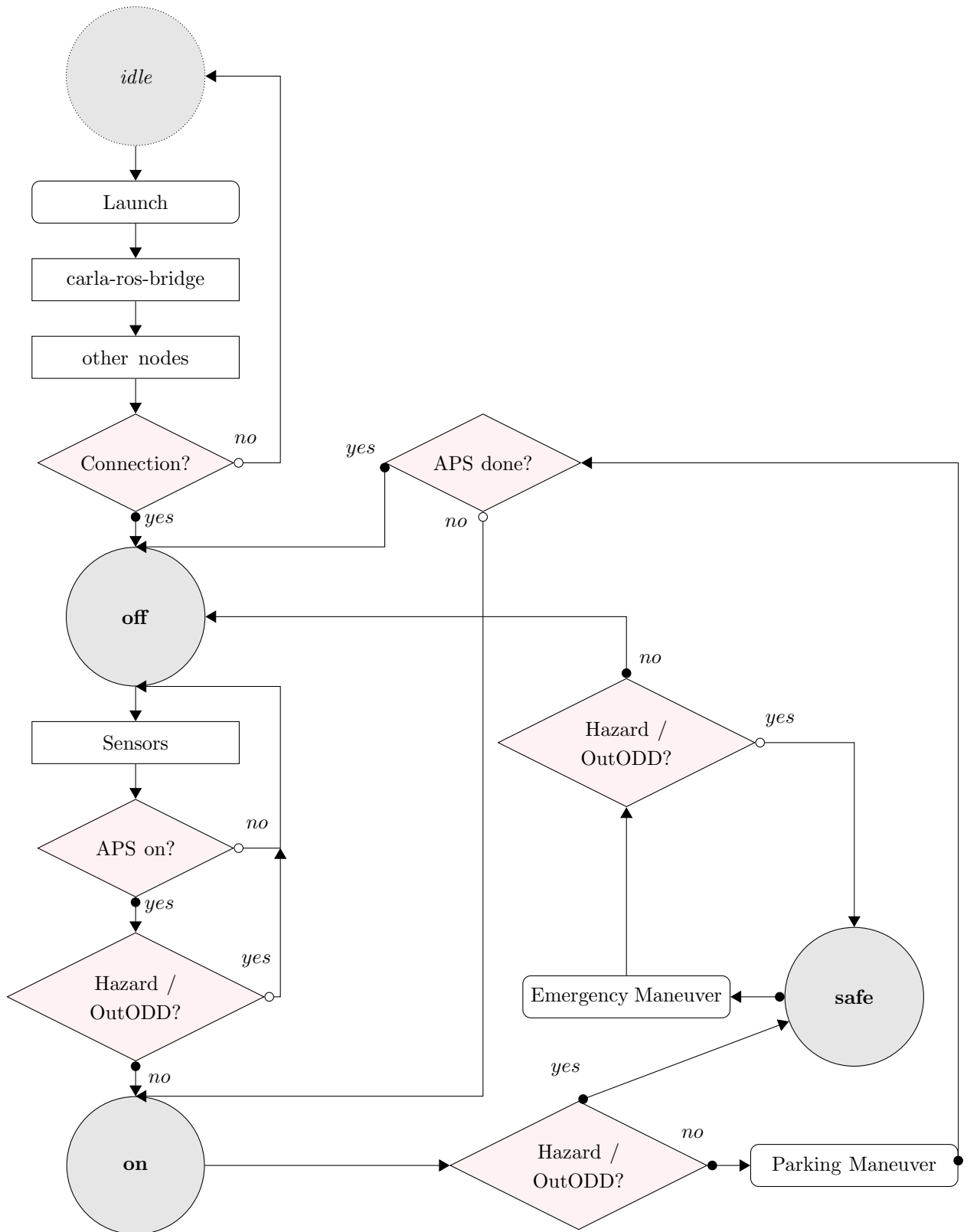


Figure 5.1: ROS2 architecture flowchart.

