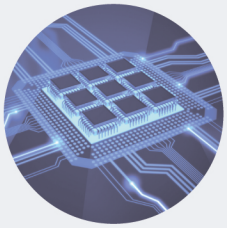
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS



Towards the simulation of cooperative perception applications by leveraging distributed sensing infrastructures

MIGUEL FERREIRA OLIVEIRA
outubro de 2023

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Towards the simulation of cooperative perception applications by leveraging distributed sensing infrastructures

Miguel Ferreira Oliveira

Dissertation submitted in partial fulfilment of the requirements for the Master's degree in Critical Computing Systems Engineering

Supervisor: Ricardo Augusto Rodrigues Da Silva Severino

Evaluation Committee:
President:
Luis Miguel Pinho, ISEP

Members:
André Dias, ISEP
Ricardo Severino, ISEP

Porto, September 27, 2023

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

The work presented in this document is original and authored by me, and performed in the scope of the Master's degree in Critical Computing Systems Engineering.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration, all references have been acknowledged and fully cited, and all text was originally produced by me (except when duly noted).

I further declare that I have fully followed the Code of Good Practices and Conduct of the Polytechnic Institute of Porto.

Porto, September 27, 2023

Abstract

With the rapid development of Automated Vehicles (AV), the boundaries of their functionalities are being pushed and new challenges are being imposed. In increasingly complex and dynamic environments, it is fundamental to rely on more powerful onboard sensors and usually AI. However, there are limitations to this approach. As AVs are increasingly being integrated in several industries, expectations regarding their cooperation ability is growing, and vehicle-centric approaches to sensing and reasoning, become hard to integrate. The proposed approach is to extend perception to the environment, i.e. outside of the vehicle, by making it smarter, via the deployment of wireless sensors and actuators. This will vastly improve the perception capabilities in dynamic and unpredictable scenarios and often in a cheaper way, relying mostly in the use of lower cost sensors and embedded devices, which rely on their scale deployment instead of centralized sensing abilities. Consequently, to support the development and deployment of such cooperation actions in a seamless way, we require the usage of co-simulation frameworks, that can encompass multiple perspectives of control and communications for the AVs, the wireless sensors and actuators and other actors in the environment. In this work, we rely on ROS2 and micro-ROS as the underlying technologies for integrating several simulation tools, to construct a framework, capable of supporting the development, test and validation of such smart, cooperative environments. This endeavor was undertaken by building upon an existing simulation framework known as AuNa. We extended its capabilities to facilitate the simulation of cooperative scenarios by incorporating external sensors placed within the environment rather than just relying on vehicle-based sensors. Moreover, we devised a cooperative perception approach within this framework, showcasing its substantial potential and effectiveness. This will enable the demonstration of multiple cooperation scenarios and also ease the deployment phase by relying on the same software architecture.

Keywords: autonomous vehicle, cooperative perception, micro-ROS, simulation, dynamic environment, distributed sensing

Resumo

Com o rápido desenvolvimento dos Veículos Autónomos (AV), os limites das suas funcionalidades estão a ser alcançados e novos desafios estão a surgir. Em ambientes complexos e dinâmicos, é fundamental a utilização de sensores de alta capacidade e, na maioria dos casos, inteligência artificial. Mas existem limitações nesta abordagem. Como os AVs estão a ser integrados em várias indústrias, as expectativas quanto à sua capacidade de cooperação estão a aumentar, e as abordagens de perceção e raciocínio centradas no veículo, tornam-se difíceis de integrar. A abordagem proposta consiste em extender a perceção para o ambiente, isto é, fora do veículo, tornando-a inteligente, através do uso de sensores e atuadores wireless. Isto irá melhorar as capacidades de perceção em cenários dinâmicos e imprevisíveis, reduzindo o custo, pois a abordagem será baseada no uso de sensores low-cost e sistemas embebidos, que dependem da sua implementação em grande escala em vez da capacidade de perceção centralizada. Consequentemente, para apoiar o desenvolvimento e implementação destas ações em cooperação, é necessária a utilização de frameworks de co-simulação, que abranjam múltiplas perspetivas de controlo e comunicação para os AVs, sensores e atuadores wireless, e outros atores no ambiente. Neste trabalho será utilizado ROS2 e micro-ROS como as tecnologias subjacentes para a integração das ferramentas de simulação, de modo a construir uma framework capaz de apoiar o desenvolvimento, teste e validação de ambientes inteligentes e cooperativos. Esta tarefa foi realizada com base numa framework de simulação denominada AuNa. Foram expandidas as suas capacidades para facilitar a simulação de cenários cooperativos através da incorporação de sensores externos colocados no ambiente, em vez de depender apenas de sensores montados nos veículos. Além disso, concebemos uma abordagem de perceção cooperativa usando a framework, demonstrando o seu potencial e eficácia. Isto irá permitir a demonstração de múltiplos cenários de cooperação e também facilitar a fase de implementação, utilizando a mesma arquitetura de software.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence.
AMCL	Adaptive Monte-Carlo Localizer.
AV	Automated Vehicle.
BEV	Bird's Eye View.
BSM	Basic Safety Messages.
BSP	Basic System Profile.
C-ITS	Cooperative Intelligent Transportation Systems.
CACC	Cooperative Adaptive Cruise Control.
CAM	Cooperative Awareness Message.
CAN	Controller Area Network.
CoVP	Cooperative Vehicular Platooning.
CRTP	Crazyflie Real-Time Protocol.
DDS	Data Distribution Service.
DENM	Distributed Environment Notification Messages.
DSRC	Dedicated Short-Range Communication.
ETSI	European Telecommunications Standards Institute.
GCS	Ground Control Station.
GNSS	Global Navigation Satellite System.
HMI	Human-Machine Interface.
IoT	Internet Of Things.
ITS	Intelligent Transportation Systems.
LDM	Local Dynamic Map.
LiDAR	Light Detection And Ranging.
LTE	Long Term Evolution.
M2M	Machine-to-Machine.
MAV	Micro Aerial Vehicle.
MCU	Microcontroller Unit.
MTU	Maximum Transmission Unit.

Nav2	Navigation 2.
OBU	Onboard Unit.
OEM	Original Equipment Manufacturers.
QoS	Quality of Service.
RAM	Random Access Memory.
RMW	ROS Middleware.
ROS	Robot Operating System.
RSU	Roadside Unit.
RTC	Real-Time Clock.
RTOS	Real-Time Operating System.
SLAM	Simultaneous Localization and Mapping.
SWOT	Strengths, Weaknesses, Opportunities, Threats.
TCP	Transmission Control Protocol.
UART	Universal Asynchronous Receiver-Transmitter.
UDP	User Datagram Protocol.
URDF	Unified Robotics Description Format.
UWB	Ultra-Wideband.
V2I	Vehicle-To-Infrastructure.
V2P	Vehicle-To-Pedestrian.
V2V	Vehicle-To-Vehicle.
V2X	Vehicle-To-Everything.
VANET	Vehicular Ad-hoc Network.
WAVE	Wireless Access in Vehicular Environments.
XRCE-DDS	eXtremely Resource Constrained Environments Data Distribution Service.
YOLO	You Only Look Once.

Chapter 1

Introduction

1.1 Problem Statement

Automated Vehicles (AV) rapid development pace has been pushing the boundaries of their functionalities, and also imposing new challenges. AVs have been increasingly integrated into several industries such as retail, logistics, the military, and healthcare. Humans have long desired machines capable of independent task performance. While progress has been made across various fields, the automotive industry has yet to realize fully automated (Level 5) vehicles, remaining in the theoretical realm. Critical considerations for achieving this goal include addressing safety, reliability, and the complexity of current autonomous vehicle technology. The challenges faced, including the need for successful road tests and the high cost of advanced technology, raise questions about the feasibility of realizing fully automated vehicles in the near future. Level 5 represents the utmost degree of vehicle autonomy, characterized by vehicles that are entirely self-driving and independent of human intervention. These vehicles can navigate diverse road conditions, weather, and travel beyond predefined geographical boundaries. At this level, vehicles are equipped with robust safety and emergency features, ensuring their ability to operate safely and autonomously. The major challenge in deploying level 5 autonomous vehicles for public use is the insufficient infrastructure in most countries. These vehicles rely on external infrastructure to facilitate their sensory and control systems for safe operation on public roads. Adapting the necessary infrastructure to support autonomous vehicles is a time-consuming process, likely taking over a decade for countries to complete. To address this, original equipment manufacturers (OEMs) should prioritize the shift towards software-defined vehicles and enhance communication capabilities between vehicles and external systems. As per a study conducted by KPMG International, which assessed countries' readiness for autonomous vehicles based on criteria such as policies, technology, infrastructure, and consumer acceptance, only 16 countries achieved a score exceeding ten. This suggests that the majority of countries are not prepared for the adoption of autonomous vehicles (Kosuru and Venkitaraman 2023). Such challenges become particularly more critical in mixed environments which may present some kind of human-machine cooperation.

In a highly dynamic environment with both humans and AVs in play, the strategies planned to fulfil certain tasks keep changing, depending on the status of the environment, the actors involved, and the progress in the planned operation. For that, it is fundamental to rely on onboard sensing, often relying on artificial intelligence (AI) to learn and reason about the environment, but also sensors deployed in the environment (e.g. cameras, LiDAR and radar) as well as infrastructure-based computing and sensor fusion, to gather the information in real time and process it. These sensing and reasoning processes, both about the environment and vehicle perception, have proven to be hard to integrate, which has been becoming a

major hurdle in deploying safe human-machine cooperation. They are usually integrated with extensive onboard sensing devices with high computational capacity, and thus are quite expensive and hard to gain access to. That is why the use of distributed sensing devices is important since it minimizes the cost and enhances the accessibility.

Since the perception of the environment is a major requirement for the vehicle's ability to adapt and overcome dynamic and unpredictable scenarios, the motivation of this work is to improve the perception ability by enabling it closer to the environment while relying on low power sensing devices. As the utilization of the sensing devices will depend on their scaled deployment, it becomes essential to explore an efficient communication method between them and the vehicle. This consideration is particularly crucial in scenarios where time constraints are at play, potentially impacting data accuracy due to communication delays. The need to mitigate such issues is paramount to maintain the data's reliability and relevance for its intended use.

Additionally, to explore the cooperative perception approach's development without requiring access to physical hardware and sensors, a method for simulating the entire environment, including sensors, vehicles, and communication, will also be investigated. This simulation capability will facilitate the development of use-case scenarios, crucial for validating new approaches, and will also aid the analysis of the transactions communication.

1.2 Research Objectives

This thesis aims at integrating distributed sensors in an infrastructure using an AV perception layer, adopting recent ROS tool-sets for these type of devices eg. micro-ROS. This approach will be supported by relying on existent co-simulation frameworks such as CO-PADRIVe or AuNa, which will be leveraged to integrate these sensing interconnections. The main goal is to contribute to a co-simulation framework that can simulate complex highly dynamic environments, while relying on heightened perception via a smart sensing wireless infrastructures.

1.3 Research Contributions

The following are the main contributions of this thesis:

- Overview of the fundamental concepts of IoT communications, cooperation, cooperative perception, and smart hybrid environments.
- Research on current distributed cooperation perception techniques, IoT technologies, and relevant simulation toolsets to support the sensing infrastructure, as well as current robotic/perception frameworks.
- Design of a proof-of-concept system's architecture to carry out distributed sensing, relying on micro-ROS, for enhanced and cooperative perception in smart hybrid environments, by extending a previous developed simulation framework named AuNa.
- Implementation of the above mentioned architecture using a simulation framework.
- Simulation analysis and evaluation of the performance and limitations of the proposed architecture.

1.4 Thesis Structure

The remain of this thesis is organized into the following chapters. Chapter 2 delves into the state of the art, providing an explanation of various concepts related to the thesis scope. It also examines related works that apply these concepts and discusses relevant simulation frameworks that served as a basis for this study. In Chapter 3, we introduce the relevant technologies utilized in developing the framework. Moving forward, Chapter 4 outlines the simulation framework's concept, showcasing a use-case scenario employed for preliminary testing. The chapter also offers an overview of the framework's overall architecture, illustrating how the technologies collaborate to achieve desired simulation scenarios. Chapter 5 delves deep into the process of developing the simulation framework, providing insights into the implementation of two specific scenarios. Chapter 6 presents the analysis of experimental results obtained from tests conducted to validate the work. Finally, the thesis concludes with Chapter 7, summarizing the key findings and drawing conclusions. Additionally, this chapter examines the limitations of the proposed approach and offers insights into potential future work to further enhance the framework's capabilities.

Chapter 2

State of the Art

2.1 Concepts

2.1.1 Smart Environments and Internet-of-Things (IoT)

An environment is considered smart whenever it is composed by systems which employ sensors to perceive their environment, interpret the perception on an abstract level to yield some non-trivial ("intelligent") decisions beyond simple reactive sense-act rules, and are able to interact with the environment or alter it using some actors. In short, smart environments and systems acquire and apply knowledge in order to assist people.

From the viewpoint of human-machine interaction, smart environments present a multitude of new options for interacting with users and offering new services. While classical user interfaces rely on explicit input from users, smart environments can offer proactive services (Wolter and Kirsch 2017).

In this regard, the IoT has enormous potential to support the communication infrastructures with sensing and actuator capabilities. Internet of Things (IoT) refers to the network of interconnected devices, such as sensors, that quickly exchange large amounts of data. The devices are connected through Machine-to-Machine communication (M2M) which is a standardized method for different machines to share information globally. IoT is commonly used for smart environments such as smart traffic management, healthcare and medicine, smart cities, self-driving vehicles, etc.

The number of devices connected to the internet is steadily multiplying, however global Internet coverage is still far from being achieved as more than half of the world's population lacks access to the Internet, even so, it could become a limitation for IoT. Another limitation is the availability of data, as IoT heavily relies on data by nature. The IoT sector is in high demand for scalable methods to handle, process, analyze and mine data in real-time as a large volume of data is exchanged daily through interconnected computing devices, multiple fields of computer science such as cloud computing, cybersecurity, AI, Big Data, and data analytics can benefit from IoT (Gazis 2021).

One of the primary goals of creating a smart environment is to enhance human life in terms of comfort and efficiency. In this context, the IoT paradigm has gained prominence as a foundational technology for establishing and advancing smart areas. This paradigm plays a pivotal role in the development of various smart environments, including smart cities, smart homes, smart campuses, and smart healthcare, by enabling automated communication among interconnected devices to facilitate a wide range of services (Ikrissi and Mazri 2021).

In this line, cooperative perception is a fundamental part to enable such environments.

2.1.2 Cooperative Perception

Automated vehicles rely on sensors (eg. LiDARs, radars and cameras) for perception and localization of the driving environment. This environment is composed by static elements (eg. road, traffic signs, other objects) and dynamic elements (eg. pedestrians, other vehicles). However, there are still limitations regarding dynamic elements perception, and that is why it is necessary cooperative perception in order to improve AVs sensing and perception capabilities.

Cooperative perception enables vehicles to exchange their sensor data, which provides additional data about the driving environment, including data beyond their on-board sensors' field of view. This improves detection accuracy and increases the confidence about the detected objects. It relies on V2X (Vehicle-To-Everything) communications, which consists in communicating with external entities (eg. other vehicles, external sensors, etc.), in order to gather information about the environment to improve its accuracy (Thandavarayan, Sepulcre, and Gozalvez 2020).

(Caillot et al. 2022) presents a survey that focuses on the cooperative environment. It offers insights into the various architectures that can be used to establish such systems, along with a comprehensive examination of the challenges arising from cooperation.

The authors assume that the purpose of perception is to represent the elements around the ego-vehicle as well as its status in the scene. It is divided into three key aspects (figure 2.1): ego-vehicle localization, the identification and tracking of other users, and the mapping of the environment. Cooperation is a crucial element in this context, it involves the utilization of data from other agents to enhance perception tasks or refine outcomes. This collaboration can occur at varying levels, ranging from raw data sharing (early fusion) to preprocessed data (mid fusion) and processed data (late fusion). The early fusion phase focuses on the aggregation of raw data, integrating sensor-provided information from connected users at a specific timestamp while considering sensor transformations. This stage also involves sharing the ego vehicle's raw data with other users. The subsequent stage is characterized by two concurrent tasks: one deduces the vehicle's location in the environment using sensor data, which can be complemented by measurements from connected users; the other task involves the detection and tracking of objects within the scene, benefiting from data contributed by connected users to enhance the global perception of the environment. These tasks collectively generate feature-level data that can be shared with other users, forming the core of the perception process. The final stage is geared towards constructing a map, providing context to the previously gathered data.

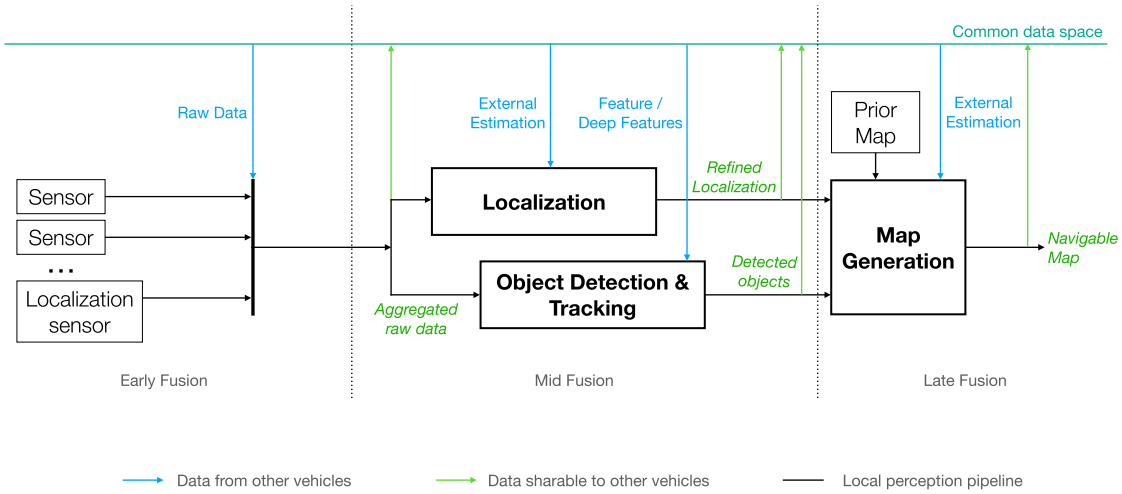


Figure 2.1: Block diagram of the minimal perception pipeline in a vehicle. (Caillot et al. 2022)

Sensors constitute the fundamental elements of any perception system, enabling us to perceive both our own presence and the surrounding environment. The authors reviewed several sensors and reached the conclusion that the sensors mentioned accomplish their respective tasks but also encounter limitations. Hence, sensor fusion becomes imperative to overcome the individual shortcomings of each.

Regarding communications, users must select a specific network architecture to facilitate data sharing, with the most prevalent choice being the Vehicular Ad-hoc Network (VANET) configuration, which connects vehicles within range of one another. In VANET, a shared channel coordinates the network, while data is disseminated on distinct channels and relayed through vehicles as they traverse between sender and receiver. Evaluating the physical layer requisites in a VANET network led to the introduction of an amendment to IEEE 802.11 known as Wireless Access in Vehicular Environments (WAVE), specifically IEEE 802.11p. In Europe, this standard was leveraged to establish the ITS-G5 standard. Consequently, two communication protocols are rooted in these standards, namely Dedicated Short-Range Communication (DSRC) and Cooperative-Intelligent Transportation Systems (C-ITS). The DSRC protocol utilizes Basic Safety Messages (BSM) to package shared information, transmitting details about the emitting vehicle to prevent collisions. Likewise, C-ITS employs Cooperative Awareness Messages (CAM), which convey vehicle data similarly to BSM but additionally introduces Distributed Environment Notification Messages (DENM). DENM serves to alert road hazards, possessing a higher priority than CAM messages.

When multiple users engage in interaction, it becomes essential to establish the communication framework. The authors categorize this into two primary approaches: the centralized approach and the distributed approach.

- **Centralized Approach** - In the centralized approach, the emphasis is on data processing. Here, users share their gathered data with a single central point, often a roadside processing unit. This unit handles the data processing and extracts relevant information, which is subsequently shared with all users. An illustration of this occurs when sensors positioned on highway gantries transmit data to a roadside processing unit, creating a digital representation of the road section accessible to all users. However,

a notable drawback of this approach is its dependency on the processing capabilities of the central unit, which affects its overall efficiency.

- **Distributed Approach** - Unlike the centralized approach, the distributed method involves users transmitting their collected data directly to all vehicles in real-time. This means that data processing takes place onboard each individual vehicle. An example of this approach was seen in a related study where vehicles broadcasted their state and intentions, leading to event anticipation and enhanced control. Nonetheless, the system employed connected cars that needed to be in close proximity, limiting the network's reach. These cases share the characteristic of processing data onboard every vehicle, yet the quality of coverage is determined by the scale of the user fleet.

Both the centralized and distributed approaches come with their own sets of advantages and drawbacks. It's evident that the distributed method is currently more prevalent, primarily due to the prevalence of vehicle-to-vehicle (V2V) approaches in most cooperative applications. However, we're witnessing a growing presence of applications based on centralized approaches in today's landscape.

Regarding the challenges imposed by cooperation, the authors identified the following:

- **Multi-Modality** - Multi-modality has been present in the field of robotic perception since its inception. The adoption of multi-modality has been driven by diverse requirements. For instance, cameras handle tasks such as detection and classification, while radars manage distance and speed measurement. One approach to tackle the complexities of multi-modality and calibration is to process the data independently for each sensor and then share the outcomes through messages. Nonetheless, handling data association during the aggregation phase remains a demanding task.