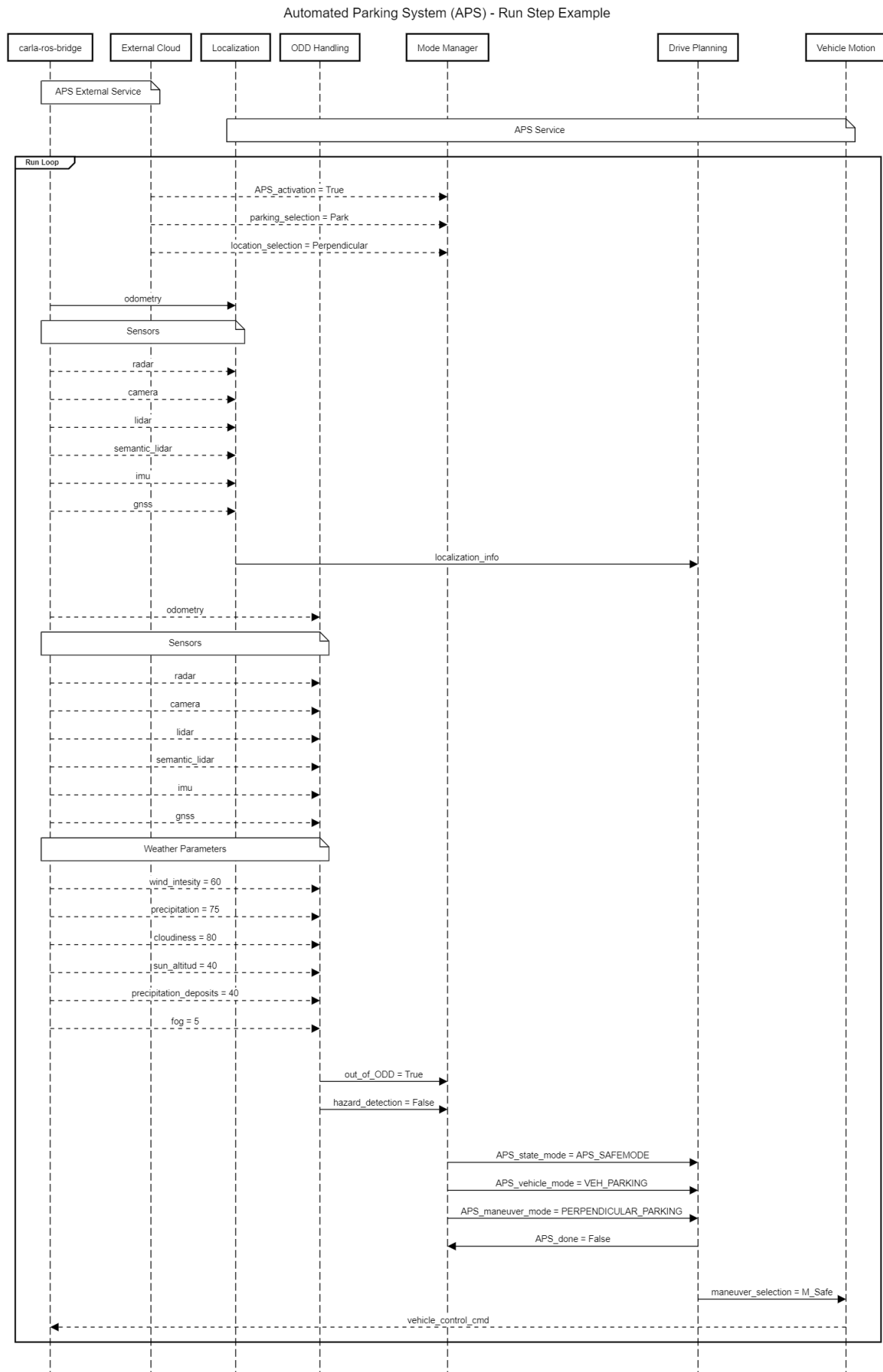


Figure 5.2: Sequence diagram example PM_Forward out of ODD situation.

Table 5.1: APS most relevant signals.

Topic	Type name	Type parameter	Description
APS_vehicle_mode	VEH_DRIVING	$Int8 = 0$	When vehicle is driving
	VEH_PARKING	$Int8 = 1$	When vehicle is parking
	VEH_UNPARKING	$Int8 = 2$	When vehicle is unparking
	VEH_STOPPED	$Int8 = 3$	When vehicle is stopped
APS_state_mode	APS_OFF	$Int8 = 0$	APS deactivated
	APS_ON	$Int8 = 1$	APS activated
	APS_SAFEMODE	$Int8 = 2$	APS safe mode activated
APS_manuever_mode	PARALLEL	$Int8 = 0$	Parallel parking
	PERPENDICULAR	$Int8 = 1$	Perpendicular parking
hazard_detection	HAZARD	$Bool = True$	Hazard detection
	NO_HAZARD	$Bool = False$	Non hazard detection
out_of_ODD	INSIDE_ODD	$Bool = True$	Inside the ODD
	OUTSIDE_ODD	$Bool = False$	Outside the ODD
APS_done	APS_COMPLETE	$Bool = True$	Parking maneuver finished
	APS_UNCOMPLETE	$Bool = False$	Parking maneuver undone

The system is able to detect, pedestrians, traffic lights, vehicles and other object as shown in Figure 5.3. When the Localization detects an obstacle, the Drive Planning stops and triggers the M_Safe mode.



Figure 5.3: Obstacle detection example.

5.2 ROS2 versus AUTOSAR

AUTOSAR Classic Platform has standardized the mode manager methodology. This methodology enables the system to describe switch mechanisms as part of the software component description in a standardized way. The communication is implemented by automated generation of the RTE (Run Time Environment).

The present dissertation has only analyzed the the AUTOSAR Classic Platform. Like ROS2, the AUTOSAR Adaptive Platform is a service oriented architecture and it is out of this master thesis scope.

ROS2 is a potential solution to compete head-to-head with AUTOSAR in the automotive domain. Table 5.2 provides a first comparison of the capabilities of ROS2 with respect to AUTOSAR Classic based on the development work of this master thesis versus [1].

Table 5.2: ROS2 vs AUTOSAR Classic

Category	ROS2	AUTOSAR Classic
Programming code	Python, C++ and Java code	C code
Platform support	micro-Processor	micro-Controller
OS support	Linux, Windows and macOS	OSEK
Embedded system support	Newcomer and commercial	Experienced and commercial
Network transport	DDS	TCP/UDP
Components	Nodes	Software Components (SWC)
Industry usage	Robotic and other prototyping	Automotive
Extendability	Modular (SOA)	Re-code/Re-config required
Scalability	High	Medium
Performance	Medium	High
Security support	High	High
Reliability	Medium	High
Real-time computing	Soft real-time	High real-time

The ROS2 architecture fulfills the requirements of a self-driving system. Some ROS2 system results exceed the expectations compared with AUTOSAR Classic. The comparison is based on the most crucial aspects required for automated system architectures, as shown in [34], and it is depicted in the spider diagram of Figure 5.4.

AUTOSAR is a widely established standard and most automotive stakeholders are committed to the standard. However, there are almost no open-source AUTOSAR solutions. On the contrary, ROS2 is an open source platform with a big number of contributors and tools. The ROS2 community reacts very quickly to the development needs and has a promising future in automotive. Since ROS2 is a service-oriented architecture, it is better

suited to adapt to changing conditions. The AUTOSAR Classic Platform has a lower adaptability. ROS2 also scores better on reusable and scalability. However, real-time and reliability are still better in AUTOSAR.