- **Calibration** - One of the major challenges in the perception pipeline is calibration, particularly in a cooperative environment. The goal of calibration is to establish the transformation between sensors, enabling the integration of data from multiple viewpoints, at least for a specific frame. While this task is already complex for a single agent, it becomes even more intricate in a multi-user mobile environment. In such scenarios, the transformation matrix between sensors undergoes constant changes as the vehicles move within the scene, often leading to long baselines. Additionally, synchronization becomes challenging due to the absence of a physical triggering line.

- **Synchronization** - Synchronization poses a significant challenge, particularly in a cooperative setting. In this context, calibration heavily depends on synchronizing the elements to establish the transformation between sensors, a task that becomes especially complex with mobile sensors. Multiple factors contribute to desynchronization, including clock offsets and communication delays. While the clocks may be synchronized, there's no guarantee that their acquisitions are triggered simultaneously, introducing uncertainty during the data merging process. Additionally, different sampling rates necessitate interpolation between acquired or predicted data, further complicating the synchronization process.

- **Point of Views** - The diversity of perspectives becomes particularly pronounced in scenes involving both infrastructure and mobile users. It raises a fundamental question about how to handle data that may seem fundamentally different, even to the extent of being non-overlapping. For instance, a scenario where one sensor observes the front

left corner of a car, while another sensor focuses on the right back corner of the same vehicle.

- **Perception Matching** - The alignment of perceptions between objects sensed by other agents and those shared with the ego vehicle, alongside the objects detected by the local sensor, poses a typical challenge in cooperative systems. A fundamental approach is to match these objects based on their positions, but the presence of sensor-induced noise can introduce errors in this process.

Throughout the rest of the document, the authors discuss and analyse different approaches regarding the three main blocks of the perception pipeline in a cooperative context, which are the localization, object detection and tracking, and mapping. The following are the authors conclusions and considerations towards the mentioned topics:

- **Localization** - Cooperative position estimation, as exemplified by GNSS (Global Navigation Satellite System), employs the multilateration technique to estimate the position of a point by measuring the distance to multiple anchor points. This approach refines standalone position estimations by incorporating additional data. However, it's essential to note that an agent's limited localization capability can significantly impact the overall results. Furthermore, cooperative localization serves as a valuable alternative for GNSS-denied environments, particularly when well-located measurement points, such as infrastructure, are available.
- **Object detection and tracking** - Leveraging multiple points of view offers a significant advantage in mitigating sensor limitations and mitigating the impact of changing scene conditions. It's worth noting that tracking methods often play a secondary role, primarily serving as a means to acquire results in other segments of the perception pipeline. The authors have observed a notable lack of research efforts within the cooperative domain, particularly in the realm of detection and tracking. The authors attribute this scarcity of experimentation in cooperative contexts to the substantial bandwidth demands of communication, as well as the sensitivity to issues like desynchronization and pose estimation errors.
- **Mapping** - The synergy of ego-localization and the identification and tracking of objects can lead to the creation of a shared map. This cooperative mapping process involves the aggregation of information from multiple agents, resulting in a collaborative map-building effort. It's evident that the current application of cooperative mapping primarily aims to optimize routes and enhance vehicle navigation by preemptively responding to various events along the user's path. However, these objectives could be extended further, particularly towards real-time trajectory prediction, benefiting from reduced latency and improved data accuracy through the sharing of information.

The comprehensive review encompassed the three core components of the perception pipeline within a cooperative framework. Additionally, the examination extended to the available cooperative system architectures, their respective strengths and limitations, as well as the challenges inherent in cooperative solutions. This thorough analysis enabled the authors to formulate a SWOT (Strengths, Weaknesses, Opportunities, Threats) assessment. This evaluation provides a concise overview of the current state of the art in cooperative perception, with a specific focus on systems that integrate infrastructure elements.

Strenghts:	Weaknesses:
• More precise localisation in environment with GNSS • Localisation possible in GNSS denied environment • No drift in Localisation • V2X Mapping • Better reliability • Cost reduction • Less occlusion • Real-Time update (solely depending on the transfer latency and computation time) • Larger field of view • Detection of unconnected User	• Similar precision of pose estimation with non cooperative system • Dependent to the number of users • Computation expensive • High throughput required • Latency
Opportunities:	**Threats:**
• Raw sensor data fusion • Various point of view of the scene • V2I Map generation • V2I Object management • Infrastructure always available and calibrated • Further Trajectory planning • Anticipation of dangers • Infrastructure offers more storage and can delete duplicate parts allowing storing HD maps • Better Bird Eyes View map creation • Existing matching methods	• Higher cost for the infrastructure • Lack of normalisation between constructors • Consistency of the accuracy of the pose estimation between the sensors • Detection and classification accuracy of each participant • Synchronisation between participants • Data association of a single object with a very different point of view. • Missing stream or data management • Data management between mobile and fixed users

Table 2.1: Cooperative Perception - SWOT (Caillot et al. 2022).

2.2 Related Works

This section explores related works that not only apply the aforementioned concepts but also leverage pertinent technologies, which will be further explained in the next chapter.

2.2.1 Seamless Distributed Sensing

Given that the utilization of distributed sensing is one of the primary objectives of this work, this subsection will provide an overview of related works that employ micro-ROS, a technology specifically designed for such devices. We will focus on the OFERA project, which serves as the foundational initiative for micro-ROS.

OFERA Project

The OFERA (Open Framework for Embedded Robot Applications) project (OFERA 2020) is an EU founded project whose objective is contribute to the faster growth of a competitive industry of small robots and robot components manufacturers, using highly resource-constrained devices. OFERA purpose is to bridge the technological gap between the ROS platform for high-performance computational devices and the low-level libraries for micro-controllers, which will allow for microprocessors and microcontrollers to be mixed together seamlessly in any robotic system.

The name of the new robotic framework targeting embedded and deeply embedded robot components with extremely constrained computational resources is micro-ROS, which will be further explained in the next chapter.

Sensor micro-ROS use-case in Modular Arm

Exploring the OFERA project public deliverables, the following OFERA publication (Goenaga, Muñoz, and Vilches 2019) describes a Modular Arm use-case that includes a micro-ROS enabled distance sensor that will be performed by Acutronic Robotics.

This use-case aims to replicate an industrial setting involving the use of a modular arm for pick-and-place operations. The used modular arm was MARA, which is the first modular robot arm that runs natively ROS 2 made by Acutronic Robotics. By default, the modular robot arm lacks any sensors enabling it to perceive its surroundings. Consequently, it can only execute pick-and-place tasks based on predetermined locations where objects are assumed to be positioned (as demonstrated in the video presented here). Thanks to the integration of a micro-ROS-enabled distance sensor, the robot arm will gain the capability to detect when a new package enters its operational range. Utilizing the distance sensor's measurements of the gap, the robot controller will be able to compute the trajectory and status of each joint, effectively coordinating the movement of MARA's joints and the closing of its gripper to grasp the object. Once secured, the arm will then deposit the load at a specified destination.

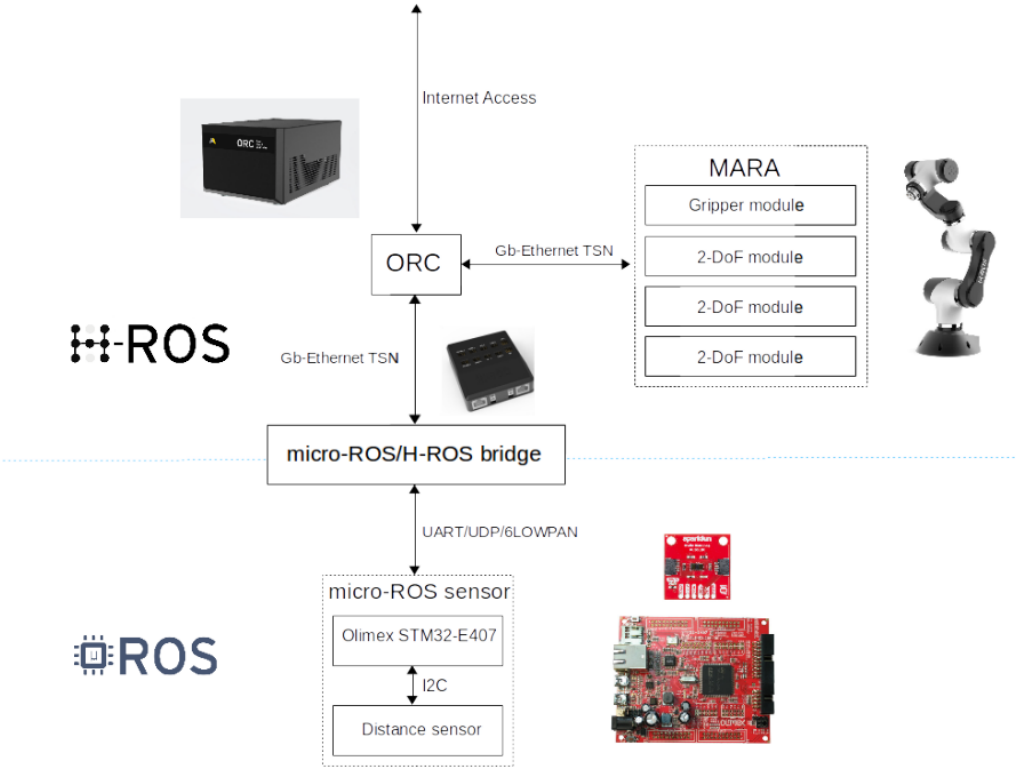


Figure 2.2: Use-case hardware topology. (Goenaga, Muñoz, and Vilches 2019)

As depicted in the image, this demonstration involves the utilization of two commercial products: MARA and the ORC robot controller, which are interconnected via the H-ROS bus, founded on TSN-compliant Gigabit-Ethernet technology. Additionally, a micro-ROS

distance sensor has been incorporated into the setup. This module is built upon the Hardware Reference Development Platform Olimex-STM32 E407 and employs a VL53L1X distance sensor, with both components linked through an I2C peripheral. The purpose of this device is to provide distance measurements to the ORC. Furthermore, the implementation of a micro-ROS to H-ROS bridge is necessary because the sensor is not inherently integrated into the H-ROS ecosystem. This bridge ensures that the sensor module can seamlessly collaborate with robot modules based on H-ROS.

In conclusion, the authors consider that this use-case demonstrates how the OFERA project's work can facilitate the integration of heterogeneous robotic applications, enabling the co-existence of microcontroller-based and microprocessor-based robot modules. Additionally, it serves as a means to validate the features that consortium members seek in micro-ROS, including ease of implementation, utilization of resource-constrained devices in cutting-edge robotics technology, and power-saving mechanisms.

Drone micro-ROS Use-Case

In this OFERA deliverable (Sanchez et al. 2021), the authors showcase a use-case whose purpose is to highlight the advantages of micro-ROS, which include its minimal resource utilization and its adaptable and segmented communication system. This emphasis is especially centered on the middleware implementation of micro-ROS, known as eProsima Micro XRCE-DDS. The use-case scenario involves a micro aerial vehicle (MAV), which is a vehicle smaller than 15 cm in any dimension, with a maximum payload of just a few grams. The primary limitation in MAVs lies in power constraints, primarily due to the significantly lower energy efficiency of batteries when compared to gasoline used in propulsion systems. The communication system, together with the propulsion system, is one of the most power demanding systems, and for that reason, it is necessary a communication system with low power consumption and weight at the cost of low bandwidth and range. The authors consider that micro-ROS fits perfectly with MAV environments for several reasons. Firstly, it's designed to work efficiently on resource-constrained devices like those found on MAVs. Secondly, it offers support for multiple transport protocols (UDP, TCP, Serial, 6LoWPAN, etc.) through its communication middleware, eProsima Micro XRCE-DDS. Lastly, micro-ROS follows a publish-subscribe pattern, which is inherited from ROS2.

In this scenario, the MAV navigates a designated area under the guidance of a flight operator using a ground control station (GCS). Within this area, various remote sensors are distributed, responsible for collecting environmental data like temperature and humidity. The flight operator directs the MAV towards these remote sensors, and upon reaching each one, the MAV establishes a connection to retrieve their data. Ultimately, the flight operator directs the MAV back to its home position.

There are three micro-ROS users, the MAV, the GCS, and the remote sensors, which interact in distinct methods. Firstly, between the GCS and the MAV, a client-server pattern is employed. The GCS serves as the server utilizing a micro-ROS agent application, while the MAV operates as the client with a micro-ROS client application. This connection, maintained throughout the mission, relies on the proprietary CRTP radio protocol. Secondly, the interaction between the MAV and the Remote Sensors adopts a peer-to-peer pattern. Both entities act as clients, using a lightweight micro-ROS agent application, known as micro-ROS agent lite, on the remote sensor side. This micro-ROS agent lite serves as a centralized broker, enabling message exchange between the MAV and the remote sensors

without ROS 2 output. It facilitates dynamic discovery of the Remote Sensors by the MAV, and also utilizes the CRTP radio protocol.

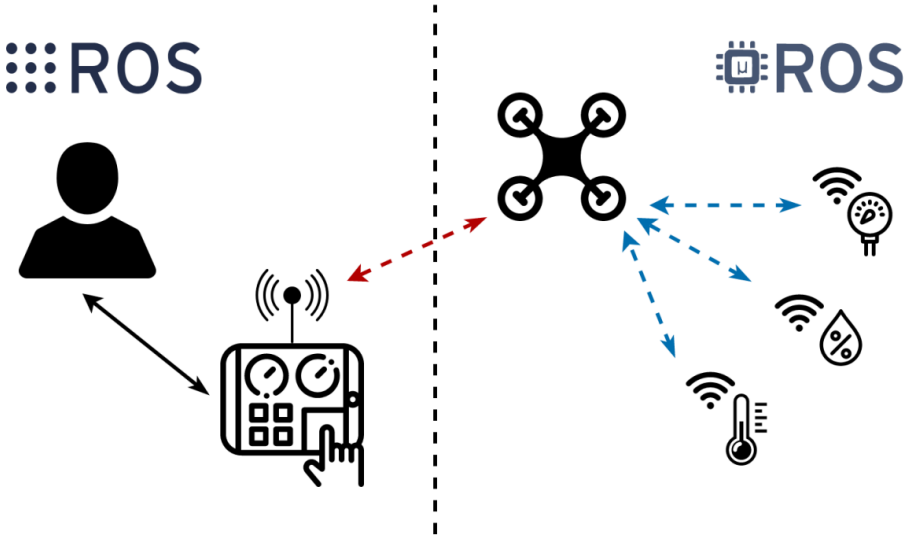


Figure 2.3: Drone use-case overview. (Sanchez et al. 2021)

Smart Warehouse micro-ROS Use-Case

The following work (Kołcon, Maciaś, and Malki 2019) depicts a use-case of a robot mobile platform that is capable of moving in a warehouse in order to perform scheduled tasks.

The purpose of this work was to demonstrate how micro-ROS can be used in a low power wireless network environment and checking how micro-ROS can operate in such environments, as providing good quality network connection over wireless in industrial application is very difficult. The robot was composed by the following components:

- Robot base.
  - PIAP Scout base.
- Autonomy module.
  - GNSS - satellite localization system for outdoor use.
  - UWB - radio localization system based on beacons, intended to use indoors.
- Robot control unit & Context system - for providing tasks to the robot, also to provide required context (world model) to execute the task, and is also the main place for monitoring system performance and task execution status.
- Radio link (WiFi) - main communication is using WiFi to communicate with the internet and also to send commands to the robot, as well as receiving status.
- Sensors and actuators.
  - Door opener - actuator mounted close to the warehouse door, used to control its opening and closing.
  - Laser scanner module - situated next to the road to provide information to the platform whether the road is empty.

– Final effector system - a amber warning light used for indication of scenario status.

– Humidity sensor - used to measure the humidity and temperature in the warehouse.

The control unit used for each sensor and actuator was a Olimex STM32-E407 board, which is a microcontroller and a reference platform for using micro-ROS.

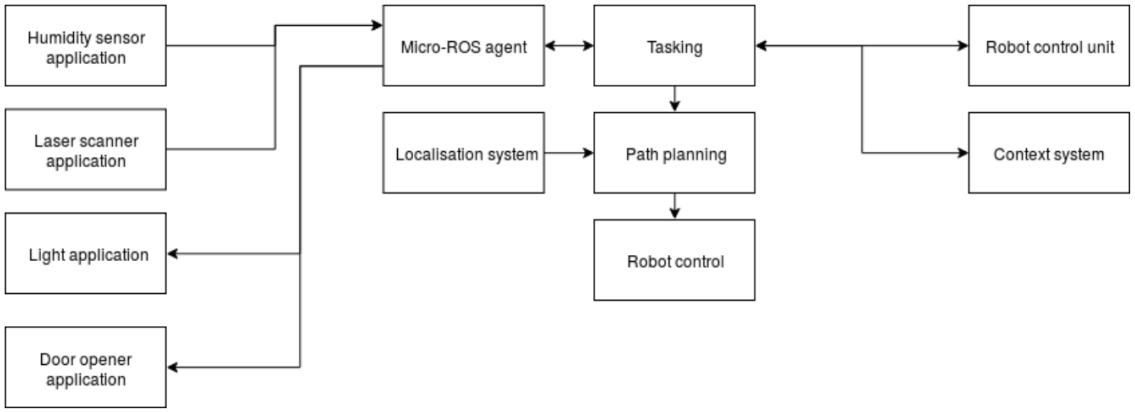


Figure 2.4: Software Architecture. (Kołcon, Maciaś, and Malki 2019)

In order to test micro-ROS features, the authors created missions for the robot to perform tasks according to the developed scenario. The scenario consists in a adapted laboratory hall which will simulate the warehouse. The robot mobile platform can go to the warehouse door in the meantime collecting data from the sensors like humidity sensor and LiDAR according to the planned tasks. After reaching the gate it can be opened using an actuator which uses 6LoWPAN and a micro-ROS stack for communication. To get to the final effector (light), the robot mobile base has to cross the road, so it must check whether crossing the road is safe.

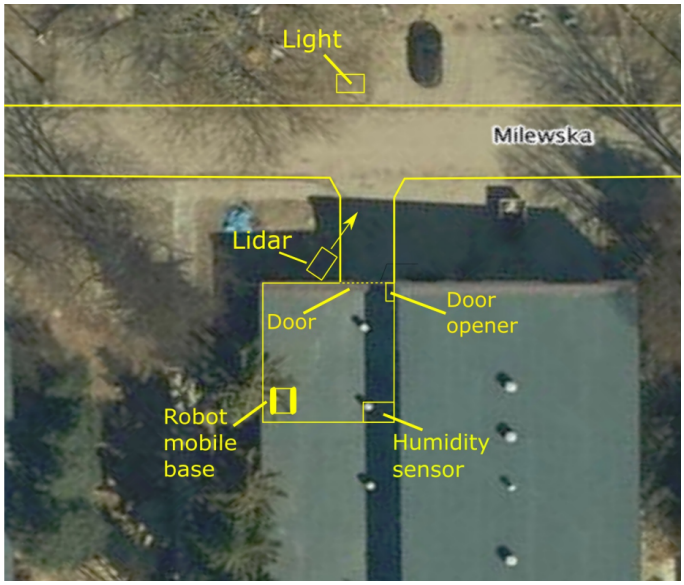


Figure 2.5: Warehouse Plan. (Kołcon, Maciaś, and Malki 2019)

According to the authors, this work showcased the possibilities of using micro-ROS tools effectively, and how simple it is to to combine basic elements such as sensors and actuators based on microcontrollers to a larger system such as the ROS/ROS2 environment.

Regarding this thesis, the analysis of this work proved to be very important due to the technologies and methods used to develop a smart environment with distributed sensors and actuators, the use of micro-ROS and the scenario used for testing demonstrated its potential.

micro-ROS default simulation environment release

The following work (Flores Muñoz and Muguruza Goenaga 2019), also inline with the OFERA project, studied how could micro-ROS be simulated in a regular computer. Several tools were analysed in order to check their suitability for micro-ROS execution. The main objective was to execute a simulation which includes the complete set of micro-ROS layers, including NuttX RTOS, Micro XRCE-DDS middleware, RMW and the micro-ROS client library.

The authors concluded that the simulator needed to fulfil the following requirements:

- UART communication with two available ports, where one is used for interfacing and the other one for micro-ROS Agent communication: Use of serial, 6LoWPAN or UDP/TCP through Ethernet to communicate with the micro-ROS Agent.
- RTC and Timer support: Basic functions of the RTOS, requires timing tools to work.
- Polling support: The communication on micro-ROS is asynchronous, so it is required a tool to check an incoming message.
- C++ support: It is heavily used by ROS2.
- 36 KB of RAM and 383 KB of flash: This requirement is based on the memory analysis made on the Olimex-STM32 E407 board and it shows the minimum memory size that is required for micro-ROS execution.

After some research, the authors found two options that could meet the requirements, NuttX official simulator and QEMU. NuttX simulator doesn't simulate any architecture, it only simulates the NuttX RTOS ifself. QEMU however, is capable of simulating CPU and microcontroller architectures.

The authors further analysed the NuttX official simulator and found the following limitations:

- Different RTOS behaviour: it does not simulate the MCU architecture.
- Limited support for ucLibc++ library: the library does not compile on X86/X64 architecture.
- Networking: the authors were able to achieve a local connection between the host PC and the simulator, but lost the internet connection.
- External Serial: It is a requirement for micro-ROS, but the authors were unable to use it, even though NuttX stated that it is possible to run a virtual serial port.

Given the previous limitations, the authors concluded that the NuttX official simulator was not suited for micro-ROS, and focused instead in QEMU. The master branch of QEMU's repository includes several MCU architectures, and also some other forks contained support

for other MCU vendors. The fork that the authors considered to be most relevant was QEMU_STM32, which contains the development of the Cortex-M3 cores for STM32 MCU family.