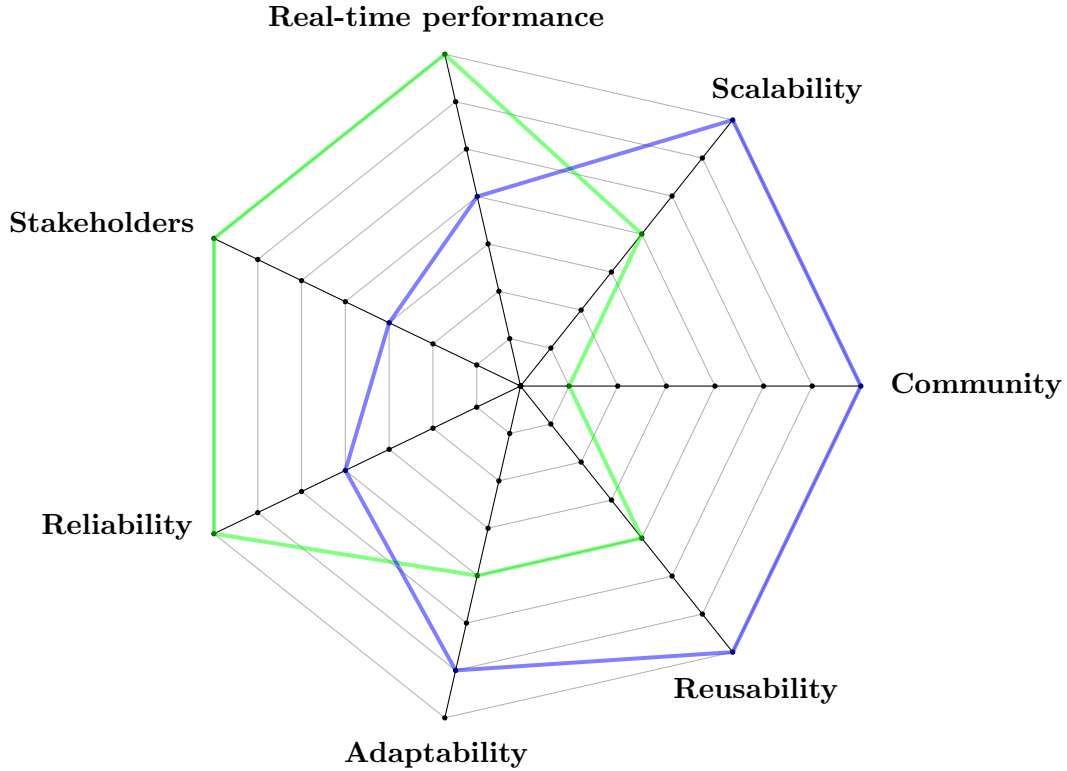


Figure 5.4: Spider diagram of AUTOSAR (■) and ROS2 (■) architectures.

Since ROS2 is becoming increasingly attractive and profitable for companies because of reusability, scalability and open community, many automotive stakeholders are currently investing in the research and innovation to deploy the ROS2 middleware in series production software. In this way, the weakness of the robotic architecture in communication with stakeholders promises to reach the same level as AUTOSAR in the coming years. More difficult will be to improve in real-time performance and reliability.

Chapter 6

Conclusions and Future Development

The increasing complexity of autonomous driving systems and the necessity to improve performance require more tools and development. The robotic middleware, ROS2, provides a wide range of open-source tools with an open community of developers. At the same time, it improves the performance of much more aspects of established automotive software architectures.

Vehicle manufacturers are taking advantage of ROS2 for building prototypical autonomous functions. The robotic middleware provides more versatile and adaptive functions. This master thesis has evaluated the use of the ROS2 framework with a focus on how a good architecture can contribute to ensuring safety. The engineered software developed in ROS2 is simply scalable and additional autonomous functions could be added by customizing the system's requirements. The ROS2 system is built on a modular structure which is extendable to new nodes which publishes and subscribes new topics and services.

This master thesis has shown that it is possible to develop an architecture for an automated driving system in ROS2. The generic architecture is not only applicable for autonomous vehicles, but also for other systems like robots, drones, trains, etc. The Automated Parking System developed in ROS2 has delivered impressive results in scalability and reusability, exceeding the capabilities of the architecture previously developed in the AUTOSAR Classic Platform. To analyze this well-established technology in the field of robotics for the automotive domain, it is compared with an AUTOSAR Classic architecture. The Automated Parking function has been integrated and validated in the CARLA simulator.

In any case, ROS2 and the AUTOSAR Adaptive Platform are service-oriented architectures. Interoperability can be achieved by using a common communication protocol, i.e. DDS, allowing the realization of sophisticated and scalable structures as well as great reusability and modularity of the code. The AUTOSAR Adaptive Platform has not been in the scope of this work but it connects with the next topic, the future work.

The following improvements and enhancements have been the identified future works:

- Evaluate standardized AUTOSAR Adaptive Platform architecture and the interoperability with ROS2 nodes.
- Expand the deployment of ROS2 bridge to simulate and incorporate more complex simulations in the customized map.
- Implement advanced algorithms for the detection of obstacles and weather conditions through the sensors available in the vehicle.
- Analyse the compatibility of the current approach to other simulated environments by reusing the implemented code.

Budget

The budget contains equipment and personal costs as shown in Table 6.1 and Table 6.2. Additionally, there is a devaluation on the equipment materials.

Table 6.1: Equipment costs.

Item	Unit Cost (€)	Devaluation	Total cost (€)
Computer	700	0.4	280
CARLA Simulator	0	0.4	0
ROS2	0	0.4	0
Working Station	4000	0.4	1600
Microsoft Office	60	0.2	12
TOTAL			1892

Table 6.2: Personal costs.