Having chosen QEMU_STM32, the authors were able to run NuttX for the STM32F103 part number inside the simulator, and were also able to run the micro-ROS client. After some tests regarding communication with micro-ROS agent, it was concluded that it was unstable, and the authors found the following limitations:

- Memory management is not done correctly by the simulation.
- UART buffer management is not done correctly, workaround is to decrease the baud-rate to the minimum.
- The option to connect a serial port of the MCU to a virtual port risks causing a overflow in the buffer when it is necessary to work with the system console. The alternative found be the authors was redirecting the serial port to a TCP server, which accepts external connections and provides stability to the system.

In a later deliverable (Flores Muñoz and Outerelo Gamarra 2019) the authors revisited the NuttX simulator and noticed some improvements regarding networking, but the lack of support for the C++ library still persists, which will make impossible to run micro-ROS applications using that API. The QEMU_STM32 did not receive significant improvements, but the authors tried to optimize micro-ROS usage and were able to obtain a slightly stability improvement, but there were still an unacceptable number of random crash situations.

Finally the authors abandoned the development of a simulator for micro-ROS, and concluded that despite QEMU_STM32 being the most promising one, the performance was weak and its development seemed abandoned because there was no relevant commits since May of 2019. So at that moment, there were no valid options that could fit the authors requirements.

2.2.2 Vehicular Cooperative Perception and Cooperative Simulation Ecosystems

This section delves into relevant literature in the autonomous vehicles and cooperative perception domain. It provides an insight on potential approaches for achieving V2X perception and highlights their benefits. Additionally, some of the examined works offer insights into co-simulation ecosystems capable of simulating cooperative scenarios in the autonomous vehicle realm. These ecosystems laid the groundwork for the simulation framework employed in this work.

AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles

The following work (Tsukada et al. 2020) consists in the development of a software named AutoC2X which enables cooperative perception by using OpenC2X for Autoware based autonomous vehicles.

Cooperative perception is achieved by sharing information about detected objects, such as vehicles and pedestrians, among neighboring nodes using the Autoware software and following ETSI ITS standards. The system is designed by combining two open-source software,

Autoware and OpenC2X, to enable cooperative perception, with the host node responsible for autonomous driving and the router handling the cooperative ITS function.

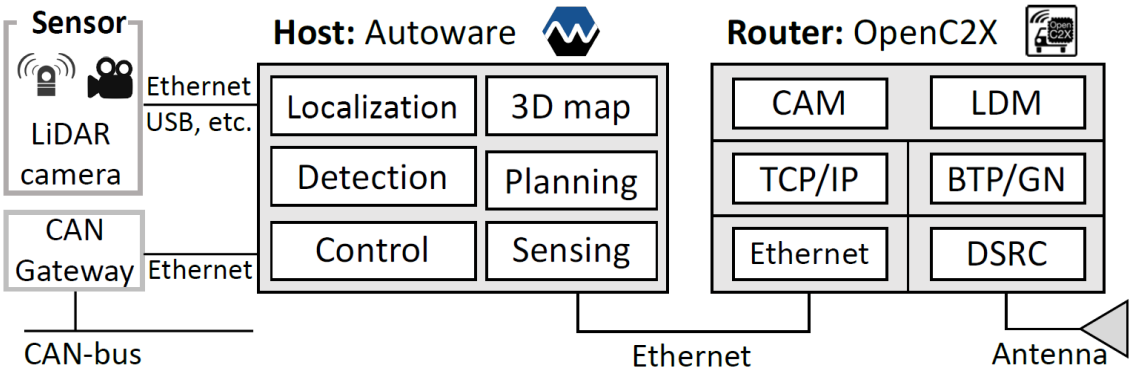


Figure 2.6: System model to integrate Autoware and OpenC2X. (Tsukada et al. 2020)

Autoware is responsible for sensing using 360-degree LiDAR scanners and cameras, as well as localization through scan matching between 3D point cloud maps and point cloud data from LiDAR scanners. The autonomous vehicle's actions such as acceleration, braking and steering is controlled via controller area network (CAN). OpenC2X is responsible for managing communications and handles almost the entire protocol stack in the ITS station architecture. Cooperative perception is enabled by sharing information through the use of cooperative awareness messages which are basic safety-related V2V (Vehicle-to-Vehicle) messages. The received presence information is stored in a local database called local dynamic map (LDM) which provides support for various ITS applications .

AutoC2X is developed using C++ by extending both Autoware (version 1.11.1) and OpenC2X (standalone version 1.5). When the system is initiated, Autoware reads the 3D maps and begins receiving sensor data. The point cloud data from the LiDAR is used to localize the ego vehicle and detect neighboring objects. The position of the ego vehicle and the objects it detects are converted from a local coordinate system to a global coordinate system and transmitted to the OpenC2X router via TCP/IP over Ethernet.

For cooperative perception, Autoware detects the surrounding objects, and after the sensor fusion, the information of the detected objects is published in the topic of "/detection/objects". The coordinates are then transformed and sent to OpenC2X who adds the received detected object information to a queue and encodes the latest information in the queue into a proxy CAM. This information includes timestamp, latitude, longitude, speed, and ITS station ID.

To assess the proposed approach, the authors examined a scenario where an RSU positioned at an intersection disseminates cooperative perception data, referred to as proxy CAM, to nearby vehicles. The evaluation employed packet delivery ratio and total delay as the key performance metrics. Based on the results, the authors concluded that in the most adverse scenario, autonomous vehicles could receive the cooperative perception message in approximately 100 ms. The point cloud measurement operated at a frequency of 10 Hz, which also represented the maximum rate for the CAM. Consequently, it is reasonable to assert that the proposed system can effectively transmit the message within the point cloud measurement interval and within the CAM interval.

V2X-Sim: Multi-Agent Collaborative Perception Dataset and Benchmark for Autonomous Driving

This study (Li et al. 2022) delves into a V2X simulation methodology designed to investigate collaborative perception for autonomous driving. It leverages multi-agent sensor data from a road-side unit and multiple vehicles to enable collaborative perception. The system incorporates multi-modality sensor streams to facilitate comprehensive perceptual analysis and diverse ground truths to support various perception tasks.

The created dataset harnesses the capabilities of SUMO, a micro-traffic simulation to generate highly realistic traffic scenarios. Simultaneously, CARLA, a renowned open-source simulator for autonomous driving research, is employed to capture meticulously synchronized sensor data from multiple vehicles and the RSU. This dataset encompasses multi-modal sensor data from various entities, enabling cross-modality perception. Furthermore, it includes a rich array of annotations such as bounding boxes, vehicle trajectories, and semantic labels, facilitating a wide range of subsequent tasks.

To ensure the dataset's comprehensiveness and robust perception capabilities, each vehicle and the RSU were outfitted with a diverse sensor suite. This suite included RGB cameras, LiDAR, GPS, IMU, and the RSU featured both RGB cameras and LiDAR sensors. The camera and LiDAR sensors on both the vehicles and RSU were strategically positioned to provide a full 360-degree horizontal coverage, enabling comprehensive perception. Each vehicle was equipped with six RGB cameras, adhering to the nuScenes configuration, while the RSU boasted four RGB cameras, each oriented towards one of the crossroad's directions. In CARLA, additional sensors such as depth cameras, semantic segmentation cameras, semantic LiDAR, and BEV (Bird's Eye View) semantic segmentation cameras were employed to acquire corresponding ground-truth data for each RGB camera.

Leveraging the proposed dataset, the authors performed benchmarking evaluations of several advanced collaborative perception methods. These assessments covered collaborative BEV detection, tracking, and semantic segmentation tasks. In terms of future research directions, the authors are considering simulating latency challenges and devising new evaluation metrics tailored to collaborative perception tasks.

Cooperative Task Execution for Object Detection in Edge Computing: An Internet of Things Application

The article that follows (Amanatidis et al. 2023), researched a task execution system for cooperative object detection in edge computing. This approach utilized Single Board Computers (SBC) with high performance and low power consumption, together with deep learning models such as You Only Look Once (YOLO) which is a well-known object detector model. The proposed edge computing topology consisted of an edge server, a Wi-Fi access point, and three end devices, which were Raspberry Pi´s. With this approach, the objective of this work was to investigate the concept of offloading the processing of the workload on multiple end devices to achieve faster processing and high-accuracy object detection.

The results of the evaluation proved that, with the optimal configuration of the parameters used in the system, offloading the workload on multiple end devices can improve the performance to respond to real-time conditions without relying on either a server or cloud resources. Given the results, it can be concluded that extending the perception closer to the environment by distributing several end devices capable of object detection, is a viable option for increasing detection accuracy and reducing response time in real-time conditions.

COPADRIVe - A Realistic Simulation Framework for Cooperative Autonomous Driving Applications

COPADRIVe (Vieira et al. 2019) is a realistic co-simulation framework that integrates Gazebo, which is an advanced robotics simulator, with the OMNeT++ network simulator, over the ROS framework, while supporting the simulation of advanced cooperative applications such as platooning in realistic scenarios.

COPADRIVe simulation framework was built over the Veins simulator, a framework for running vehicular network simulations, and the Vanetza communications stack, which is a implementation of ETSI C-ITS protocol suite. For the integration of OMNeT++ with Gazebo, it is used ROS publish/subscribe mechanisms.

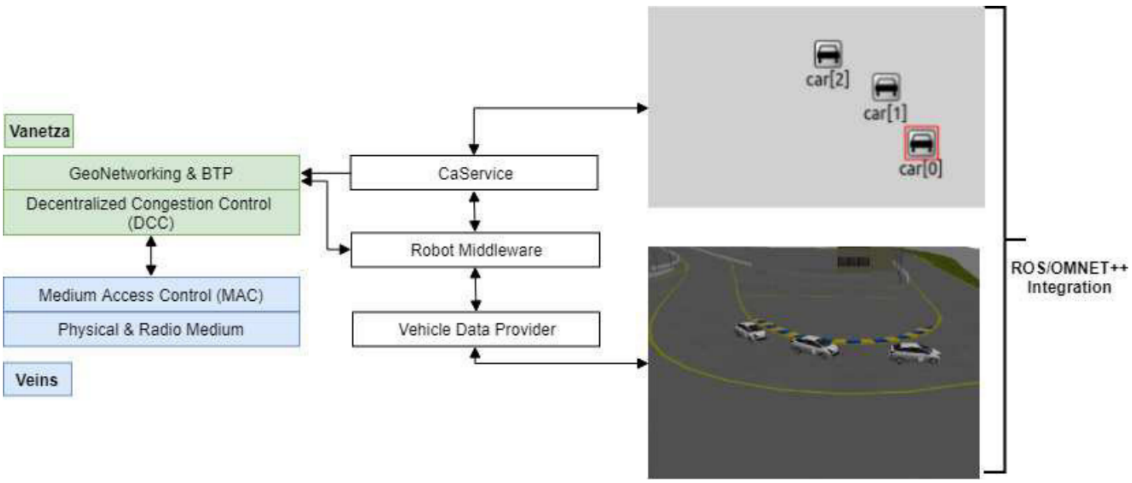


Figure 2.7: COPADRIVe Framework Architecture. (Vieira et al. 2019)

Utilizing COPADRIVe as a preliminary proof-of-concept, the authors implemented and validated various scenarios to assess the behavior of a CoVP control model solely reliant on CAM exchanges. They scrutinized the impact of different CAM exchange frequencies and ITS-G5 basic system profile (BSP) recommendations to validate the accuracy of the simulation framework. COPADRIVe effectively facilitated this analysis within a rich and realistic simulation environment, encompassing both control and communication aspects.

COPADRIVe's pioneering capability to simulate communications proved invaluable, enabling an in-depth examination of latency in V2V communications—a critical factor in validating the proposed scenarios. This groundwork laid the foundation for the subsequent related work named AuNa, which expanded upon its features to enhance its capabilities.

AuNa - Modularly Integrated Simulation Framework for Cooperative Autonomous Navigation

AuNa is a modularly integrated simulation framework for cooperative autonomous navigation. The main objective of the project that created AuNa was to integrate the simulation tools of robotics, communication and control namely ROS2, OMNeT++, and MATLAB to simulate cooperative driving scenarios (Teper et al. 2022). AuNA is based in COPADRIVe, as its framework is very similar, but with some improvements, it addresses some of its issues such as the efficiency in the synchronization of the robot simulation and the communication

simulation, which addresses flexibility and event management and synchronization. Also unlike COPADRIVe, AuNa is based on ROS2, and the architecture of Artery is not adjusted to only support the platooning scenario. Its framework is composed as follows:

- Robot Simulation: the simulation environment should allow multiple robots to be spawned dynamically, with each robot having a unique name or index. For each robot, a connection is established via ROS2 nodes, which allows the control of the robots and publishes sensor data. For the robot simulation, Gazebo provides plugins for implementing sensors and robot controls. By adding a namespace to the Gazebo plugins, each robot is given a unique name or index, so that each node and topic corresponds to its own robot.
- Communication Simulation: for the communication architecture, the framework uses OMNeT++ and Artery. In order to synchronize OMNeT++ and the other simulations using ROS2 nodes, it was implemented a custom scheduler within OMNeT++ that runs the execution of OMNeT++ events, the synchronization between OMNeT++ and Gazebo, as well as the updates between the navigation system in ROS2 and communication module in OMNeT++ of each robot.
- Control Simulation: MATLAB and Simulink provide a complete feature set for designing control systems of any complexity. The 2022 release of MATLAB and Simulink provides ROS2 support with the ROS Toolbox, and thus, instead of implementing the controllers directly into the code, the authors opted to integrated MATLAB and Simulink into the framework.
- Navigation Systems: it is required a navigation system in order for the robots to autonomously reach a target destination. For this purpose the authors considered using the Nav2 package in ROS2 which provides a collection of algorithms for localization, mapping, and navigation that are designed for dynamic environments (Teper et al. 2022).

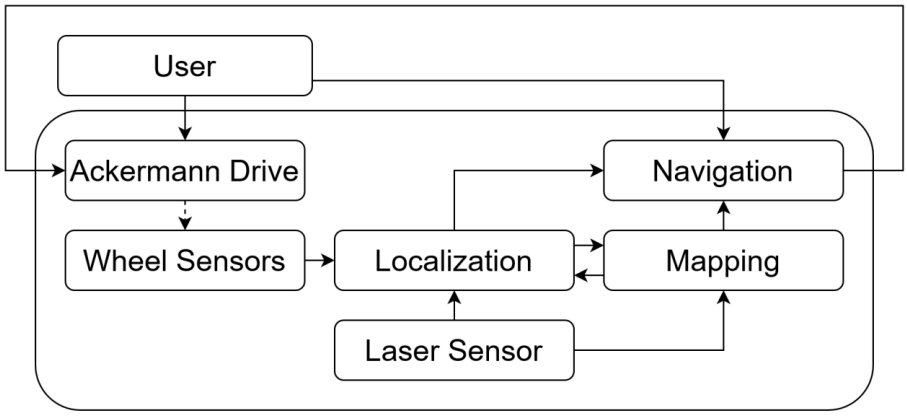


Figure 2.8: Robot navigation system overview. (Teper et al. 2022)

The default implemented scenario in AuNa is a platooning scenario under cooperative adaptive cruise control (CACC), which is simulated in a small racetrack environment. Every robot comes equipped with wheel sensors for motion tracking and a LiDAR sensor for environmental perception. These robots are controlled using an Ackermann-drive system, necessitating input parameters for velocity and steering angle during navigation. The functionality of the wheel sensors, LiDAR, and the drive system is realized through ROS2 Gazebo plugins.

The environment was mapped using SLAM, and the localization method used for each robot is AMCL. The navigation system employs a grid map to depict the environment, employing a grid of cells to denote both occupied and unoccupied areas. Throughout navigation, each robot continuously monitors its position using the odometry data from its Ackermann-drive.

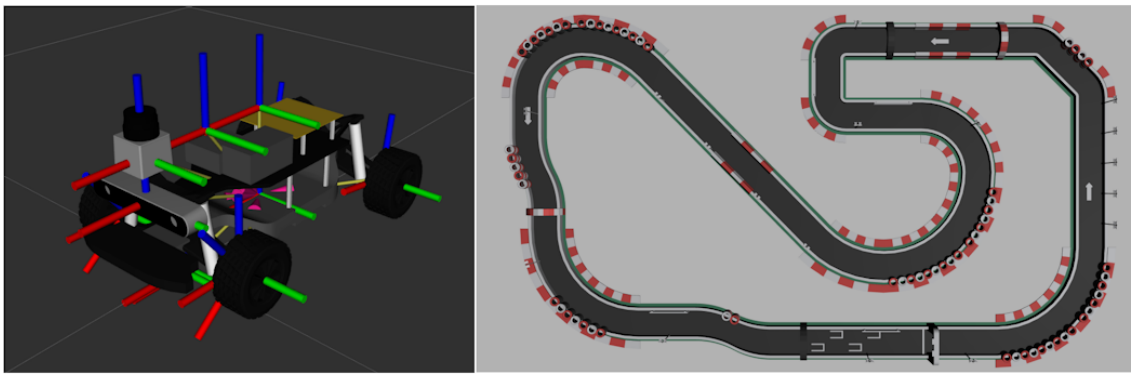


Figure 2.9: Robot model with link frames and simulated racetrack. (Teper et al. 2022)

The resulting robot can autonomously navigate dynamic environments and offers the necessary data to facilitate communication with other robots and the implementation of control systems for various scenarios.

Regarding the communications simulation evaluation, the authors analysed the CAM transmission delay. Compared to an ideal ROS2 topic connection, the transmission of CAMs introduces consistent delays ranging from 0.1 to 0.2 seconds. Two primary factors contribute to this delay, the processing time for transmitting and receiving CAMs and the transmission delay between vehicles. Additionally, CAMs adhere to the ETSI ITS-G5 standard, dictating a transmission interval between 0.1 and 1 second, resulting in lower data frequency. The decreased data resolution is attributed to CAMs using a 16-bit integer format, unlike the 64-bit float employed by ROS2 topics. Despite these effects, the framework successfully integrates ROS2 and Artery for simulating robots and vehicle communication. These delays and reduced data resolution may influence other aspects of the robot system, underscoring the need to apply ROS2 and OMNeT++ capabilities for system implementation and interaction evaluation.

2.3 Critique Analysis

This section performs a critique analysis regarding the mentioned approaches, particularly by reviewing which of them are most relevant for this thesis.

Starting with the OFERA project (OFERA 2020), its motivation is aligned with this thesis, as it aims to contribute to bridge the gap between high-performance devices and microcontrollers, in order to allow cooperation between them. The robotic framework developed in this work, micro-ROS, will be essential for the development of this work. The smart warehouse use-case (Kołcon, Maciaś, and Malki 2019) is a fine example of how micro-ROS can be used in the development of a smart environment using microcontrollers as the main component. The technologies used proved to be a good example to follow for the approach of this thesis. The OFERA work regarding micro-ROS default simulation environment (Flores Muñoz and Outerelo Gamarra 2019) provided an insight in the state of the micro-ROS

simulation. The approach studied by the authors appeared to be a complete solution capable of simulating a microcontroller architecture and a RTOS, but unfortunately, the authors were unable to obtain a stable simulation, leading them to abandon the research.

The work regarding AutoC2X (Tsukada et al. 2020) proves to be similar in terms of the desired approach for achieving cooperative perception, and has an interesting method of handling the communications.

The article centered around cooperative task execution for object detection in edge computing (Amanatidis et al. 2023) has offered valuable insights into the benefits of expanding perception capabilities through IoT devices situated closer to the environment. The findings strongly indicate that this approach is not only viable but also effective in enhancing detection accuracy and decreasing response time, particularly within real-time conditions.

The simulation approach introduced by the authors in the AuNa project (Teper et al. 2022) is mostly related to the objectives of this thesis regarding the development of a simulation environment and coincide with most of the needs for simulation regarding this thesis. The framework introduced by AuNa will be the main reference for the simulation framework of this work.

Chapter 3

Technologies

This chapter introduces the key technologies essential for the framework's development. These technologies encompass robotics frameworks, forming the system's core, communication technologies responsible for data transport, and simulation and emulation technologies that enable us to implement the explored concepts and technologies without relying on physical hardware. Additionally, we conduct a critical analysis comparing these technologies and providing individual justifications for their selection.

3.1 Robotics Frameworks

Robotics frameworks play a pivotal role in facilitating software development for robots and robotic systems. They serve as comprehensive toolsets enabling the seamless integration of various system components, including sensors, actuators, and robot control. This section provides an overview of two widely utilized robotics frameworks: ROS2, a common choice for general robotics applications, and micro-ROS, its counterpart tailored for embedded systems.

3.1.1 ROS2

ROS2 is the latest version of ROS (Robot Operating System), it is an open source software development kit for robotics applications (Steven Macenski et al. 2022). At its core, ROS provides a message-passing system, often called "middleware" or "plumbing". Communication is one of the first needs to arise when implementing a new robot application, or really any software system that will interact with hardware. ROS's built-in and well-tested messaging system saves time by managing the details of communication between distributed nodes via an anonymous publish/subscribe pattern (Robotics 2021).

ROS breaks complex systems down into many modular nodes, each node should be responsible for a single, module purpose (e.g. one node for controlling wheel motors, one node for controlling a laser range-finder, etc). Each node can send and receive data to other nodes via topics, services, actions, or parameters (Open Robotics 2022). It introduces several key concepts that help manage the complexity of robotic applications and facilitate inter-process communication. The following are some of the essential ROS 2 concepts (Open Robotics 2023):

- **Nodes** - Nodes are individual processes in a ROS 2 system. Each node is an independent unit responsible for performing a specific task, such as controlling a sensor or executing a specific algorithm. Nodes can communicate with each other by publishing and subscribing to topics, and also by providing or using services and actions.