Position	Unit Cost (€/h)	Time (h)	Total cost (€)
Researcher	20	600	12000
Supervisor	25	40	1000
TOTAL			13000

The final costs of the project is the sum of them, which gives a final value of 14.892€.

Environmental Impact

Today mobility is one of the main drivers in the economy of developed countries. Mobility is a basic commodity in our life but this mobility has had a big impact on the environment and its effects on the climate change. The energy cost of the computational demands of the development of the system represents one of the most significant negative aspects of this project.

Nonetheless, the implementation of this dissertation provides a benefit to society and to safety by reducing the accidents rates, since most accidents occur due to human error. This reduction of accidents translates into less vehicle's production and damage caused by accidents.

Another factor to take into account is the reduction of emissions through the automated parking system. The time spent looking for parking in large metropolitan areas should be reduced. In these regions, parking search often involves much time of emissions each time the vehicle is picked up. These positive points of view, together with the decrease in fuel costs for consumers, contribute to the reduction of pollution emissions.

Bibliography

- [1] Y. El Kabdani. *Software architecture design for safety in automated driving systems*. UPC, Barcelona, 2022.
- [2] R. Tremosa. *Software architectural design for safety in automated parking system*. UPC, Barcelona, 2022.
- [3] Garrett Shea. 2.0 Fundamentals of Systems Engineering, February 2019.
- [4] Javier. Fernandez de Canete. *System Engineering and Automation An Interactive Educational Approach*. Studies in fuzziness and soft computing ; 268. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 2011. edition, 2011.
- [5] Wei-Tek Tsai. Service-oriented system engineering: a new paradigm. In *IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, pages 3–6. IEEE, 2005.
- [6] Alan N Steinberg. Data fusion system engineering. In *Proceedings of the Third International Conference on Information Fusion*, volume 1, pages MOD5–3. IEEE, 2000.
- [7] Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4):40–52, 1992.
- [8] Bob Leigh and Reiner Duwe. Software architecture of autonomous vehicles. *ATZ-electronics worldwide*, 14(9):48–51, 2019.
- [9] Harry H. Dreany and Robert Roncace. A cognitive architecture safety design for safety critical systems. *Reliability engineering system safety*, 191:106555–, 2019.
- [10] *Software architecture : second international conference, ecsa 2008 paphos, cyprus, september 29-october 1, 2008 proceedings*. Programming and Software Engineering ; 5292. Springer, Berlin, Germany, 1st ed. 2008. edition, 2008. Backup Publisher: ECSA 2008 Corporate Author Publication Title: Software architecture : second international conference, ecsa 2008 paphos, cyprus, september 29-october 1, 2008 proceedings.
- [11] David Garlan. Software architecture: a roadmap. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, pages 91–101, Limerick, Ireland, 2000. ACM Press.
- [12] *Advanced Microsystems for Automotive Applications 2012 Smart Systems for Safe, Sustainable and Networked Vehicles*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 2012. edition, 2012.
- [13] Alessandro Frigerio, Bart Vermeulen, and Kees G. W Goossens. Automotive architecture topologies: Analysis for safety-critical autonomous vehicle applications. *IEEE access*, 9:62837–62846, 2021.
- [14] Sangeeth Kochanthara, Niels Rood, Arash Khabbaz Saberi, Loek Cleophas, Yanja Dajsuren, and Mark van den Brand. A functional safety assessment method for cooperative automotive architecture. *The Journal of systems and software*, 179, 2021.

- [15] Kenneth Lind and Rogardt Høldal. Automotive system development using reference architectures. In *2012 35th Annual IEEE Software Engineering Workshop*, pages 42–51. IEEE, 2012.
- [16] D. Chen, R. Johansson, H. Lönn, H. Blom, M. Walker, Y. Papadopoulos, S. Torchiaro, F. Tagliabo, and A. Sandberg. Integrated safety and architecture modeling for automotive embedded systems. *Elektrotechnik und Informationstechnik*, 128(6):196–202, 2011.
- [17] AUTOSAR development cooperation. Autosar. Accessed: 2022-11-07 [Online] Available: <https://www.autosar.org/>.
- [18] Ahmed Hamed, M. Watheq El-Kharashi, Ashraf Salem, and Mona Safar. Two-layer bus-independent instruction set architecture for securing long protocol data units in automotive open system architecture-based automotive electronic control units. *Electronics (Basel)*, 11(6):952–, 2022.
- [19] Matthias Traub, Alexander Maier, and Kai L. Barbehon. Future automotive architecture and the impact of it trends. *IEEE software*, 34(3):27–32, 2017.
- [20] Simon Fürst and Markus Bechter. Autosar for connected and autonomous vehicles: The autosar adaptive platform. In *2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 215–217. IEEE, 2016.
- [21] M. Wood C. Knobel D. Wittman and Y. Wang. Safety first for automated driving.
- [22] AUTOSAR development cooperation. Classic Platform Accessed:2022-11-07 [Online]. Available:<https://www.autosar.org/standards/classic-platform/>.
- [23] AUTOSAR development cooperation. Adaptive Platform. Accessed:2022-11-07 [Online]. Available:<https://www.autosar.org/standards/adaptive-platform/>.
- [24] Tully Foote Steven Macenski and Bryan Gerkey. Robot Operating System 2: Design, architecture, and uses in the wild. (English). 7(66), 2022.
- [25] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, May 2022.
- [26] Weng Zeyu, Yu Jinsong, and Sheng Wubin. Distributed test system based on publish/subscribe middleware. In *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, pages 10–15. IEEE, 2017.
- [27] Endre Erős, Martin Dahl, Kristofer Bengtsson, Atieh Hanna, and Petter Falkman. A ROS2 based communication architecture for control in collaborative and intelligent automation systems. *Procedia Manufacturing*, 38:349–357, 2019.
- [28] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