- **Topics** - Topics are named communication channels through which nodes exchange data. A node can publish data to a topic, and other nodes can subscribe to that topic to receive the published data. Topics allow for loosely coupled communication between nodes.
- **Messages** - Messages are the data structures used to exchange information on ROS 2 topics. Each topic has a specific message type, and nodes must use compatible message types to communicate effectively.
- **Services** - Services provide a request-response communication pattern between nodes. A node can offer a service, and other nodes can send requests to that service. The service node processes the request and returns a response.
- **Actions** - Actions extend the service concept to support asynchronous communication, allowing for more complex and time-consuming tasks to be executed in a non-blocking manner.
- **Parameters** - Parameters are a way to store configuration data that can be dynamically updated during runtime. Nodes can access and modify parameters to adjust their behavior without the need for restarting.
- **Bags** - Bags are a format for storing and playing back ROS 2 data, making it convenient to record and replay data for analysis and debugging purposes.
- **Launch System** - The launch system in ROS 2 is used to start and manage multiple nodes and configure their parameters. It allows for easier management of complex robot systems with multiple nodes.
- **Discovery** - Enables nodes to automatically find and establish communication with each other without the need for manual configuration. When nodes start up in a ROS 2 system, they use the discovery mechanism to identify other nodes and determine how to communicate with them.

These concepts provide a solid foundation for building modular, distributed, and scalable robotic systems using ROS 2. By understanding and leveraging these core concepts, developers can design complex robotic applications in a well-organized and efficient manner.

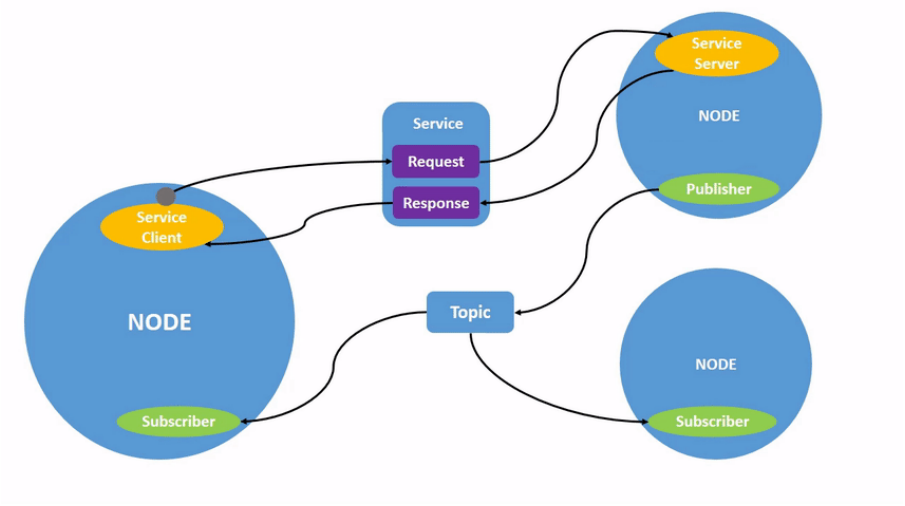


Figure 3.1: ROS2 nodes, topics and services. (Open Robotics 2022)

Regarding ROS2 development, ROS client libraries facilitate communication between nodes developed in various programming languages. The core ROS client library (RCL) plays a vital role in implementing shared functionalities required for different ROS APIs across languages. This approach simplifies the development of language-specific client libraries and ensures more consistent behavior among them. The ROS2 team currently maintains two client libraries: rclcpp, a C++ client library, and rclpy, a Python client library. ROS2 also includes **RVIZ**, an essential tool for 3D visualization and data debugging within the developed robot.

3.1.2 micro-ROS

micro-ROS is a framework for enabling ROS2 in microcontrollers. Microcontrollers are important because they are used in almost every robotic product. Typical reasons are:

- Hardware access.
- Hard, low-latency real-time.
- Power saving.

Some of its key features are, supporting all major ROS concepts, following the ROS2 architecture, and multi-RTOS (Real-Time Operating System) support (micro-ROS 2023a).

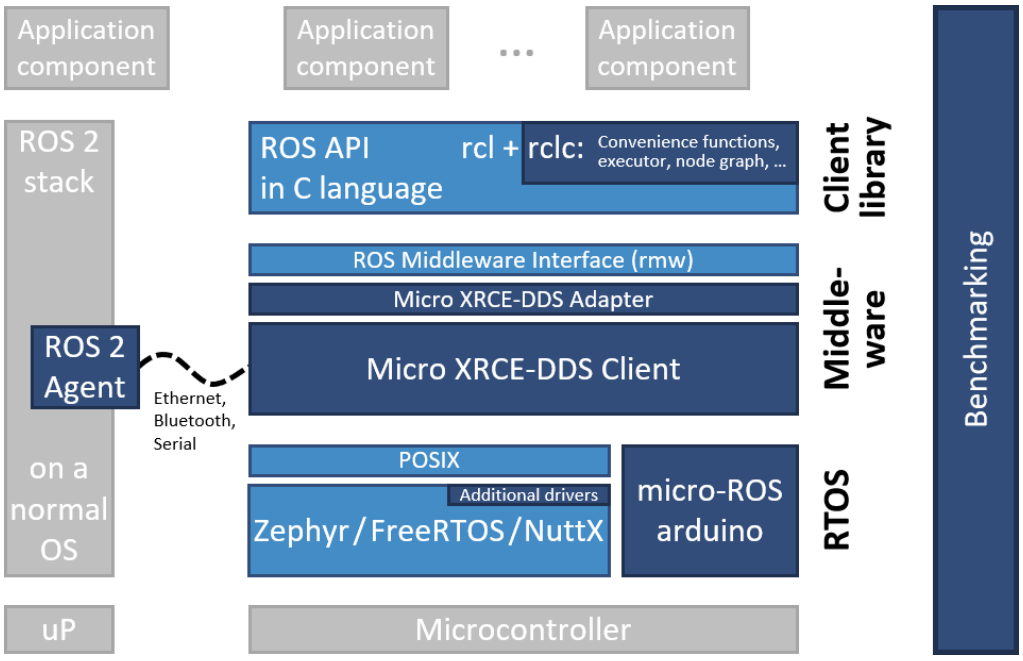


Figure 3.2: micro-ROS architecture. (micro-ROS 2023a)

micro-ROS not only follows the ROS 2 architecture but also takes advantage of its middleware pluggability to implement **DDS-XRCE**, a communication protocol specially designed and optimized for microcontrollers. DDS-XRCE is a protocol developed by the Object Management Group (OMG) as an extension of the traditional Data Distribution Service (DDS) standard. It is specifically designed to enable efficient and reliable communication between resource-constrained devices, such as microcontrollers and other embedded systems, and more powerful devices in distributed systems (Object Management Group et al. 2020).

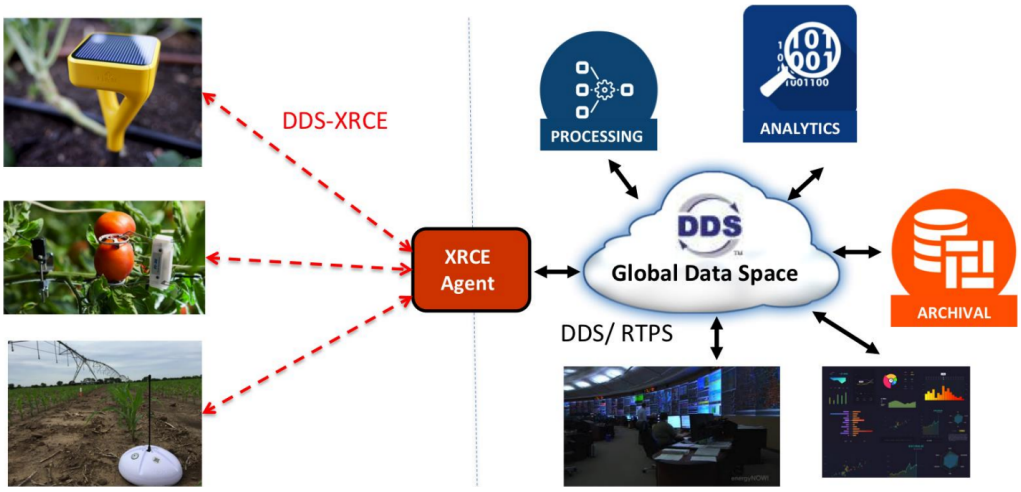


Figure 3.3: Scope of XRCE Protocol. (Object Management Group et al. 2020)

Some of the key features of DDS-XRCE include:

- **Lightweight and Efficient** - DDS-XRCE is designed to have a small code footprint and low overhead, making it suitable for resource-constrained devices with limited processing power and memory.
- **Data-Centric Communication** - Like DDS, DDS-XRCE adopts a data-centric communication model, allowing devices to exchange data based on the content and type of data rather than explicit addressing.
- **Scalability** - DDS-XRCE supports scalability, enabling communication in both small-scale and large-scale systems with various data distribution patterns.
- **Extensible and Interoperable** - The protocol is extensible, allowing developers to define custom data types and adapt it to different application scenarios. It is also designed to be interoperable with other DDS implementations.
- **Quality of Service Support** - DDS-XRCE supports a range of Quality of Service settings, allowing applications to define the reliability, priority, and timing requirements of data exchanges.
- **Real-Time Communication** - DDS-XRCE is suitable for real-time applications, ensuring that data is delivered within specified time constraints.

micro-ROS extends the fundamental concepts of ROS 2, encompassing nodes, publish/subscribe, client/service, node graph, lifecycle, and more, to be compatible with microcontrollers (MCUs). The micro-ROS client API, developed in the C programming language, is built upon the standard ROS 2 Client Support Library (rcl) and incorporates a set of extensions and convenient functions (rclc). This combination of rcl and rclc is specially optimized for MCUs, enabling it to operate without the need for dynamic memory allocations after an initialization phase. Moreover, the rclc package offers sophisticated execution mechanisms, allowing the implementation of well-established scheduling patterns commonly utilized in embedded systems engineering (micro-ROS 2023d).

micro-ROS is specifically tailored to support mid-range 32-bits microcontroller families as its main hardware targets. To successfully run micro-ROS on embedded platforms, meeting the

Table 3.1: ROS2 Feature Comparison. (micro-ROS 2023g)

ROS 2 Feature	**Availability in micro-ROS**
Publish/subscribe over topics	Available
Clients and services	Available
ROS 1 – ROS 2 communication bridge	Available
Actions	Available
Parameters	Available with some WIP features
Node Graph	Available
Discovery	Available
Shared API for inter- and intra-process comms	Available with some WIP features
Extensible cmd-line introspection tools	Available
Launch system for coordinating multiple nodes	Currently unavailable
Namespace support for nodes and topics	Available
Static remapping of ROS names	Further investigation required
Support of rate and sleep with system clock	Further investigation required
Support for simulation time	Further investigation required

memory constraints is crucial. In general, micro-ROS requires MCUs with tens of kilobytes of RAM memory and communication peripherals that enable seamless micro-ROS Client to Agent communication (micro-ROS 2023h).

Besides micro-ROS, there are other similar approaches for enabling communication between microcontrollers and ROS systems. ROSSerial is a communication protocol that encapsulates standard ROS serialized messages and allows multiple topics and services to be multiplexed over a device, such as a serial port or network socket. ROSSerial client libraries simplify the process of deploying ROS nodes on different systems. These libraries are adaptations of the rosserial_client library, written in general ANSI C++. Some of its limitations include a restriction on the number of Publishers and Subscribers, which is capped at 25. Additionally, the size of serialization and deserialization buffers is limited to 512 bytes by default for rosserial_client. Another limitation is that ROSSerial is exclusively designed for ROS1 and does not support ROS2. Another similar approach to micro-ROS is RIOT-ROS2, which is a customized version of the primary ROS 2 stack, tailored to run on microcontrollers with the support of the RIOT Operating System. This approach modifies several ROS2 layers to make it able to run on a microcontroller (micro-ROS 2023b).

In Table 3.2, it is compared the 3 approaches, and it is notable that micro-ROS stands out for its unique architecture, providing a full ROS 2 implementation on microcontrollers and enabling seamless communication with the ROS 2 network. In conclusion, for users seeking a comprehensive ROS 2 implementation on microcontrollers with efficient communication, micro-ROS would be the preferred option.

3.2 Communications technologies

The communications technologies are an important factor for the exchange of of information and data between various components of the system. This data could include sensor readings, control commands, status updates, and more. Efficient and reliable data exchange is essential for the system to function correctly. In this section it will be overviewed two

Table 3.2: micro-ROS Related Approaches Comparison. (micro-ROS 2023b)

	ROSSerial	**RIOT-ROS2**	**micro-ROS**
OS	bare-metal	RIOT	NuttX, FreeRTOS and Zephyr
Communications architecture	Bridged	N/A	Bridged
Message format	ROS1	N/A	CDR (from DDS)
Communication links	UART	UART	UART, SPI, IP (UDP), 6LowPAN
Communication protocol	Custom	NDN	XRCE-DDS (or any rmw implementation)
Code Base	Independent implementation	Standard ROS 2 stack up to RCL	Standard ROS 2 stack up to RCL (RCLCPP coming)
Node API	Custom rosserial API	RCL,RCLC	RCL (soon RCLCPP)
Callback execution	Sequential, in order of messages	N/A	Choice of ROS 2 executors or MCU optimized executors
Timers	Not included	Not included	Normal ROS 2 timers
Time sync to host	Custom	N/A	NTP/PTP
Lifecycle	Not supported	Partial	Partial, full coming

communications technologies: 6LoWPAN, which was used in the previously mentioned work regarding the smart warehouse use-case, and is commonly employed in micro-ROS and IoT applications. Additionally, we will discuss ETSI ITS-G5, a communication standard commonly applied in intelligent transportation systems.

3.2.1 6LoWPAN

6LoWPAN, is an open standard defined in RFC 6282 by the Internet Engineering Task Force (IETF). It enables the use of IPv6 over IEEE 802.15.4 low-power wireless networks, which means that low-power devices with limited processing capabilities have their own IPv6 address, and are able to connect directly with the Internet using open standards. 6LoWPAN characteristics make this technology ideal for markets such as home automation with sensors and actuators, street light monitoring and control, residential lighting, smart metering and generic IoT applications with Internet connected devices (Olsson 2014).

6LoWPAN networks communicate natively with IP, making it simple to connect them to other networks using IP routers. They are often positioned on the network edge, functioning as stub networks where incoming data is destined for devices within the 6LoWPAN. Multiple 6LoWPAN networks can be connected to other IP networks through one or more edge routers that forward IP datagrams across different media like Ethernet, Wi-Fi, or 3G/4G. Additionally, edge routers may support IPv6 transition mechanisms, such as NAT64 defined in RFC 6146, to establish connections between 6LoWPAN networks and IPv4 networks without requiring the 6LoWPAN nodes to implement IPv4 entirely.

The use of edge routers in 6LoWPAN networks offers several advantages over other network architectures, such as ZigBee, Z-wave, Bluetooth, or proprietary networks. Unlike those networks, edge routers do not maintain any application-layer state, which eliminates the need for complex and stateful application gateways to connect to IP-based networks like the Internet. These gateways would otherwise require a deep understanding of various application profiles used in the network, making them more demanding in terms of processing power and complexity. However, IP-based border routers, like edge routers, remain agnostic to the specific application protocols employed in 6LoWPAN networks. This simplifies the edge router's design, allowing the use of lower-cost embedded devices with simpler software and less complex hardware. Despite the simplicity, IP architecture does not rule out the use of proxies and caches to optimize network performance, which are commonly employed in the Internet today.

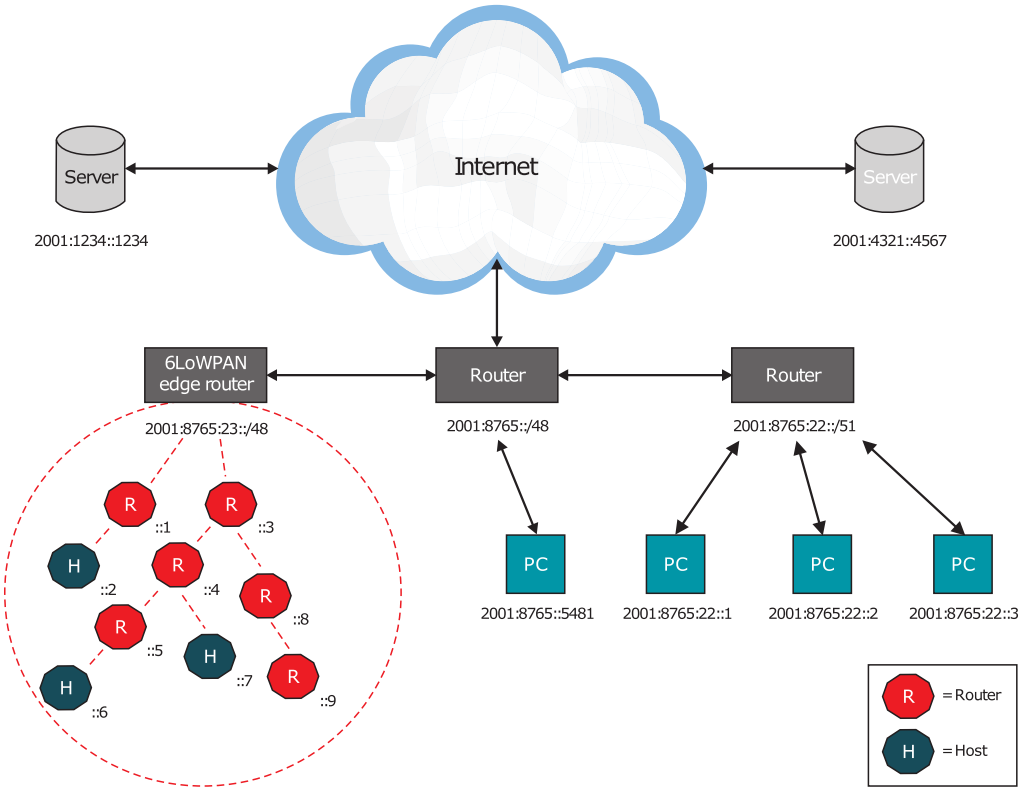


Figure 3.4: An example of an IPv6 network with a 6LoWPAN mesh network. (Olsson 2014)

A typical 6LoWPAN network consists of two additional device types: routers and hosts. As the name suggests, routers are responsible for routing data to other nodes within the 6LoWPAN network. On the other hand, hosts, also known as end devices, do not have the capability to route data to other devices in the network. Some hosts can operate as "sleepy devices," waking up periodically to check their parent (a router) for data, which allows for very low power consumption.

3.2.2 ETSI ITS-G5

ETSI ITS-G5 is a set of standards developed by the European Telecommunications Standards Institute (ETSI) for intelligent transportation systems using the Dedicated Short Range

Communications (DSRC) technology. ITS-G5 is also known as C-ITS in Europe.

The main goal of ETSI ITS-G5 is to enable communication between vehicles (V2V), between vehicles and infrastructure (V2I), and between vehicles and pedestrians (V2P). It establishes the foundation for Cooperative Intelligent Transport Systems, where vehicles and infrastructure can exchange information to improve road safety, traffic efficiency, and overall transportation experience. The communication in ETSI ITS-G5 is based on the IEEE 802.11p standard, which is an amendment to the Wi-Fi standard specifically tailored for vehicular communication in the 5.9 GHz frequency band. This allows for low-latency and reliable communication in a fast-moving vehicular environment. ITS-G5 facilitates communication in diverse environments, encompassing rural, urban, suburban, and highway settings, with an impressive operational range of up to 1000 meters. This robust communication standard accommodates vehicles moving at relative speeds of up to 110 km/h, making it suitable for various transportation scenarios (Altinel, Wollenschläger, and Hein 2019).

The ETSI ITS-G5 standards define protocols, data formats, and communication methods to support various C-ITS applications, such as collision avoidance, traffic signal phase and timing, roadworks warning, and more. The technology has gained significant attention and adoption in Europe, with real-world deployments and pilot projects to showcase the potential benefits of connected vehicles and infrastructure for safer and more efficient transportation systems.