- [29] Michael Reke, Daniel Peter, Joschua Schulte-Tigges, Stefan Schiffer, Alexander Ferrein, Thomas Walter, and Dominik Matheis. A Self-Driving Car Architecture in ROS2. In *2020 International SAUPEC/RobMech/PRASA Conference*, pages 1–6, Cape Town, South Africa, January 2020. IEEE.
- [30] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ROS2. In *Proceedings of the 13th International Conference on Embedded Software*, pages 1–10, Pittsburgh Pennsylvania, October 2016. ACM.
- [31] Endre Eros, Martin Dahl, Atieh Hanna, Anton Albo, Petter Falkman, and Kristofer Bengtsson. Integrated virtual commissioning of a ROS2-based collaborative and intelligent automation system. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 407–413, Zaragoza, Spain, September 2019. IEEE.
- [32] Data Distribution Service (DDS) | Object Management Group. Accessed:2022-11-22 [Online]. Available:<https://www.omg.org/omg-dds-portal/>.
- [33] Erwin Lejeune, Audrey Queudet, Sampreet Sarkar, and Vincent Lebastard. Real-time Jitter Measurements under ROS2: the Inverted Pendulum case. page 4.
- [34] Michael Reke, Daniel Peter, Joschua Schulte-Tigges, Stefan Schiffer, Alexander Ferrein, Thomas Walter, and Dominik Matheis. A self-driving car architecture in ros2. In *2020 International SAUPEC/RobMech/PRASA Conference*, pages 1–6. IEEE, 2020.
- [35] Hang Cui, Jiaming Zhang, and William R Norris. An enhanced safe and reliable autonomous driving platform using ros2. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 290–295. IEEE, 2020.
- [36] ROS 2 Documentation — ROS 2 Documentation: Foxy documentation. Accessed:2022-11-07 [Online]. Available: <http://docs.ros.org.ros.informatik.uni-freiburg.de/en/foxy/index.html>.
- [37] Concepts — ROS 2 Documentation: Rolling documentation. Accessed:2022-11-07 [Online]. Available: <https://docs.ros.org/en/rolling/Concepts.html>.
- [38] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *Proceedings of the 13th International Conference on Embedded Software*, pages 1–10, 2016.
- [39] ISO 26262-1:2018. Accessed:2022-11-15 [Online]. Available: <https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/83/68383.html>.
- [40] ISO/TR 4804:2020. Date:2022-11-15 [Online] Available: <https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/08/03/80363.html>.
- [41] SAE J3016 automated-driving graphic. Accessed: 2020-05-15 [Online]. Available: <https://www.sae.org/site/news/2019/01/sae-updates-j3016-automated-driving-graphic>.
- [42] Dr Siddartha Khastgir. ASAM OpenODD Concept Release. [online]. Available: <https://www.asam.net/standards/detail/openodd/>.

- [43] Krzysztof Czarnecki. Operational design domain for automated driving systems. *Taxonomy of Basic Terms “, Waterloo Intelligent Systems Engineering (WISE) Lab, University of Waterloo, Canada*, 2018.
- [44] Chung Won Lee, Nasif Nayeer, Danson Evan Garcia, Ankur Agrawal, and Bingbing Liu. Identifying the operational design domain for an automated driving system through assessed risk. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1317–1322. IEEE, 2020.
- [45] CARLA Simulator. Accessed:2022-11-07 [Online]. Available: <https://carla.org/>.
- [46] Python API - CARLA Simulator. Accessed:2022-11-07 [Online]. Available: https://carla.readthedocs.io/en/latest/python_api/carla.SensorData.
- [47] The ROS bridge package - CARLA Simulator. Accessed:2022-11-07 [Online]. Available: https://carla.readthedocs.io/projects/ros-bridge/en/latest/run_ros/.
- [48] Gustavo Velasco-Hernandez, John Barry, Joseph Walsh, et al. Autonomous driving architectures, perception and data fusion: A review. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 315–321. IEEE, 2020.
- [49] Jorge Vargas, Suleiman Alsweiss, Onur Toker, Rahul Razdan, and Joshua Santos. An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions. *Sensors*, 21(16), 2021.
- [50] Sensors reference - CARLA Simulator. Accessed:2022-11-08 [Online]. Available: https://carla.readthedocs.io/en/latest/ref_sensors.
- [51] Atsushi Sakai. PythonRobotics, November 2022. original-date: 2016-03-21T09:34:43Z.

Appendices

Appendix A

System uses cases and constraints

Table A.1 shows the use cases of the automated parking system.

Table A.1: Use Cases of the system.

Use Case	Definition
UC-1 [1]	APS manages the driving maneuver of the vehicle instead of the driver.
UC-2	APS parking maneuver: Parallel and Perpendicular.
UC-3 [1]	APS available driving zones: Indoor parking, Outdoor parking, Urban Road, and Interurban Road.
UC-4 [1]	APS controlled by Smart Phone App.
UC-5a [1]	APS App shows the available parking space and books the parking space selected by the user, activates APS parking when the client selects the option “Park”, activates APS unparking when selecting “Unpark”, allows to stop the APS function when safety cannot be guaranteed selecting “Safe Mode”, and notifies if the result of the parking maneuver.
UC-5b [1]	APS App and External Cloud Service (ECS) are transferring information allowing them to book parking slots and providing their location.
UC-6 [1]	APS App shows the status with “Active” or “Inactive”, the selectable parking slots, and notifications about the APS status with “Parked”, “Safe Mode” or “Stop”.
UC-7 [1]	APS App offers a ”Park” button to start the vehicle parking function and select the desired parking spot and start the maneuver.
UC-8 [1]	The client selects the parking spot and accepts the parking spot to start the parking maneuver of the vehicle. The APS function will not proceed if the user does not select any spot or rejects the selection.
UC-9 [1]	If there is no parking space, APS functionality is not going to start.
UC-10 [1]	The APS App ”Unpark” option allows the user to define where to pick up the vehicle.
UC-11	ECS gives the position of the parking spot to the vehicle.
UC-12 [1]	APS parks in the parking spot provided by the APS App.
UC-13 [1]	APS considers that there are no users inside the vehicle and it is unresponsive to verify it.
UC-14	If APS detects an obstacle, it will not finish its maneuver until the obstacle leaves the space.
UC-15 [1]	APS states in “Safe Mode” if during the parking maneuver happens any collision possibility. or the user stops the maneuver by APS App.
UC-16	When the vehicle is parked, APS functionality finishes returning to initial state.
UC-17 [1]	APS locks the vehicle whenever the system is running and after its functionality.

To simplify the safety requirements, the APS assumed the constraints in Table A.2.

Table A.2: Constrains of the system.

Constrains	Definition
CON-1 [1]	Misuse of the APS function is not considered.
CON-2 [1]	Technical failures of the vehicle, sensors, and actuators are not to be considered during the parking maneuver.
CON-3 [1]	The vehicle can fulfill the parking maneuver and return to the initial location. Verification of fuel or electric power.
CON-4	Indoor and outdoor roads only provide parallel parking spots.
CON-5	Indoor and outdoor parking only provide perpendicular parking spots.
CON-6 [1]	Connection failures between the vehicle and the APS App are not considered.
CON-7 [1]	ECS is always available.
CON-8 [1]	ECS reliably provides parking spaces.
CON-9 [1]	When a parking spot is booked, it cannot be offered to other vehicles.
CON-10 [1]	The client has verified that the parking spots respect the minimum measures to park the vehicle.
CON-11 [1]	The client must be accessible during APS operation until receiving the notification.
CON-12 [1]	The drivable zones must have traffic lane types for identifying the parking area spot. The APS must recognize the scenario.
CON-13 [1]	APS respects traffic and driving rules.
CON-14 [1]	APS drives to a safe zone if the weather conditions are outside of the ODD defined.

Appendix B

CARLA Simulator

CARLA is open-source driving simulator based on scalable client-server architecture. It provides open digital assets (urban layouts, buildings, vehicles). The simulation supports adaptable specification of sensor sets, environmental conditions, full control of all static and dynamic actors, map generation and much more. The simulator is based on Unreal Engine when the simulation is running using the OpenDRIVE standard.

The world represents the simulation and contains the main methods to generate i.e. the actors and the weather. Figure B.1 shows its visualization.



Figure B.1: CARLA World - Town 5.

B.1 Parking spots

The simulator offers the possibility to simulate several maps with different scenarios. In this case, the simulations have been carried out on Map_5_Opt. The chosen map is best suited to the parking situations required by this system, as it contains all the types of parking zones required by the system. In Figure B.2, the green areas show availability of parallel parking and the orange is a perpendicular outdoor parking. In such a way the simulation will have the possibility of parallel parking on the street and perpendicular parking in outdoor parking.

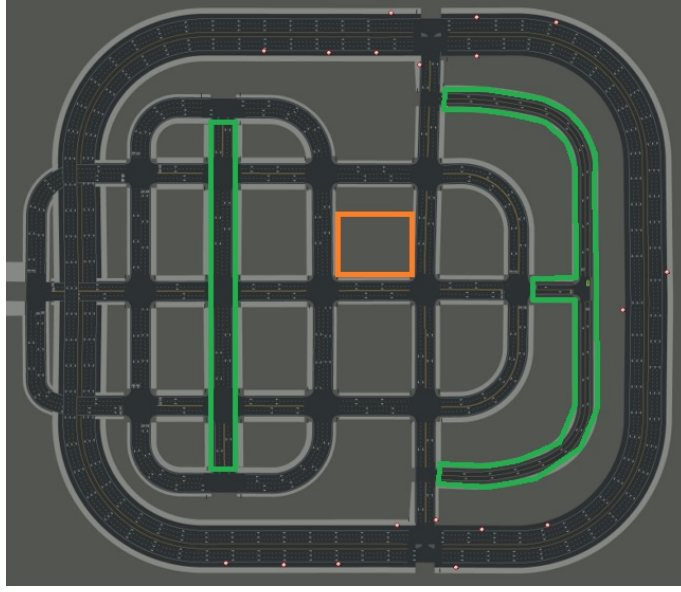


Figure B.2: Map_5_Opt parking zones.

Parking in the green regions is simpler due to the CARLA autopilot driving to the starting position of the parking maneuver. However, parking in the outdoor parking requires different implementation because the waypoints cannot be located inside the outdoor parking. For this reason, the route chart within the outdoor parking, which was already implemented in, has been re-adapted to the robotic environment.

The graph route inside the parking has concrete circulation directions, as shown in Figure B.3. This rule has been defined to ensure the circulation within the parking structure remains organized and concrete.



Figure B.3: Outdoor parking graph.

Appendix C

ROS2 architecture topics

The Table C.1 illustrates topics which aren't shown in the methodology.

Table C.1: Topic list.