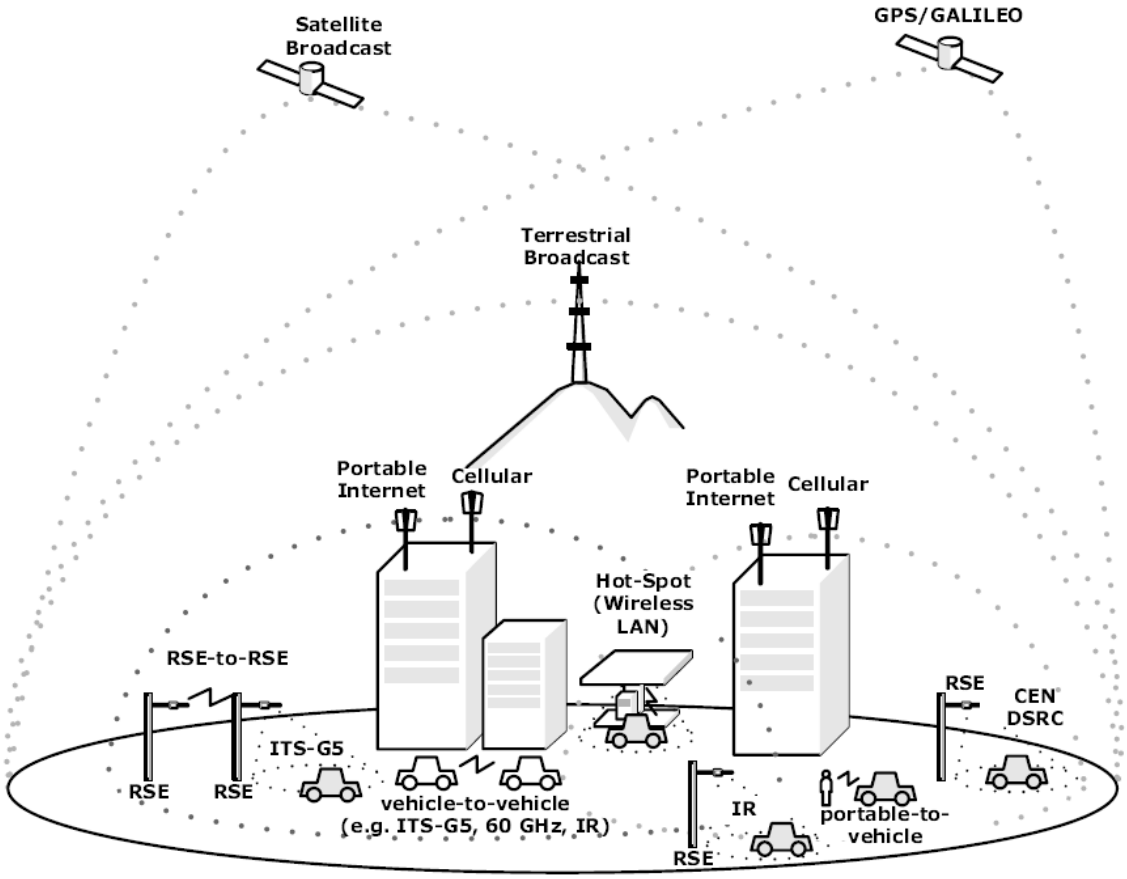


Figure 3.5: C-ITS Scenario. (European Telecommunications Standards Institute 2010)

Cooperative awareness in road traffic involves exchanging information about the position, dynamics, and attributes of road users and roadside infrastructure. Road users include vehicles, bicycles, pedestrians, and roadside infrastructure such as road signs and traffic lights. This awareness forms the foundation for various road safety and traffic efficiency applications, as detailed in ETSI TR 102 638. The exchange of information occurs through wireless V2X networks, facilitating communication between vehicles (V2V), between vehicles and infrastructure (V2I and I2V), and is an integral part of ITS. The crucial information for cooperative awareness is packaged into the periodically transmitted Cooperative Awareness Message (CAM).

Cooperative Awareness Messages (CAMs) play a crucial role in the ITS network, facilitating communication and awareness among Intelligent Transport Systems (ITS-Ss). These messages are exchanged between ITS-Ss, enabling vehicles to establish and maintain awareness of each other on the road network. CAMs contain vital status and attribute information related to the originating ITS-S. For vehicle ITS-Ss, the status information includes time, position, motion state, activated systems, etc., while the attribute information covers dimensions, vehicle type, role in road traffic, etc. By receiving a CAM, an ITS-S gains awareness of the presence, type, and status of the originating ITS-S. This received information is utilized to support various ITS applications. For instance, a receiving ITS-S can estimate collision risk by comparing its status with the originating ITS-S, and if necessary, inform the vehicle's driver through the Human-Machine Interface (HMI).

The periodic transmission of CAMs emphasizes the crucial timing aspect, especially considering the high-quality requirements for the Applications relying on such messages. To meet ETSI's specifications for various ITS scenarios in the Basic Service Profile, the CAM generation service must adhere to specific generation rules. These rules ensure that CAMs satisfy the necessary criteria and perform effectively across a wide range of ITS applications. As an example, the CAM generation interval is specified to be no less than 100 ms, equivalent to a CAM generation rate of 10 Hz. Conversely, it should not exceed 1000 ms, corresponding to a CAM generation rate of 1 Hz. These specific intervals are defined to ensure a proper balance between the frequency of CAM updates and the efficient utilization of network resources (European Telecommunications Standards Institute 2014).

3.3 Emulation and Simulation tools

The emulation and simulation tools play a pivotal role in the development of the system. These tools are of utmost importance as they allow us to comprehensively explore and test various concepts and technologies without the constraints of physical hardware. Through emulation and simulation, we can replicate real-world scenarios, evaluate system performance, and fine-tune algorithms, all within a controlled and efficient virtual environment. The tools under examination will facilitate the emulation of micro-ROS, simulate the ETSI ITS-G5 communications standard discussed earlier, and create simulations encompassing the vehicle, its intricate systems, sensors, and the surrounding environment.

3.3.1 QEMU

QEMU is a generic and open source machine emulator and virtualizer. Its most common use is for system emulation where it provides a virtual model of an entire machine (CPU, memory and emulated devices) to run a guest OS (The QEMU Project Developers 2022).

As previously seen in the work by (Flores Muñoz and Muguruza Goenaga 2019), QEMU is capable of simulating MCU architectures such as the STM32 ARM architecture.

3.3.2 Zephyr Emulator

The Zephyr Emulator is a feature in the Zephyr Project, which is an open-source real-time operating system (RTOS) designed for resource-constrained devices such as microcontrollers and IoT devices. The Zephyr Emulator allows developers to run and test Zephyr applications on a host machine (e.g., a PC) instead of the target hardware. Emulators serve the purpose of simulating hardware devices, facilitating the testing of different subsystems. For instance, an emulator can be designed to mimic an I2C compass, creating a virtual representation of the device on the I2C bus, and enabling it to be utilized in a manner similar to an actual hardware component (Zephyr Project 2023).

The Zephyr Emulator provides a simulated environment that mimics the behavior of the target hardware, allowing developers to validate their code, perform debugging, and test various functionalities without the need for actual hardware. This is particularly useful during the early stages of development when the hardware might not be readily available or to speed up the development cycle.

3.3.3 OMNeT++

OMNeT++ is a simulation library and framework primarily for building network simulators. It includes wired and wireless communication networks, on-chip networks, queueing networks, etc. Its main components are the following (OpenSim Ltd. 2019):

- Simulation kernel library (C++).
- The NED topology description language.
- Simulation IDE based on the Eclipse platform.
- Interactive simulation runtime GUI (Qtenv).
- Command-line interface for simulation execution (Cmdenv).
- Utilities (makefile creation tool, etc.).
- Documentation, sample simulations, etc.

Its simulation framework is discrete event driven, it provides several essential concepts to facilitate the simulation process:

- **Modules** - Modules are the building blocks of an OMNeT++ simulation. They represent entities in the system being modeled and can encapsulate behavior, state, and communication capabilities. Modules can communicate with each other through message passing.
- **Message** - Messages are used to exchange information between modules. They contain data and are sent from one module to another. In OMNeT++, messages are scheduled to be delivered at a specific time, enabling discrete event simulation.
- **Channels** - Channels define the communication paths between modules. They determine how messages are transmitted and the delay between sending and receiving a message.

- **Events** - OMNeT++ operates on discrete events, where each event represents a change in the system state at a specific point in time. Events are processed in chronological order, allowing the simulation to progress in discrete steps.
- **Simulation** - Simulations in OMNeT++ are run by scheduling events and processing them in the correct order. During simulation, events are processed until the simulation's end condition is met.
- **Networks** - OMNeT++ is widely used for simulating communication networks. It provides built-in support for network modeling, including protocols, routers, and traffic generation.
- **NED Files** - The Network Description (NED) files define the simulation model's structure. They specify the modules, connections, and parameters used in the simulation.
- **Result Recording** - OMNeT++ allows recording simulation results, including statistics, messages, and events. This recorded data can be analyzed and visualized later to understand the system's behavior.
- **Configurations** - Different configurations can be used to vary simulation parameters and study system behavior under various conditions.
- **GUI Environment** - OMNeT++ provides an integrated graphical user interface (GUI) that allows users to create, configure, and visualize simulations easily.

These concepts collectively form the foundation for building complex simulations in OMNeT++ and enable researchers and developers to study and analyze a wide range of systems, including communication networks, distributed systems, and other complex dynamic systems.

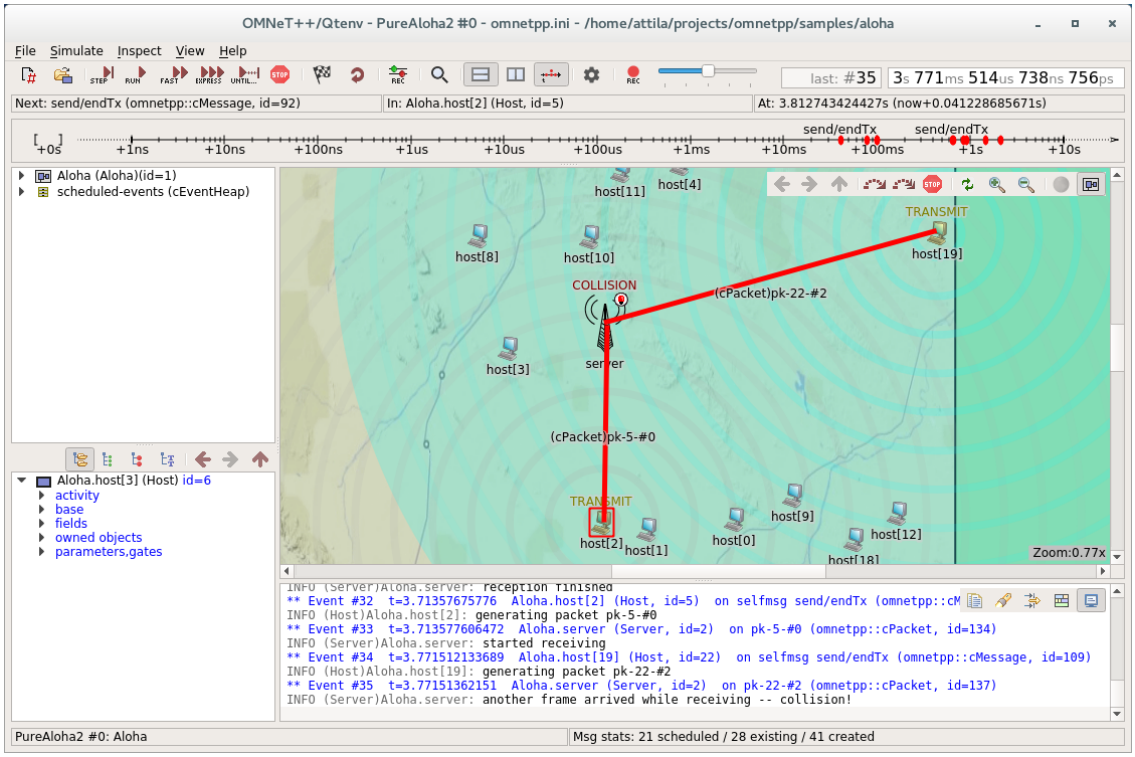


Figure 3.6: OMNeT++ GUI. (OpenSim Ltd. 2019)

3.3.4 ns-3

ns-3 (nsnam 2023) stands as a discrete-event network simulator with its primary focus on research and educational applications. This software is freely available under the GNU GPLv2 license, accessible for research, development, and practical use. The ns-3 project's objective is to establish a leading open simulation platform for networking research. It aims to align with the contemporary requirements of networking research simulations, fostering community engagement, peer assessment, and software validation.

The ns-3 software framework promotes the creation of simulation models that offer a high level of realism, enabling ns-3 to function as a real-time network emulator closely connected to the actual environment. This design also facilitates the incorporation of numerous real-world protocol implementations into ns-3. The foundational simulation core of ns-3 accommodates research pertaining to both IP and non-IP networks. However, the majority of its user base primarily engages with wireless/IP simulations, encompassing models for Wi-Fi, WiMAX, or LTE at layers 1 and 2, and an array of static or dynamic routing protocols like OLSR and AODV for IP-based scenarios. ns-3 also includes a real-time scheduler that enables various "simulation-in-the-loop" scenarios for interacting with actual systems. This means users can transmit and receive packets generated by ns-3 on real network devices. Moreover, ns-3 can act as an interconnection framework to introduce link effects among virtual machines.

3.3.5 Gazebo

Gazebo is an open-source 3D robotics simulator. By using multiple high-performance physics engines, it is capable of creating dynamic simulation replicating gravity, friction, torques, and any other real life conditions that could affect a simulation's success. Gazebo is a good choice when there is no access to actual robotic hardware or we want to test multiple robots simultaneously (FS Studio 2021).

The following are some of the key features and concepts of Gazebo:

- **Physics Engine** - Gazebo comes with a robust physics engine that simulates real-world dynamics, including gravity, friction, collisions, and interactions between objects.
- **Plugins** - Gazebo is highly extensible through plugins. Plugins allow users to add custom sensors, controllers, and other functionalities to the simulation. This feature enables seamless integration with various robotics frameworks and libraries.
- **ROS Integration** - Gazebo is commonly used with ROS. ROS nodes can interact with the Gazebo simulation, allowing developers to test and validate their robot algorithms in a virtual environment.
- **Models** - In Gazebo, a model represents a 3D object, which can be a robot, sensor, or any other object in the simulated world. Models can be imported from 3D modeling software or generated using Gazebo's built-in tools.
- **Worlds** - Worlds in Gazebo define the environment in which models interact. A world file contains the layout of objects, terrain, lighting, and other environmental properties. Multiple worlds can be created and switched during a simulation.
- **Sensors** - Gazebo supports a wide range of sensors, including cameras, LiDARs, depth sensors, and IMUs. These sensors generate data that can be used by the simulated robots for perception and navigation.

- **Controllers** - Gazebo allows users to define and implement controllers for the simulated robots. Controllers can use sensor data to control the robot's motion and behavior.
- **Real-Time Visualization** - Gazebo provides real-time visualization of the simulation, allowing users to observe the robot's movements and interactions with the environment.

Overall, Gazebo offers a comprehensive set of features and concepts that make it a valuable tool for robotics researchers, developers, and enthusiasts. Its flexibility, realism, and integration with ROS have made it a standard choice for simulating and testing robotic systems.

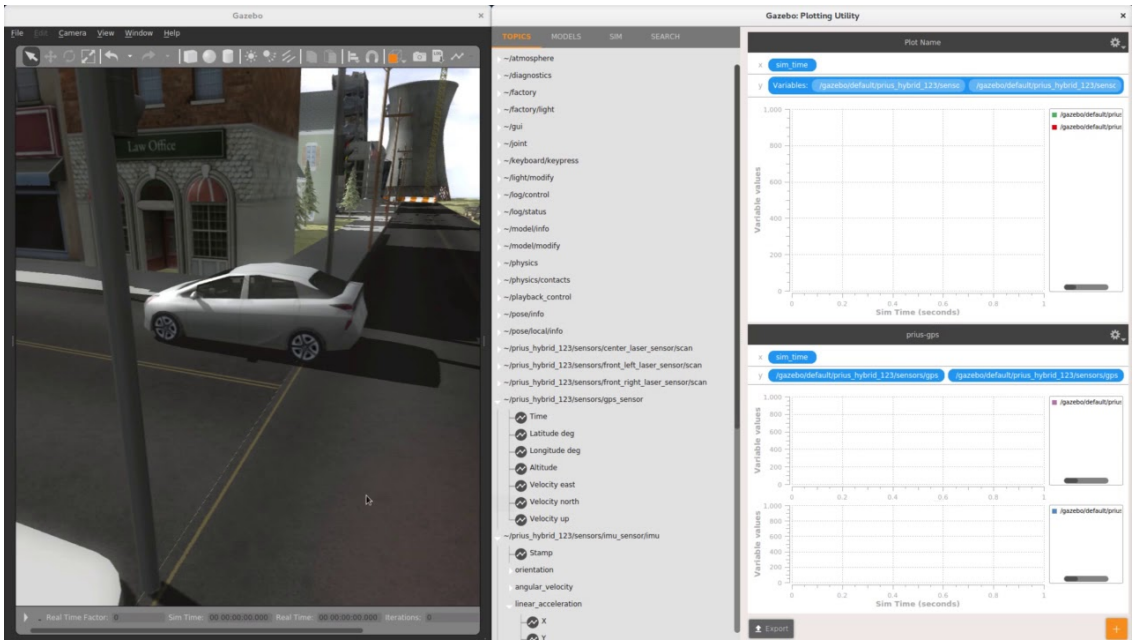


Figure 3.7: Gazebo vehicle and city simulation. (Gazebo 2017)

3.3.6 CARLA

CARLA (Car Learning to Act) (Dosovitskiy et al. 2017), is an open simulator that has been meticulously crafted to cater to the requirements of developing, training, and validating autonomous driving systems. Beyond offering open-source code and protocols, CARLA supplies freely available open digital assets such as urban layouts, buildings, and vehicles, all specifically designed for this purpose. The simulation platform boasts adaptable sensor suite specifications, the ability to manipulate environmental conditions, comprehensive control over both stationary and moving entities, map generation capabilities, and a host of other functionalities. CARLA has been designed with a focus on rendering and physics simulation that emphasizes flexibility and realism. It is developed as an open-source layer integrated with Unreal Engine 4.

The following are some of its highlighted features (CARLA Team 2023):

- **Scalability via a server multi-client architecture** - multiple clients in the same or in different nodes can control different actors.

- **Flexible API** - CARLA exposes a powerful API that allows users to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors, and much more.
- **Autonomous Driving sensor suite** - users can configure diverse sensor suites including LiDARs, multiple cameras, depth sensors and GPS among others.
- **Fast simulation for planning and control** - this mode disables rendering to offer a fast execution of traffic simulation and road behaviors for which graphics are not required.
- **Maps generation** - users can easily create their own maps following the OpenDrive standard via tools like RoadRunner.
- **Traffic scenarios simulation** - CARLA's engine **ScenarioRunner** allows users to define and execute different traffic situations based on modular behaviors.
- **ROS integration** - CARLA is provided with integration with ROS via its ROS-bridge.
- **Autonomous Driving baselines** - CARLA provides Autonomous Driving baselines as runnable agents, including an AutoWare agent and a Conditional Imitation Learning agent.

3.4 Critique Analysis

This section conducts a critique analysis regarding the technologies introduced earlier. The aim is to compare each technology and provide an explanation for their individual selection within this work.

Starting with the robotics framework, given that the usage of ROS2 is a prerequisite for this project, our primary focus has naturally been on ROS2. However, we are aware of the presence of alternative frameworks like PyRobot, OROCOS, YARP, and Orca. Nevertheless, we remain convinced that ROS2 is the most suitable choice due to several compelling reasons. First and foremost, ROS2 boasts an extensive library of packages and tools, making it a versatile and powerful platform for robotic development. Community support is another crucial factor, and in the case of ROS2, it has an active community, making it easier to find solutions to issues and seek guidance when needed. This robust support network is a significant advantage in troubleshooting and advancing the project efficiently. Additionally, our existing familiarity with ROS2 played a pivotal role in our decision, which allowed for a smoother and more rapid development process, thanks to our prior experience with the framework. Regarding the choice of micro-ROS for enabling ROS2 in microcontrollers, the previously demonstrated comparison with its similar approaches clearly proves its superiority and is still a continuously evolving technology.

Regarding the communications technologies, the main reason we explored 6LoWPAN is due to it being already implemented by default in micro-ROS. Furthermore, 6LoWPAN aligns well with the specific requirements of IoT applications, especially in the context of wireless sensor networks, where numerous sensors collect data from the physical world and transmit it wirelessly to a central point for processing. Its low power consumption and support for mesh networking enhance the scalability and reliability of these networks. Also, the previously mentioned work regarding the smart warehouse micro-ROS use-case also utilized 6LoWPAN and reinforces its suitability for building smart environments using wireless sensor

networks. We also explored ETSI ITS-G5 which was used for our approach regarding the communications between the vehicle and a road side unit which will be further explained in a later section. Besides being implemented in AuNa, which facilitated the process of integration with our desired approach, it is a european standard for V2V comms.

As mentioned previously, one of the main goals of this work is the simulation of the components of the system without the need for physical hardware. And thus, we explored a method to accurately replicate the execution of micro-ROS on real hardware. The method we investigated, as detailed in the earlier work addressing the micro-ROS simulation environment, leveraged QEMU, a tool capable of emulating the architectures of certain microcontrollers. Unfortunately, it failed to establish a reliable connection with the micro-ROS agent and exhibited other limitations, as indicated by the original authors. In contrast, the Zephyr emulator, officially endorsed by the micro-ROS project, emerges as the optimal solution. It excels in providing stable execution of micro-ROS while emulating a real-time operating system, namely Zephyr.

The reasoning behind our choice of the communications simulator aligns with the logic behind our choice of ETSI ITS-G5. OMNeT++ was already seamlessly integrated into AuNa, coupled with ROS2 compatibility, which in turn eased its integration into our approach. However, we are also cognizant of a noteworthy alternative, namely ns-3, which allows for high performance real-time simulations. But after thorough research, OMNeT++ emerges as the preferred choice due to its superior modularity, community support, and advanced visualization tools that greatly enhance the analysis of simulation outcomes.

In the realm of robotics simulators, which are instrumental for simulating vehicles, sensors, and their respective environments, we thoroughly evaluated both Gazebo and CARLA. Our choice leaned towards Gazebo for several compelling reasons, with the main reason being that it is also used in AuNa. Furthermore, it it is seamlessly integrated in ROS2 which enables easy communication between various robotic components and access to a vast library of ROS packages. Also, its versatility is another noteworthy factor as it accommodates simulations of a wide array of robotic systems, not exclusively autonomous vehicles. On the other hand, CARLA is focused on simulating autonomous driving scenarios, which makes it ideal for autonomous vehicle development. However, to our knowledge, CARLA is less extensible in terms of custom robot creation or complex simulations beyond autonomous vehicles. Given our specific project requirements, which necessitate sensor placement beyond the vehicle, this constraint played a crucial role in our decision. Nonetheless, we acknowledge CARLA's significance as a compelling alternative to Gazebo, its concentrated focus on autonomous vehicle development, when further explored, could potentially yield more intricate scenarios, enhancing the validation of the studied cooperative perception approaches.

Chapter 4

Architecture & Concept

This section provides an overview of the concept and the overall system architecture of the framework. It will be presented the composition of the framework and the functions of each component, as well as how they interact with each other. It will also be presented the use-case scenario used for the preliminary testing of the application.

As stated before, this thesis focuses on the design and development of a proof-of-concept co-simulation framework to simulate dynamic environments that rely on embedded devices for perception. The framework will employ micro-ROS due to its ability to integrate recent ROS toolsets into embedded systems. The approach taken is inspired by AuNa, adapted to suit the specific objectives of this thesis. Key tools utilized in AuNa, such as ROS2, Gazebo, and OMNeT++, play essential roles in this work:

- **ROS2**: Serves as the core of the system, facilitating integration and communication among all subsystems through the node and topic architecture.
- **Gazebo**: Responsible for simulating the desired environment, vehicles, and sensors.
- **OMNeT++**: Responsible for simulating the communications network. It will simulate the communication between the sensing devices and the vehicle.

Using the previous tools, the overall framework is composed by the following components:

- **micro-ROS Zephyr App**: Developed micro-ROS Zephyr application responsible for receiving the sensor data from Gazebo and forwarding it, acting as a microcontroller.
- **ROS2 Package**: Developed ROS2 package inspired by AuNa that serves as the core of the system. It is responsible for the vehicle's system which includes the navigation system and driving controller. It also contains the necessary nodes for the communication between the sensors and the vehicle.
- **Custom OMNeT++ Artery Model**: This custom OMNeT++ model draws inspiration from AuNa, which itself is based on the Artery model—a V2X simulation framework designed for ETSI ITS-G5. The customized implementation enables efficient and accurate simulations of vehicular-to-everything (V2X) communications in the context of our research. Thanks to AuNa, the integration with ROS2 is already developed.

Each component interacts as depicted in figure 4.1, which represents the system architecture. The transmission of obstacle data to the vehicle involves utilizing an ITS-G5 communication system through a Road-Side Unit (RSU). This approach will be elaborated on in a subsequent section for better understanding. The RSU's navigation system utilizes the sensor data to generate obstacle data within its map. Subsequently, it forwards this data to a dedicated node responsible for creating a new message destined for OMNeT++. In the simulation,

OMNeT++ emulates the message transmission from the RSU to the vehicle using ITS-G5. Finally, the vehicle's communication node, designed to receive messages from OMNeT++, receives the transmitted message. The communication node then extracts the obstacle map data from the received message and incorporates it into its own map.

It's important to note that the communication simulation currently encompasses only the RSU-to-vehicle communication. Simulating the communication from the sensors to the RSU, which would involve the micro-ROS client-to-agent connection, is pending implementation. This aspect demands further exploration and a deeper understanding of the XRCE-DDS protocol to accurately replicate the connection in OMNeT++. A suitable technology for micro-ROS such as 6LoWPAN, could be considered for this purpose.

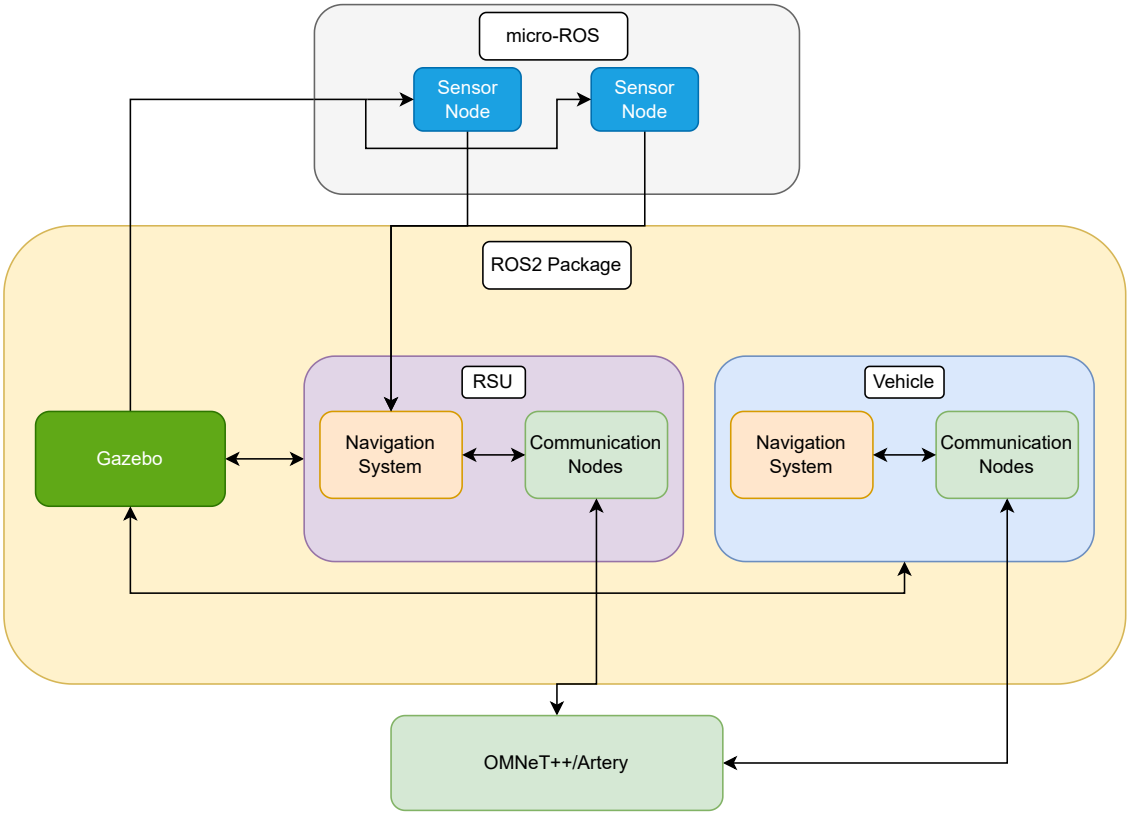


Figure 4.1: System Architecture.

To support the development of the co-simulation framework, and better understand the concept, we proposed a preliminary use-case scenario. The scenario consists in the cooperation between an autonomous vehicle with a predefined route, and a sensing device running micro-ROS. A common real world scenario is the blocking of a road due to road works or a natural phenomenon such as a fallen tree blocking the road. These type of situations cannot be predicted by a vehicle with a predetermined route, so the purpose of this use-case is to warn with precedence the vehicles approaching the road block so they can take the detour.

In practise, this will consist in using the sensing device next to the road block to aid the approaching vehicles by providing them obstacle information about the obstacle blocking the road. The vehicles will then receive the message and obtain the information that the road is blocked and the current trajectory cannot be used, which will force them to use another route.

The following figure (4.2) represents the use-case scenario described previously. In it, we can see a vehicle with a predefined route approaching a road that is blocked, which then obtains the information about the road block, allowing it to calculate a detour route.

This scenario was also used in the evaluation and validation of this work, which will be analysed in a later section.

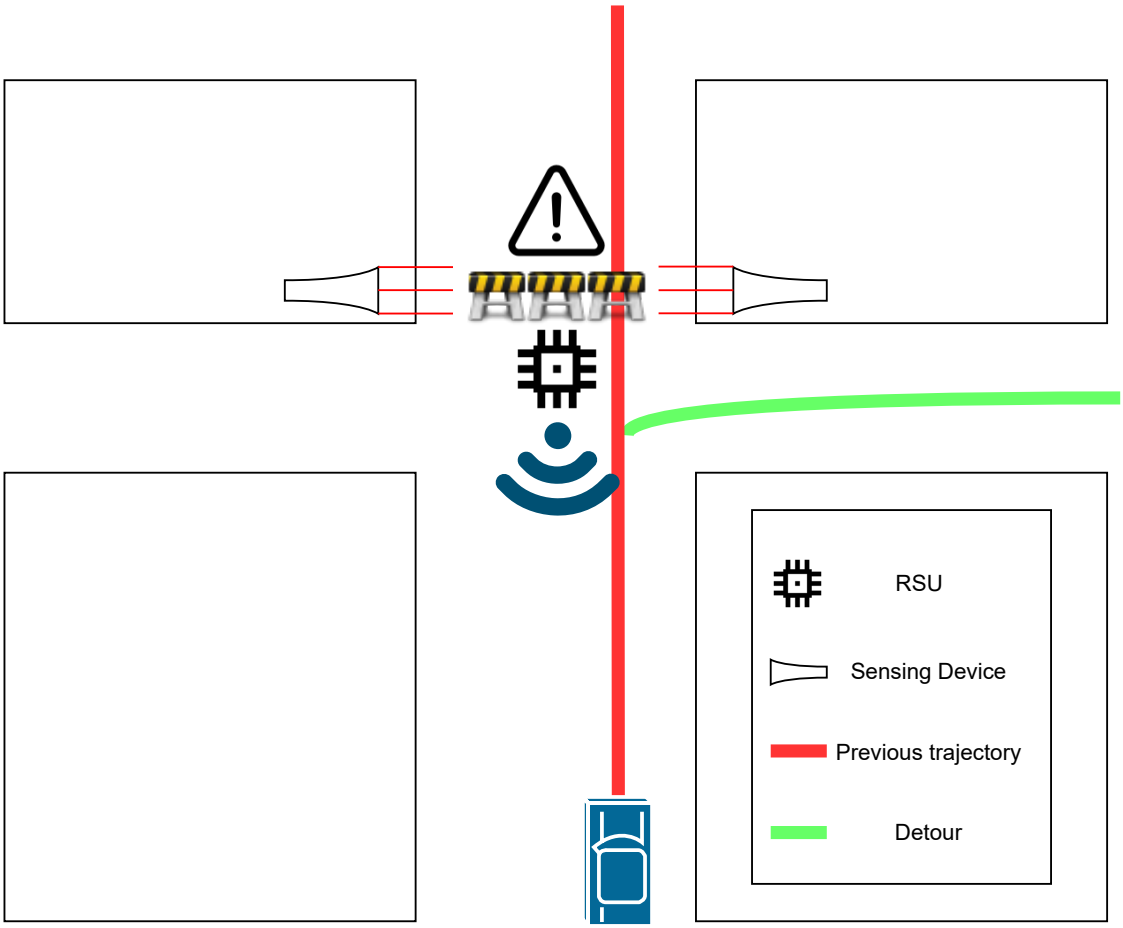


Figure 4.2: Use-Case scenario.

In conclusion, the developed framework aims to simulate scenarios involving cooperative interactions between environment-proximate sensors and the vehicle. This approach involves using sensing devices to gather sensor data, while micro-ROS facilitates communication and integration with the RSU, which is running ROS2. The RSU receives sensor data from the sensing devices, consolidates it into a single obstacle message, and communicates with the vehicle through ITS-G5. The vehicle's navigation system utilizes this received data to plan and respond accordingly. Such setup will enable us to simulate cooperation scenarios focusing on:

- Sensing nodes application development and pre-deployment validation.
- RSU to vehicle communications setup and validation.
- A.V cooperative application development and validation.

The previous topics will be further explained in the next chapter, and later in the experimental results chapter, it will be depicted the two use-case scenarios used for the validation of the co-simulation framework and the cooperative perception approach. These use-case scenarios serve as illustrations of how external sensors can enhance a vehicle's environmental perception. Furthermore, they showcase the co-simulation framework's utility in assessing whether the communication systems meet the stringent time constraints associated with specific scenarios.

Chapter 5

Framework Development

This chapter presents the development of the co-simulation framework. First, it is explained the approach used to simulate micro-ROS, and the process behind the micro-ROS app development. Following the first section, it is explained the system's core and its modus operandi, as well as the required steps to achieve the desired use-case scenarios.

5.1 micro-ROS Application Development

This section overviews the method used for the micro-ROS simulation as well as its operation. It also explains the development of the micro-ROS application.

5.1.1 micro-ROS Simulation Environment

As stated previously, it was necessary to define a method of running micro-ROS in order to simulate the communication between a sensing device running a micro-ROS app and the framework. Like the previously cited work (Flores Muñoz and Outerelo Gamarra 2019), it was also explored the use of QEMU to simulate the microcontroller and run the micro-ROS stack, but due to the lack of updates since the date stated by the authors, this approach was abandoned given that the authors explained that the simulation achieved using QEMU STM32 was unstable regarding the communication between the micro-ROS client and agent.

Since in this case the priority was to simulate the micro-ROS stack, we opted for the solution presented in the official micro-ROS website, which, while it does not emulate a microcontroller board, it emulates a RTOS capable of stably running the micro-ROS stack. This approach utilized the Zephyr RTOS emulator which allows to emulate and simulate the behavior of the Zephyr RTOS on a host machine.

Regarding the operation of the communication between the micro-ROS node and the ROS2 framework in the host machine, micro-ROS uses **eProsima Micro XRCE-DDS** (eProsima 2018). It is a software solution that implements a client-server protocol that enables resource-constrained devices (clients) to take part in DDS communications. The Micro XRCE-DDS Agent (server) acts as a bridge to make this communication possible, which allows the micro-ROS node to communicate with a ROS2 host, enabling interoperability between a microcontroller running a RTOS, and a host machine running a regular OS.

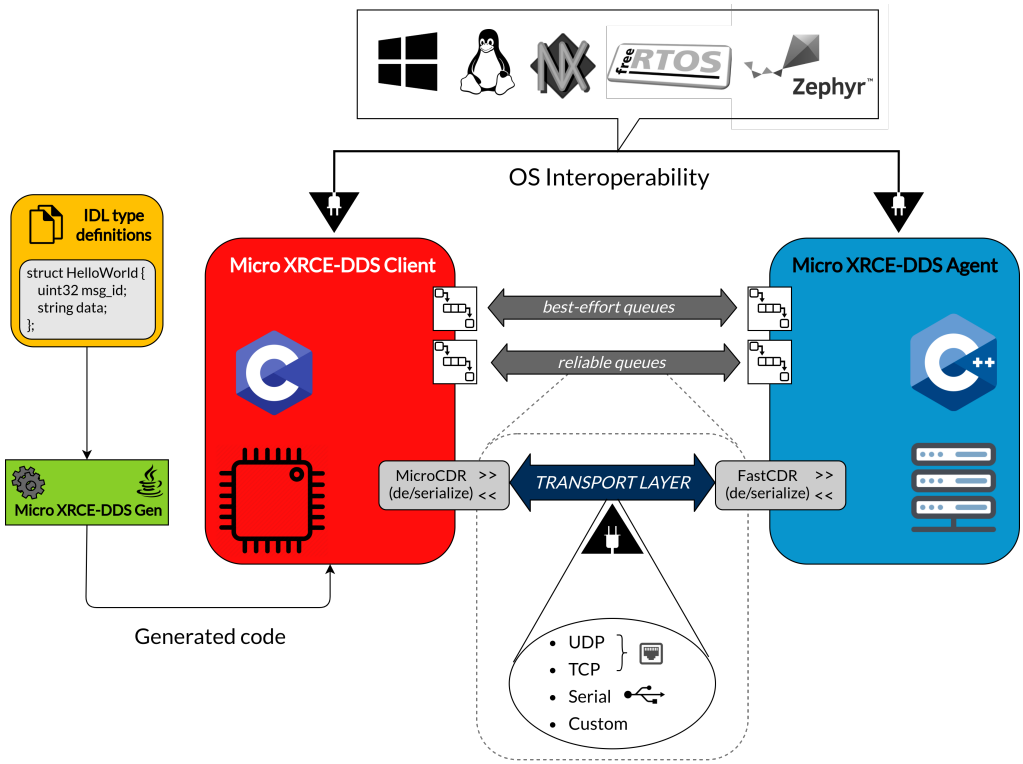


Figure 5.1: Micro XRCE-DDS Client-Agent Architecture. (eProsima 2018)

5.1.2 Developing the micro-ROS Zephyr Application

In order to develop a new zephyr app, it is necessary to setup the required files, which consist in the following (micro-ROS 2023f):

- **main.c** - contains the app logic.
- **app-colcon.meta** - contains the micro-ROS app specific colcon configuration.
- **CMakeLists.txt** - contains the script for compiling the application.
- **<transport>.conf** - a Zephyr specific and transport-dependent app configuration file, which can be serial, serial-usb and host-udp.

Basis Application

In this case, the transport used was UDP. The developed app used as a basis the micro-ros_reconnection app which is one of the demo apps. This an app that simply establishes a connection with the micro-ROS agent and publishes an Int32 to the topic "/counter", but uses an important approach that makes it capable of reconnecting to the agent in case of connection loss. To achieve this, it relies on four different states, and the micro-ROS middleware function for pinging the agent is responsible for managing the states. The flowchart of the states is depicted as follows:

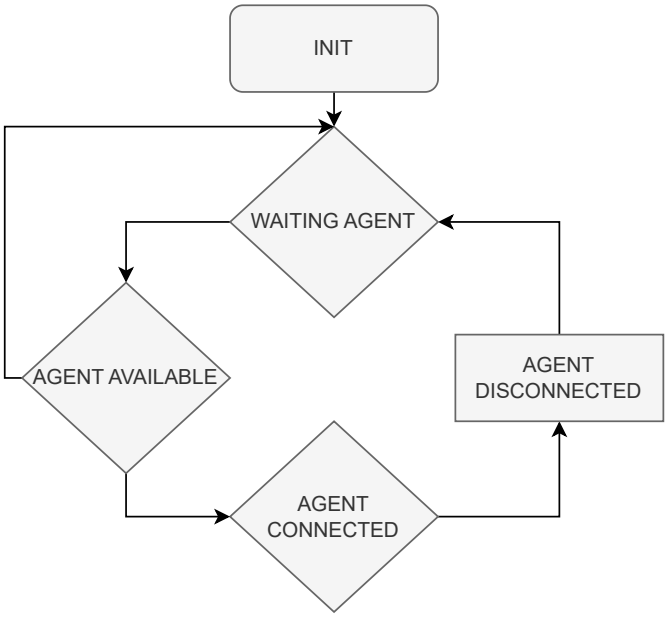


Figure 5.2: Zephyr base app flowchart.

In the first state, the app simply pings the micro-ROS agent until it is successfully reached, changing the state to **AGENT AVAILABLE**. In this state, the program creates the required entities for the connection with the micro-ROS agent, initializes the publisher and the state is changed to **AGENT CONNECTED**. In case the entity creation is not successful, the connection is considered to be lost, and the state goes back to **WAITING AGENT**. The next state simply continuously pings the agent to check the connection status while also spinning the executor which keeps the node active and responsive by continuously processing incoming messages, handling callbacks, and executing the associated logic. In case of no response by the agent, the state is forwarded to **AGENT DISCONNECTED** where all the entities are destroyed so the state can return to **WAITING AGENT**. The advantage of this approach is the ability to always attempt reconnecting to the agent in case of a problem with the connection, which makes it mandatory in a more realistic scenario.

Developed Application

As explained previously, the developed application used the micro-ros_reconnection app as its basis due to the huge advantage of being able to keep a persistent connection to the micro-ROS agent. The purpose of the sensing devices is to sense the environment, and thus, the focus of the developed application was to simulate micro-ROS nodes sending sensor data. Since the environment is simulated, in order for the micro-ROS node to gather sensor data it needs to obtain it from ROS2 Gazebo which is responsible for simulating the environment and its actors. In Gazebo, we created an entity whose purpose is to simulate a sensing device with a 2D LiDAR, more specifically, a Hokuyo LiDAR. Further details regarding Gazebo and the environment will be explained in the next section. The sensor generates a ROS2 LaserScan message that contains obstacle data, which is composed as follows (ROS 2022):

- **Header** - Contains two fields:
  - **stamp** - The aquisition time of the first ray in the scan.

  - **frame_id** - The id of the related coordinate frame.
- **angle_min** - Start angle of the scan in radians.
- **angle_max** - End angle of the scan in radians.
- **angle_increment** - Angular distance between measurements in radians.
- **time_increment** - Time between measurements.
- **scan_time** - Time between scans.
- **range_min** - Minimum range value in meters.
- **range_max** - Maximum range value in meters.
- **ranges** - Array of laser scan measurements (range values) corresponding to each laser beam in the scan. Each range value represents the distance from the sensor to an obstacle or target at a specific angle.
- **intensities** - Optional field that provides intensity or reflectivity measurements associated with each laser beam. It represents additional information that can be captured by some laser scanners.

The configured settings for the laser hokuyo in Gazebo used (-0.8, 0.8) for minimum and maximum angle, and (0.2, 12) for minimum and maximum range. Another important setting is the samples parameter, which define the angular resolution of the laser scanner by specifying the number of evenly spaced rays or beams in a full revolution, controlling the level of detail or granularity in the laser scan measurements, which means that a higher value leads to a more detailed measurement, but also increases computational load. In this case, the used samples value was 640. Regarding its 3D representation in the environment, it was used an actual hokuyo model from the Gazebo demo files.