Topic	Message
/carla/actor_list	CarlaActorList
/carla/client/walker_control_cmd	CarlaWalkerControl
/carla/control	CarlaControl
/carla/debug_maker	MarkerArray
/carla/ego_vehicle/APS_activation	Bool
/carla/ego_vehicle/APS_done	Bool
/carla/ego_vehicle/APS_maneuver_mode	Int8
/carla/ego_vehicle/APS_state_mode	Int8
/carla/ego_vehicle/APS_vehicle_mode	Int8
/carla/ego_vehicle/collision	CarlaCollisionEvent
/carla/ego_vehicle/control/ set_target_velocity	Twist
/carla/ego_vehicle/control/ set_transform	Pose
/carla/ego_vehicle/depth_front/camera_info	CameraInfo
/carla/ego_vehicle/ depth_front/image	Image
/carla/ego_vehicle/dvs_front/camera_info	CameraInfo
/carla/ego_vehicle/dvs_front/events	PointCloud2
/carla/ego_vehicle/dvs_front/image	Image
/carla/ego_vehicle/enable_autopilot	Bool
/carla/ego_vehicle/gnss	NavSatFix
/carla/ego_vehicle/hazard_detection	Bool
/carla/ego_vehicle/imu	Imu
/carla/ego_vehicle/lane_invasion	CarlaLaneInvasionEvent
/carla/ego_vehicle/lidar	PointCloud2
/carla/ego_vehicle/location_selection	Int8
/carla/ego_vehicle/next_target	Marker
/carla/ego_vehicle/objects	ObjectArray
/carla/ego_vehicle/odometry	Odometry
/carla/ego_vehicle/out_of_ODD	Bool

Topic	Message
/carla/ego_vehicle/parking_maneuver_activation	Bool
/carla/ego_vehicle/parking_maneuver_ready	Bool
/carla/ego_vehicle/parking_selection	Bool
/carla/ego_vehicle/radar_front	PointCloud2
/carla/ego_vehicle/rgb_front/camera_info	CameraInfo
/carla/ego_vehicle/rgb_front/image	Image
/carla/ego_vehicle/rgb_view/camera_info	CameraInfo
/carla/ego_vehicle/rgb_view/control/set_target_velocity	Twist
/carla/ego_vehicle/rgb_view/control/set_transform	Pose
/carla/ego_vehicle/rgb_view/image	Image
/carla/ego_vehicle/semantic_lidar	PointCloud2
/carla/ego_vehicle/semantic_segmentation_front/camera_info	CameraInfo
/carla/ego_vehicle/semantic_segmentation_front/image	Image
/carla/ego_vehicle/speed_command	Float64
/carla/ego_vehicle/speedometer	Float32
/carla/ego_vehicle/target_speed	Float64
/carla/ego_vehicle/vehicle_control_cmd	CarlaEgoVehicleControl
/carla/ego_vehicle/vehicle_control_cmd_manual	CarlaEgoVehicleControl
/carla/ego_vehicle/vehicle_control_manual_override	Bool
/carla/ego_vehicle/vehicle_info	CarlaEgoVehicleInfo
/carla/ego_vehicle/vehicle_status	CarlaEgoVehicleStatus
/carla/ego_vehicle/waypoints	Path
/carla/map	String
/carla/makers	MarkerArray
/carla/makers/static	MarkerArray
/carla/objects	ObjectArray
/carla/status	CarlaStatus
/carla/traffic_lights/info	CarlaTrafficLightInfoList
/carla/traffic_lights/status	CarlaTrafficLightStatusList
/carla/weather_control	CarlaWeatherParameters
/carla/world_info	CarlaWorldInfo
/clock	Clock
/final_parking_maneuver_goal_pose	Pose
/initial_maneuver_goal_pose	Pose
/initial_parking_maneuver_goal_pose	Pose
/initialpose	PoseWithCovarianceStamped
/parameter_events	ParameterEvent
/tf	TFMessage

Appendix D

ROS2 architecture messages

The first message is named CarlaWeatherParameters and is part of carla-msgs type, structured in nine Float 32 parameters.

Table D.1: CarlaWeatherParameters message.

Variable	Units	Type	Description
cloudiness	-	Float32	Cloudiness proportion
precipitation	-	Float32	Precipitation proportion
precipitation_deposits	-	Float32	Precipitation deposits proportion
wind_intensity	-	Float32	Wind intensity proportion
fog_density	-	Float32	Fog density proportion
fog_distance	-	Float32	Fog distance proportion
wetness	-	Float32	Wetness proportion
sun_azimuth_angle	-	Float32	Sun azimuth angle proportion
sun_altitude_angle	-	Float32	Sun altitude angle proportion

Another type of message is known as std-msgs which contains different possibilities of messages, for example, Pose and PoseStamped. This project has preferred the use of Pose because it is the best message to communicate with the carla-waypoint node.

Table D.2: Pose message.

Variable	Units	Type	Description
position.x	-	Float64	Position x
position.y	-	Float64	Position y
position.z	-	Float64	Position z
orientation.x	-	Float64	Orientation x
orientation.y	-	Float64	Orientation y
orientation.z	-	Float64	Orientation z
orientation.w	-	Float64	Orientation w

Appendix E

ROS2 rqt-graph

The Figure represents the rqt-graph visualization through ROS2, showing all the node connections of the APS architecture when it is running.