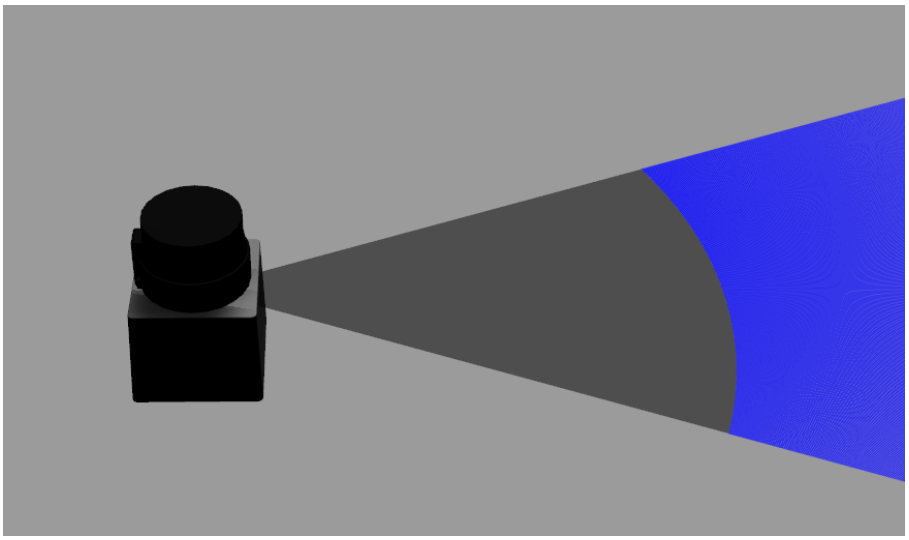


Figure 5.3: Laser Model in Gazebo.

With the laser scan simulation ready, the micro-ROS node could now subscribe to the LaserScan message and obtain the obstacle data, so it could publish it to a topic in order to simulate that it is a sensing device running micro-ROS who is sharing the LaserScan

data. When attempting this, an issue was encountered due to the fact that the MTU (Maximum Transmission Unit) size of a message in micro-ROS is 512 Bytes, and the LaserScan message that was being generated with the previous mentioned parameters had a size of 5192 Bytes. Normally this would not be an issue due to micro-ROS's ability to perform message fragmentation which allows the node to send and receive messages longer than the MTU (micro-ROS 2023e), but this is only achievable using a Reliable QoS (Quality of Service) stream, which is not adequate for laser scan messages due to time constraints such as higher latency due to all messages being ensured that are being delivered, and also higher bandwidth caused by the additional communication overhead. On the other hand, for Best-Effort streams, while they have lower latency and efficient bandwidth usage, there is no message fragmentation because Best-Effort streams consist in a single data buffer where only one message at a time is handled. They send and receive data leaving the reliability to the transport layer, and the message size handled by such a stream must be less or equal than the MTU defined in the transport used (micro-ROS 2023e).

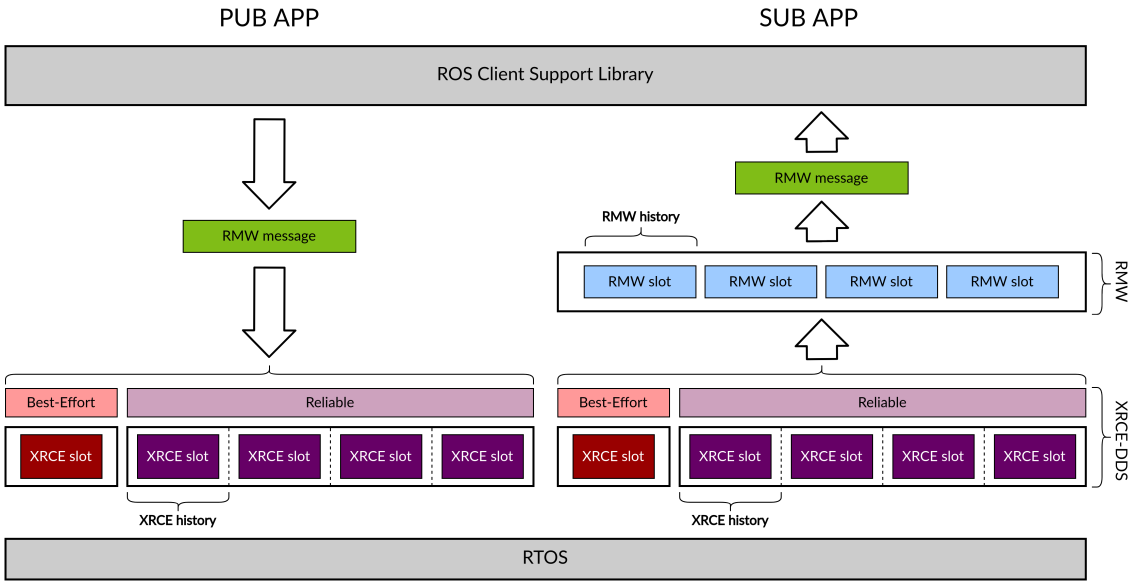


Figure 5.4: Memory management of the micro-ROS Client library in publishers and subscribers applications. (micro-ROS 2023e)

Given this issue, the first attempt to mitigate it was to lower the message size by reducing the number of samples, which meant that the object detection would not be as accurate but it was still acceptable. This solution reduced the message size to 488 Bytes, making it possible to receive and send LaserScan messages using a Best-Effort stream. The problem with this approach is that there is an issue with the algorithm used for adding the object to a map, which will be further discussed in later section. It is not able to remove the object from the map when it is no longer being detected if the samples value is too low. This makes it not possible to use the desired approach for sharing the objects with the vehicle, and thus, this solution is not suitable.

Therefore, the only valid solution that suits the approach used in navigation package, is using a Reliable stream. This required creating a ROS2 node that receives the Best-Effort LaserScan message and publishes it to a Reliable stream, "converting" the message. With this approach, the micro-ROS node was now able to receive the LaserScan message and publish it to another topic, which simulated the idea that the sensing device was the one

providing the data. This data could now be visualized in RVIZ by subscribing to the topic provided by the micro-ROS node.

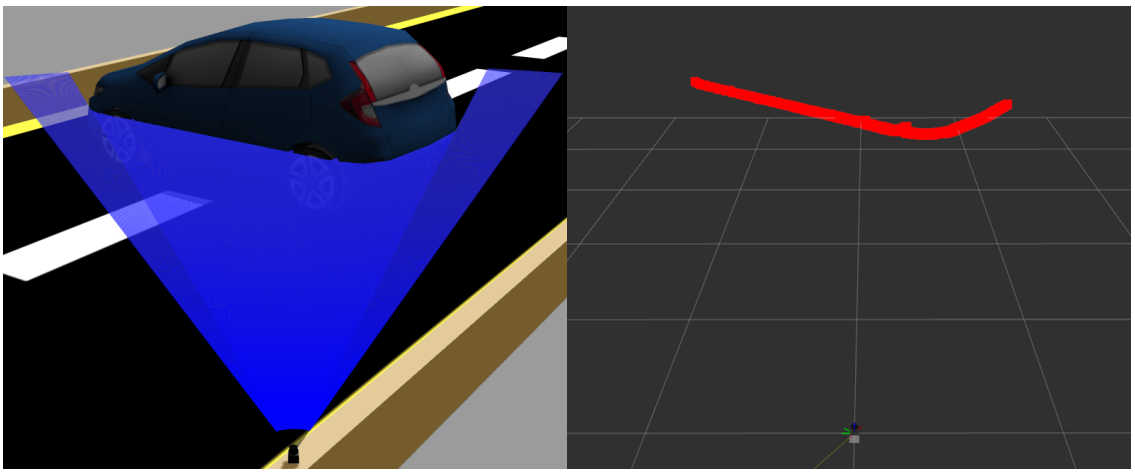


Figure 5.5: Laser detecting an object (Gazebo on the left, RVIZ on the right).

Since the framework supports multiple sensors, the micro-ROS node also adopts this approach. This was achieved using environment variables due to the fact that in micro-ROS the zephyr_app_main function does not take any arguments, so the workaround was to set an environment variable with the desired <ID> value when the firmware was flashed and the Zephyr emulator was launched. With this approach, the micro-ROS node was capable of subscribing and publishing to the topics that correspond with its <ID>.

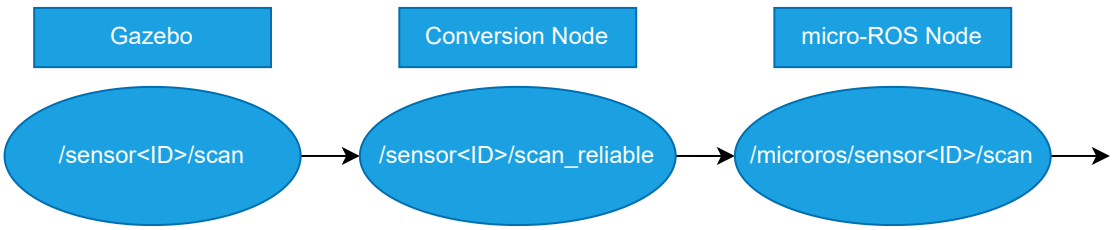


Figure 5.6: Published and subscribed topics.

```
moliveira@moliveira-PE72-7RD:~/trabalhotese/microros_galactic$ ros2 run
    micro_ros_agent micro_ros_agent udp4 --port 8888
[1687269728.508316] info     | UDPv4AgentLinux.cpp | init
    | running...             | port: 8888
[1687269728.508536] info     | Root.cpp           | set_verbose_level        |
     logger setup           | verbose_level: 4
[1687269735.418481] info     | Root.cpp           | create_client
    | create                 | client_key: 0x404814C6, session_id: 0x81
[1687269735.418579] info     | SessionManager.hpp | establish_session        |
     session established    | client_key: 0x404814C6, address:
    192.168.1.74:25255
[1687269735.466369] info     | ProxyClient.cpp    | create_participant       |
     participant created    | client_key: 0x404814C6, participant_id: 0x000(1)
[1687269735.472763] info     | ProxyClient.cpp    | create_topic             |
     topic created          | client_key: 0x404814C6, topic_id: 0x000(2),
    participant_id: 0x000(1)
```

```
[1687269735.476977] info     | ProxyClient.cpp    | create_publisher          |
    publisher created      | client_key: 0x404814C6, publisher_id: 0x000(3),
   participant_id: 0x000(1)
[1687269735.499359] info     | ProxyClient.cpp    | create_datawriter         |
    datawriter created     | client_key: 0x404814C6, datawriter_id: 0x000(5),
    publisher_id: 0x000(3)
[1687269735.500510] info     | ProxyClient.cpp    | create_topic              |
    topic created          | client_key: 0x404814C6, topic_id: 0x001(2),
   participant_id: 0x000(1)
[1687269735.500727] info     | ProxyClient.cpp    | create_subscriber         |
    subscriber created     | client_key: 0x404814C6, subscriber_id: 0x000(4),
    participant_id: 0x000(1)
[1687269735.505742] info     | ProxyClient.cpp    | create_datareader         |
    datareader created     | client_key: 0x404814C6, datareader_id: 0x000(6),
    subscriber_id: 0x000(4)
```

Listing 5.1: micro-ROS Agent Console Output

```
moliveira@moliveira-PE72-7RD:~/trabalhotese/microros_galactic$ export ID=1;
ros2 run micro_ros_setup flash_firmware.sh
Flashing firmware for zephyr platform host
WARNING: Using a test - not safe - entropy source
*** Booting Zephyr OS build zephyr-v2.6.0  ***
Waiting for uROS agent.
ID: 1
Successfully Allocated Memory For Sub
Connected to uROS agent and created entities.
NEW LASERSCAN MESSAGE RECEIVED!
The Size of the ranges is: 640
The Size of the intensities is: 640
Published msg successfully
```

Listing 5.2: micro-ROS Client Node Console Output

As previously mentioned, the developed micro-ROS app is capable of reconnecting to the micro-ROS agent, so it is constantly checking the status of the connecting. Ignoring the steps for maitaining the connection with the agent which were already explained, the micro-ROS app operates as follows, when the connection is established with the agent, it enters the **create_entities** function where it obtains the ID in case it is the first time entering the function, initializing the node after, and then it allocates memory for receiving the LaserScan message using the **micro_ros_utilities_create_message_memory**. This package is able to auto-assign memory to a certain message struct using default dynamic memory allocators, and inits the size for strings, ROS 2 types sequences, and basic type sequences (micro-ROS 2023c), and then creates the publisher and subscriber using the ID obtained for defining the topic names. Having the subscriber created, as long the connection is established with the agent, the application will constantly wait for messages, and whenever a message is received, it enters the **subscription_callback** function. This function is responsible for handling the received message, copying it to a new message, and publishing to the new topic. In case the connection is lost, the entities are destroyed, and when the connection is reestablished, the **create_entities** is entered again, repeating the whole process.

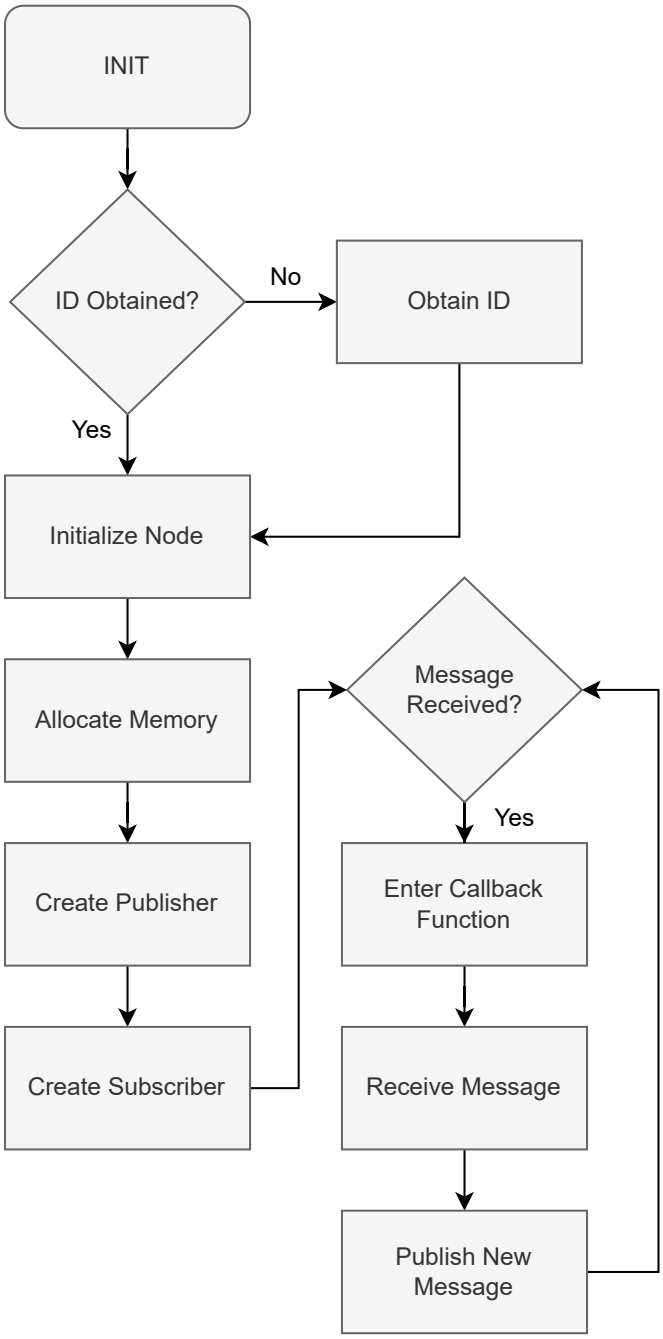


Figure 5.7: micro-ROS Application Flowchart.

5.2 Simulation Framework Core

This section overviews the core of the simulation framework which consists in a ROS2 package that contains several nodes, and Artery, which is a OMNeT++ V2X simulation framework for ETSI ITS-G5. It will first present the ROS2 package as well as its modus operandi. Then, it presents the Artery OMNeT++ model used.

5.2.1 ROS2 Package

The developed ROS2 package was heavily inspired by the AuNa package in terms of its layout and components used for simulating the environment. The package directory layout is organized as depicted in the following:

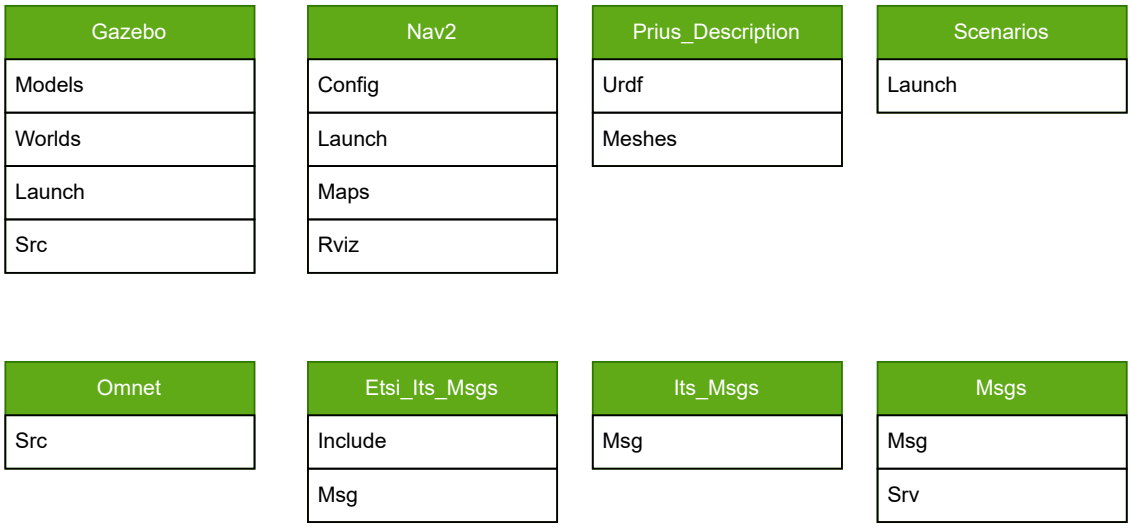


Figure 5.8: ROS2 Package Layout.

Starting with the **Gazebo** directory, it contains the models and world files used by Gazebo, source code that belongs to several nodes in the Src directory, and several launch files for initializing the simulation. The **Nav2** directory contains the configuration files for the ROS2 Navigation2 package, and also the generated maps for the different worlds, and the Rviz configuration file for visualizing the robot within the map. The **Prius_Description** directory contains the configuration and model for the vehicle used in the simulation. In the **Scenarios** folder, there are launch files for launching the desired scenario. The next directories are related to the integration with the OMNeT++/Artery framework, starting with the **Omnet** directory, it has the source code for the node used to send the message to OMNeT++, the **Etsi_Its_Msgs** is the definition of the CAM message type, the **Its_Msgs** is a simplified version of the CAM message, and finally, the **Msgs** is the definition of the commonly used custom message types for integration with OMNeT++.

Since the purpose of this work is to simulate an autonomous vehicle in a simulated environment, the development of the ROS2 package was focused towards Gazebo. The objective was to build a scenario where the vehicle was able to receive data from external sensors that are located around the environment. In order to achieve this goal, it was necessary to define a method of sharing obstacle data by simply sending a message with this type of information.

Robotic Vehicle

The defined robot for the simulation is based on a common model that is available online, it is a model that resembles the Toyota Prius. The definition and configuration of the robot properties are located in an URDF (Unified Robotics Description Format) file. This is a XML-based file format used in the field of robotics to describe the physical and visual properties of a robot, it contains information about a robot's mechanical structure, including

its links, joints, sensors, and other physical properties. It describes the geometry, inertial properties, visual appearance, and kinematic relationships of the robot components. The file format allows robots to be modeled in a hierarchical manner, where individual links and joints are defined, and their connections and transformations are specified. This enables the creation of complex robot models with articulated structures and multi-link systems. URDF files can also be used for robot control, simulation, motion planning, and other robotic applications.

A robot visual properties define the visual 3D appearance, it is defined as follows:

- **Meshes** - Detailed 3D models or meshes associated with the robot's links, representing the visual appearance of each link. These meshes define the shape, texture, and visual attributes of the robot.
- **Materials** - The material properties assigned to the visual meshes, including color, texture, reflectivity, transparency, and shininess.
- **Transformations** - The transformations that define the position and orientation of the visual meshes relative to the corresponding links. They align the visual representation with the physical structure of the robot.

The physical properties define the robot's physical behaviour and dynamics, which are crucial for ensuring realistic simulations.

- **Links** - The robot's physical components or rigid bodies, such as arms, grippers, or chassis. Each link has a specific shape, size, and mass.
- **Joints** - The connections between the links that allow relative movement or rotation. Joints define the robot's range of motion and can be of various types, such as revolute (rotational), prismatic (linear), or fixed.
- **Inertial** - The mass distribution and moments of inertia of the robot's links. These properties affect the robot's dynamics and motion behavior.
- **Collision Geometry** - The simplified geometric representation of the robot's links used for collision detection. It helps in preventing collisions with other objects or robots in the environment.

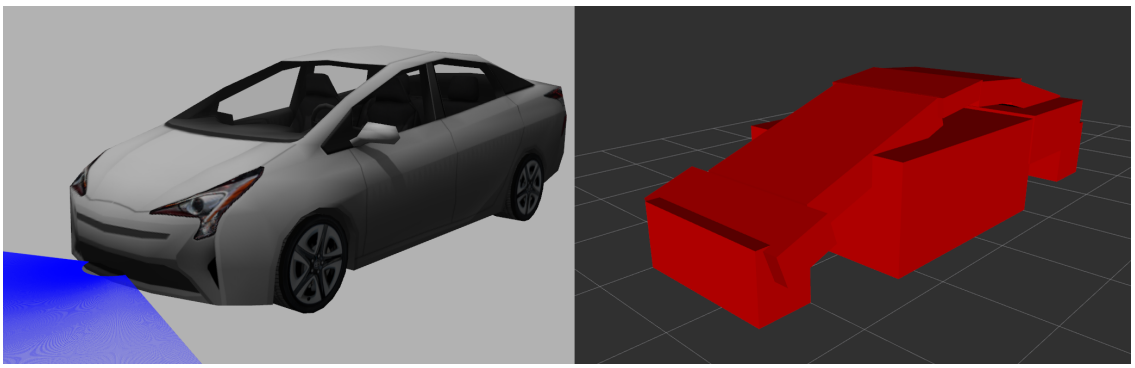


Figure 5.9: Prius Model (Gazebo on the left, RVIZ collision model view on the right).