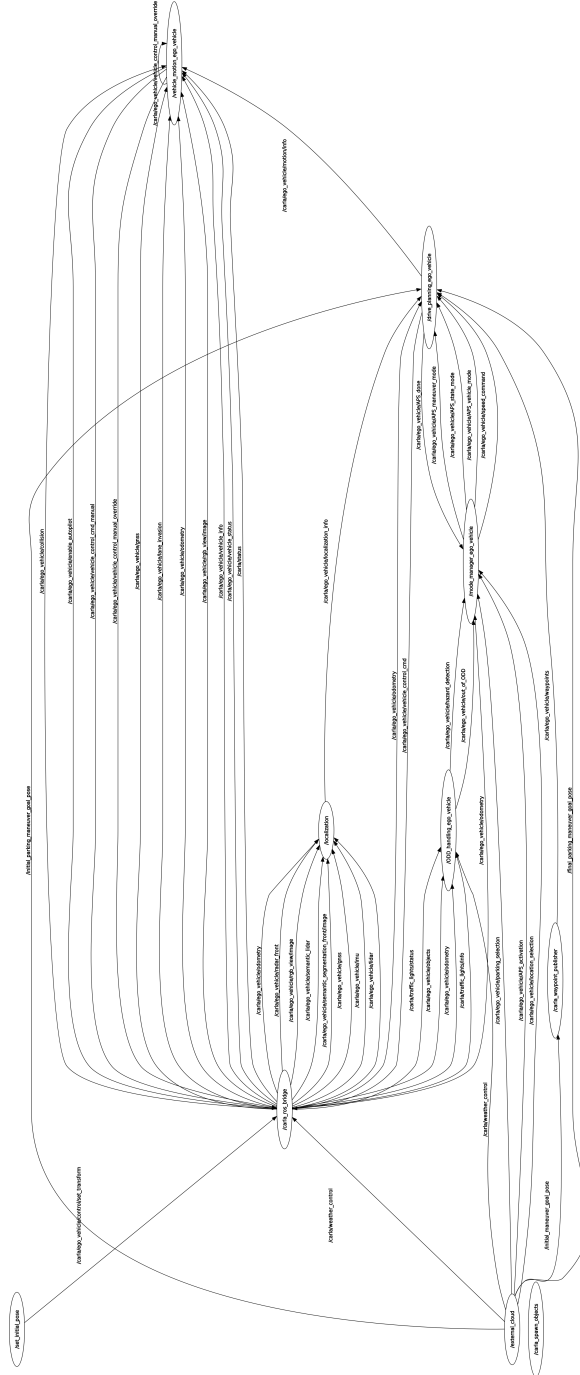


Figure E.1: ROS2 rqt-graph plot.

Appendix F

ROS2 sequence diagram example

Figure F.1 shows the perpendicular maneuver sequence diagram without obstacle hazards and without exit the ODD.

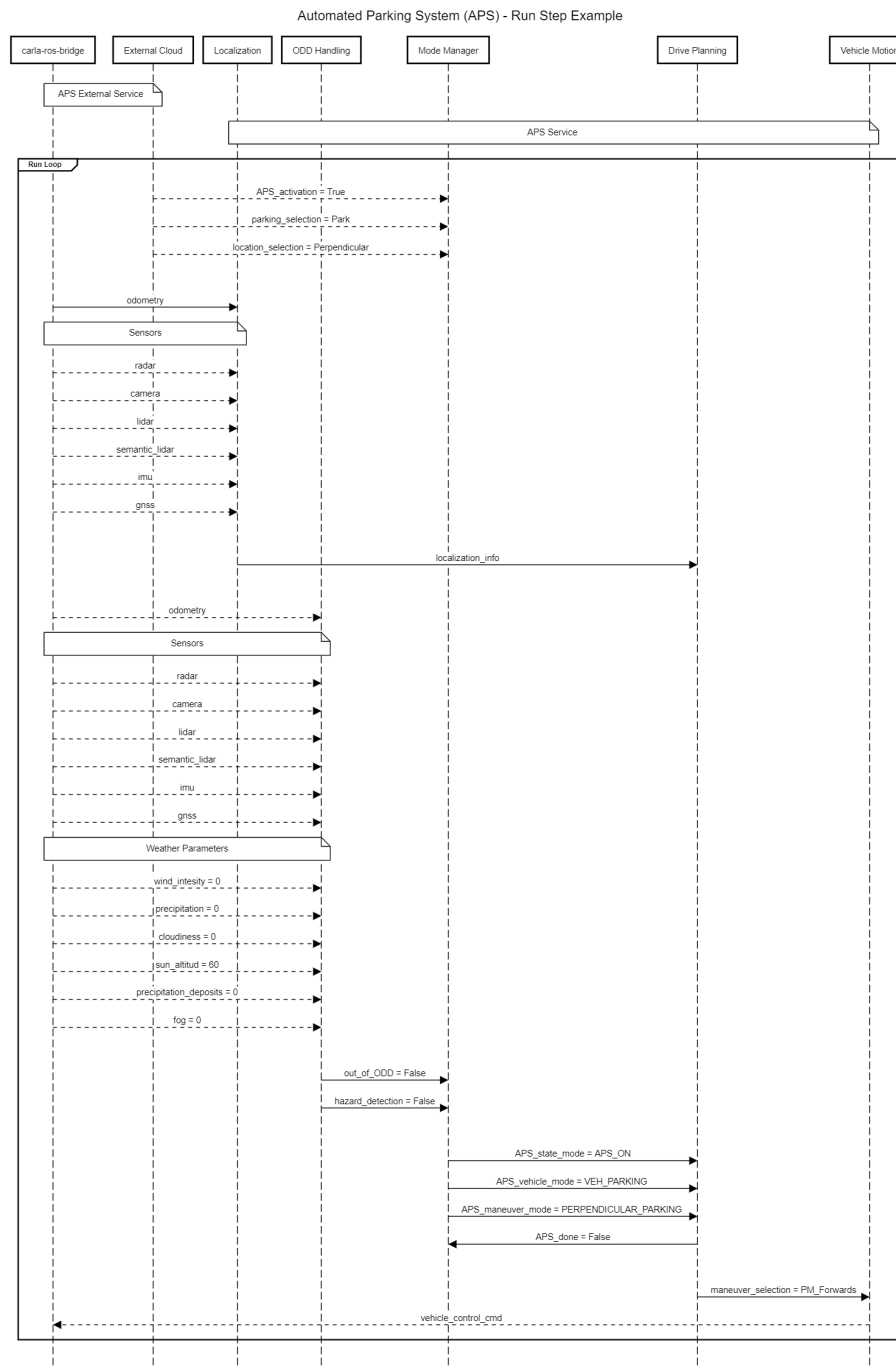


Figure F.1: Sequence diagram example PM.Forward in correct situation.