In the case of the Prius model, it is composed by several links and joints to make it act like an actual vehicle, such as a chassis, rear wheels and front wheels with an axle, a steering

wheel, and a link for the sensor in the front. In the URDF file, the definition of a link refers to a physical component or rigid body of a robot that is part of its mechanical structure. It represents a distinct part of the robot, such as an arm segment, a gripper, or as in this case, a chassis. Links are connected to each other through joints, forming the kinematic chain of the robot. They have associated geometries, inertial properties (mass, moments of inertia), and visual representations (meshes, materials) that define their physical and visual properties. When defining a link, it is also defined a frame which represents a reference coordinate system in three-dimensional space. It defines the position and orientation of objects or components relative to a common origin. Frames are used to express the spatial relationships between different parts of a robot, such as its base, joints and sensors.

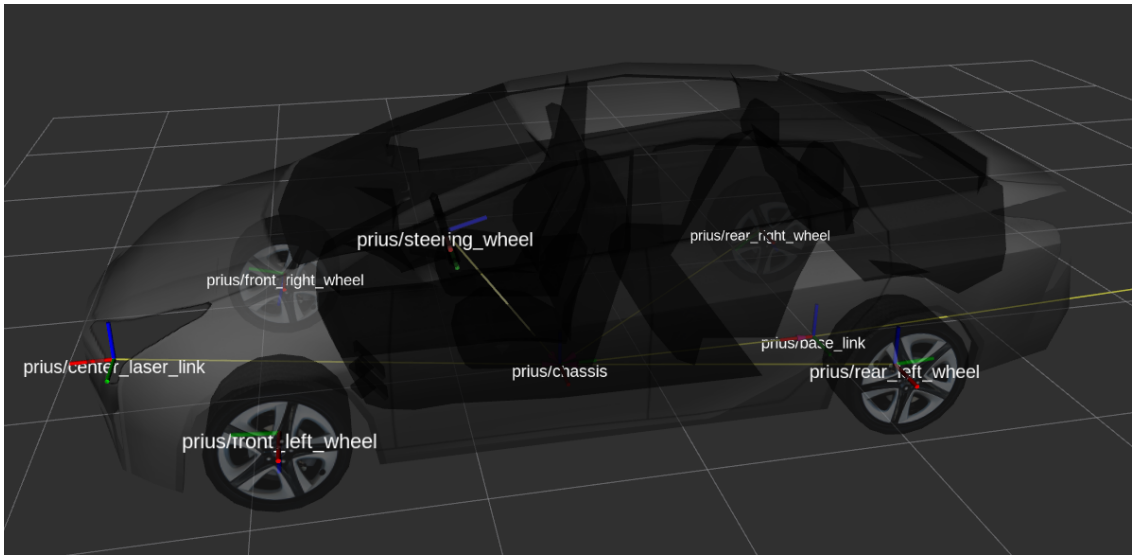


Figure 5.10: Prius Frames in RVIZ.

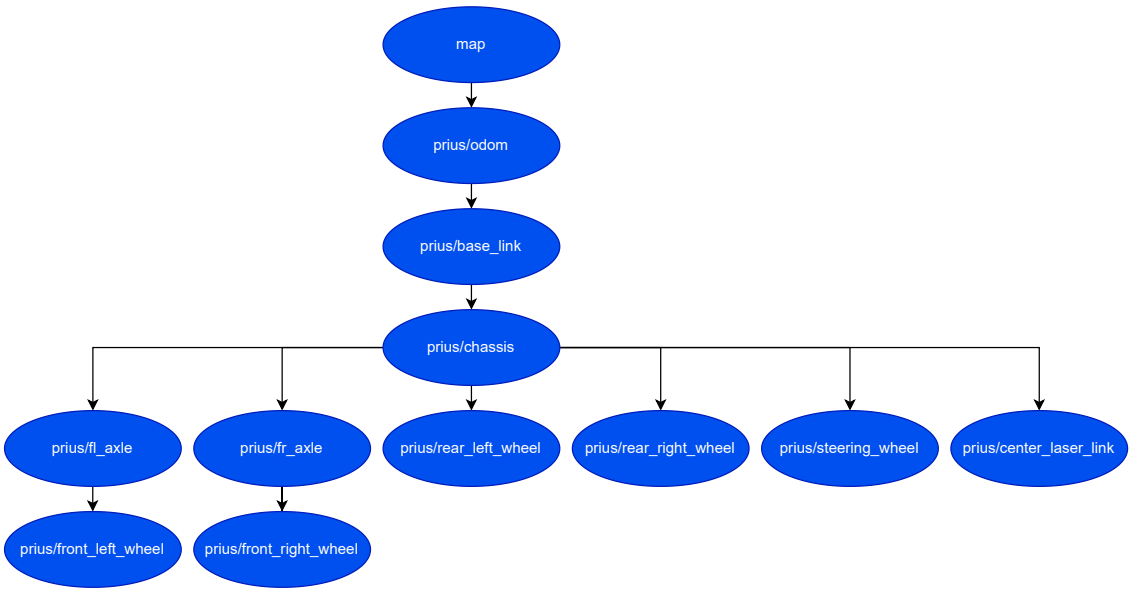


Figure 5.11: Prius Frame Tree.

In a ROS2 Gazebo scenario, an URDF file can also include Gazebo specific plugins that

provide additional functionalities within the simulation environment. The plugins used by the Prius are the following:

- **joint_state_publisher** - used to publish the joint states of a robot model within the Gazebo simulation environment for the integration between the Gazebo simulator and the ROS2 ecosystem.
- **ackermann_drive** - this plugin is designed to simulate and control Ackermann-steering wheeled vehicles within the Gazebo simulation environment. The Ackermann steering mechanism is a common steering system used in vehicles with differential drive, where the front wheels are controlled independently to achieve steering. With this plugin, it is possible to control the robot using the ROS2 topic created by it.
- **gazebo_ros_head_hokuyo_controller** - a plugin for simulating and controlling a Hokuyo laser scanner in Gazebo, it simulates the behavior of a Hokuyo laser scanner and provides an interface to access the laser scan data within the ROS ecosystem, publishing the simulated laser scan data as a ROS topic, which allows other ROS nodes and components to subscribe to it and utilize the laser scan information for various purposes, such as mapping, obstacle avoidance, or localization.

Still regarding the **ackermann_drive** plugin, it is important to mention that it also provides odometry data. Odometry involves measuring the changes in position and orientation of a robot over time by analyzing its internal sensors, such as wheel encoders, gyroscopes, or accelerometers. By tracking these changes and integrating them over time, odometry provides an estimation of the robot's current pose relative to a reference frame. It is crucial for localization, mapping, and navigation.

Navigation

The navigation system plays a crucial role in the realm of autonomous robotics, enabling robots to operate independently and intelligently in dynamic environments. It empowers robots to perceive their surroundings, plan optimal paths, and execute precise movements to accomplish various tasks. By leveraging sensor data and sophisticated algorithms, the navigation system enables robots to perceive obstacles, map their surroundings, and make informed decisions to navigate from one point to another.

The system used in the developed framework is **Nav2** (Navigation 2) (Steve Macenski, Martin, et al. 2020), which is the successor of the ROS Navigation Stack. Nav2 uses behavior trees to create customized and intelligent navigation behavior via orchestrating many independent modular servers. A task server can be used to compute a path, control effort, recovery, or any other navigation related task. These separate servers communicate with the behavior tree (BT) over a ROS interface such as an action server or service. A robot may utilize potentially many different behavior trees to allow a robot to perform many types of unique tasks (Nav2 2020a). Some of its relevant tools and features are the following:

- Load, serve, and store maps **(Map Server)**.
- Localize the robot on the map **(AMCL)**.
- Plan a path from A to B around obstacles **(Nav2 Planner)**.
- Control the robot as it follows the path **(Nav2 Controller)**.
- Smooth path plans to be more continuous and feasible **(Nav2 Smoother)**.

- Convert sensor data into a costmap representation of the world **(Nav2 Costmap 2D)**.
- Build complicated robot behaviors using behavior trees **(Nav2 Behavior Trees and BT Navigator)**.
- Compute recovery behaviors in case of failure **(Nav2 Recoveries)**.
- Follow sequential waypoints **(Nav2 Waypoint Follower)**.
- Manage the lifecycle and watchdog for the servers **(Nav2 Lifecycle Manager)**.
- Plugins to enable your own custom algorithms and behaviors **(Nav2 Core)**.
- Monitor raw sensor data for imminent collision or dangerous situation **(Collision Monitor)**.
- Python3 API to interact with Nav2 in a pythonic manner **(Simple Commander)**.
- A smoother on output velocities to guarantee dynamic feasibility of commands **(Velocity Smoother)**.

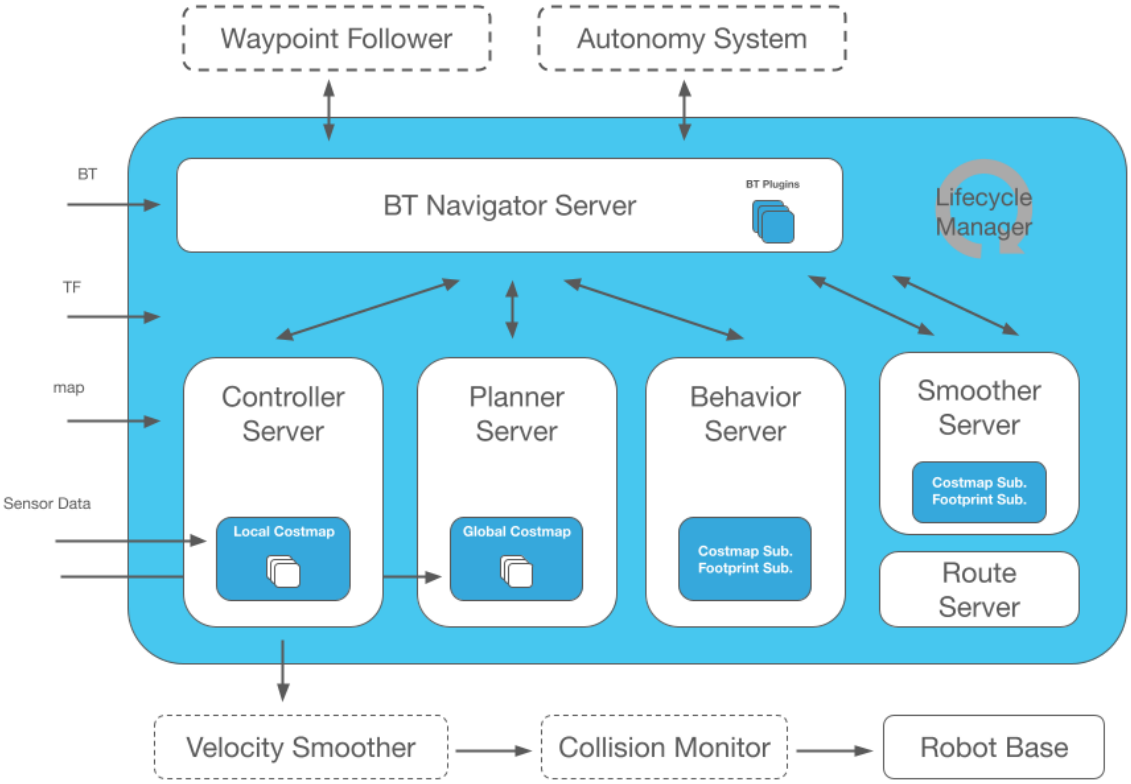


Figure 5.12: Nav2 Architecture. (Nav2 2020a)

Since the vehicle needs to be autonomous for our simulation environment, some of the Nav2 features were implemented in the Prius robot. The implemented features that were relevant to simulate the desired scenarios allowed it to load maps, localize itself on the map, plan paths, and add obstacles to the map.

In order for the navigation system to work, it is required a map, so, the first step was to create a map of the Gazebo environment which was achieved using **SLAM Toolbox** (Steve Macenski and Jambrecic 2021). SLAM (Simultaneous Localization and Mapping) Toolbox

is a collection of software tools and algorithms for 2D SLAM. SLAM refers to the process of constructing a map of an unknown environment while simultaneously estimating the robot's pose within that map. While SLAM Toolbox was running, the robot was driven around the environment using the ackermann_drive plugin until the whole environment was mapped, in the end of the process, it was created a PGM (Portable Gray Map) file of the mapped environment.

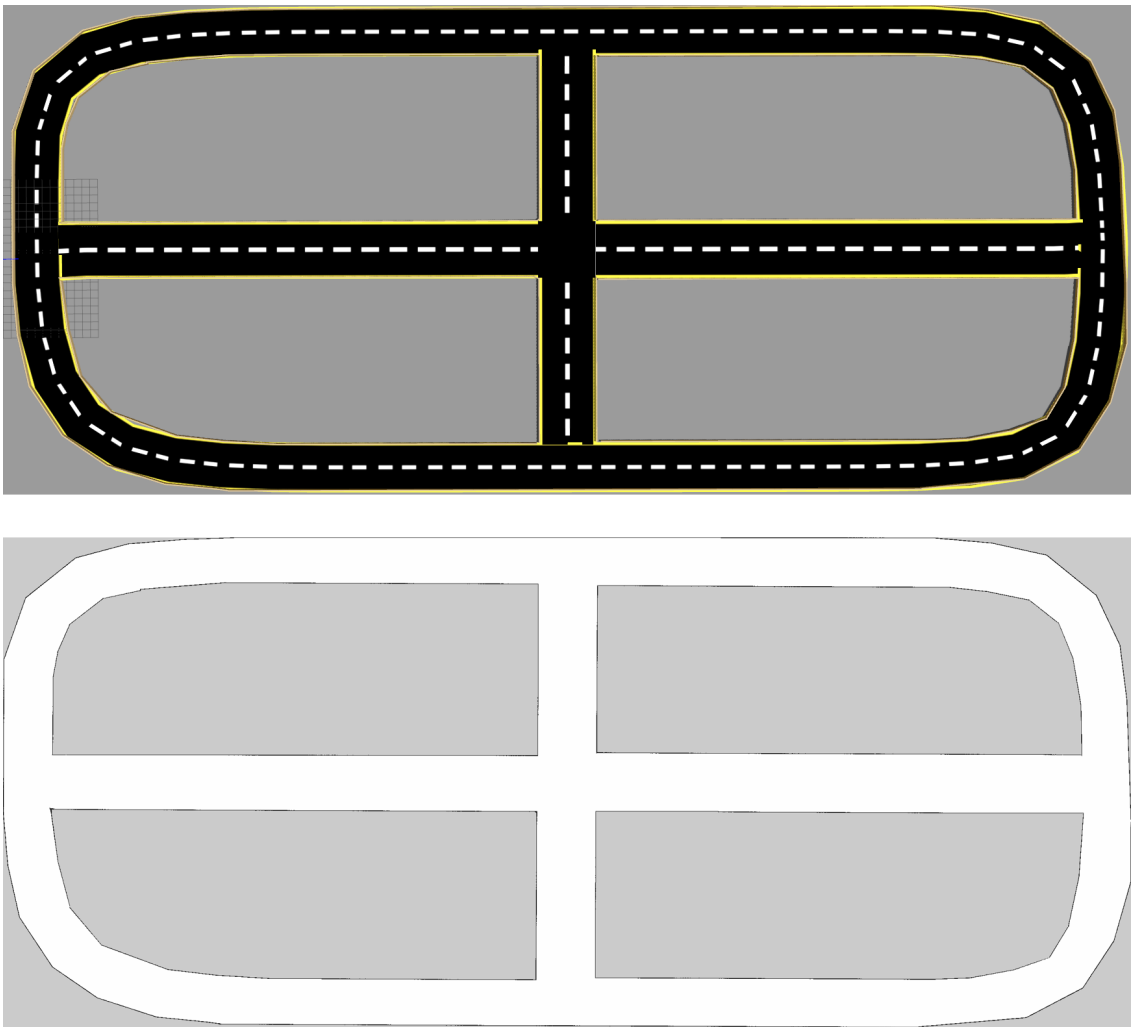


Figure 5.13: Gazebo World and the Created Map Respectively.

The used Gazebo world was obtained from another ROS2 package, it is a simple environment with a road that is not demanding in terms of graphical capacity, which is very useful for testing. In the created map file the three different colors represent the following:

- **White** - Represents free or unoccupied space. It indicates areas where the robot has detected no obstacles or obstacles that are far away.
- **Gray** - Represents areas of uncertainty or ambiguity in the map. These areas may correspond to partially observed or poorly localized regions. Gray can indicate areas where the SLAM algorithm is less confident about the presence or shape of objects.

- **Black** - Black is commonly used to represent obstacles or occupied areas in the map. It signifies regions where the robot has detected obstacles or objects. Black areas typically correspond to walls, or any other objects that are present in the environment.

With the generated map, and with the adequate RVIZ configuration file, it was now possible to visualize the map with RVIZ but not to locate the robot within the map. In order for the rest of the navigation stack to work, it was necessary to configure the Nav2 parameters file. This file is used to specify various parameters and settings for the navigation stack, these parameters control various aspects of the navigation system, such as planner behavior, sensor settings, robot kinematics and costmap configuration. To enable the robot localization on the map, it was necessary to configure the **AMCL** (Adaptive Monte Carlo Localization) parameters. AMCL is a localization algorithm used for estimating the pose of a robot in an environment using sensor data, such as laser scans or depth images.

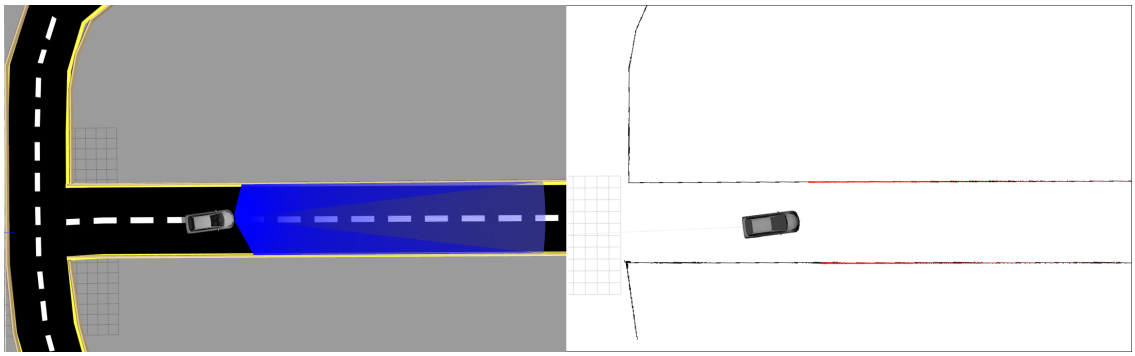


Figure 5.14: Robot Localization on the Map (Gazebo on the left, RVIZ on the right).

Another important component for the navigation is the **bt_navigator**, it is responsible for executing high-level navigation behaviors based on a behavior tree. The bt_navigator, along with the **planner** which is responsible for computing a path to a goal, and the **controller** which controls the effort of following the path, allows the robot to perform certain tasks like for instance navigating to a certain pose. In this case, the plugin used for the controller path following is the **Regulated Pure Pursuit** (Steve Macenski, Singh, et al. 2023), it is a variation on the Pure Pursuit controller, it regulates the linear velocities by curvature of the path to help reduce overshoot at high speeds around blind corners allowing operations to be much more safe, and also better follows paths than any other variation currently available of Pure Pursuit. While these components already allow the robot to autonomously navigate to a certain pose, there is still a vital component missing which allows it to avoid obstacles during navigation.

For the robot to safely navigate, it is not enough using only the static map. While the static map provides an initial layout of the surroundings, it lacks crucial information about dynamic obstacles that can impede the robot's path, rendering it incapable of dynamically adapting to the environment and avoiding obstacles. The purpose of the **Costmap 2D** is to overcome this issue, it is a two-dimensional grid representation of the environment, where each grid cell contains information about the occupancy and associated cost. It provides a spatial representation of obstacles, free space, and other relevant information that the robot needs to consider while planning and executing its motion. It is continuously updated based on the robot's sensor inputs, such as laser scans or depth images, to reflect the changing environment, which allows the robot to perceive changes in the environment and plan its

navigation accordingly. The Costmap 2D can have two separate instances used for different purposes in the robot's navigation process:

- **Global Costmap** - The global costmap represents the overall environment and is used for long-term or global path planning. The global costmap is typically larger in size and covers a broader area than the local costmap. It is updated less frequently compared to the local costmap and is primarily used for initial path planning.
- **Local Costmap** - The local costmap represents the immediate surroundings of the robot and is used for short-term or local path planning. The local costmap is smaller in size and covers a limited area around the robot. It helps the robot to perform obstacle avoidance, adjust its trajectory, and generate collision-free paths in real-time while considering both static and dynamic obstacles.

In Costmap 2D, multiple layers are used to represent different aspects of the environment. Each layer provides specific information that contributes to the overall understanding of the robot's surroundings. The default layers that compose the Costmap are the following:

- **Static Layer** - It represents static obstacles in the environment. This layer provides information about walls, buildings, and other fixed objects that the robot should avoid.
- **Inflation Layer** - It is responsible for inflating the obstacles. It assigns costs to the cells surrounding the obstacles, creating a buffer zone to ensure that the robot maintains a safe distance from them. This layer helps prevent the robot from getting too close to obstacles, considering its size and footprint.
- **Obstacle Layer** - The obstacle layer detects and incorporates information about dynamic obstacles into the costmap. It receives data from sensors such as laser scanners or depth cameras to identify and track moving objects in real-time. This layer is particularly important for avoiding pedestrians, vehicles, or other objects that can move within the robot's environment.

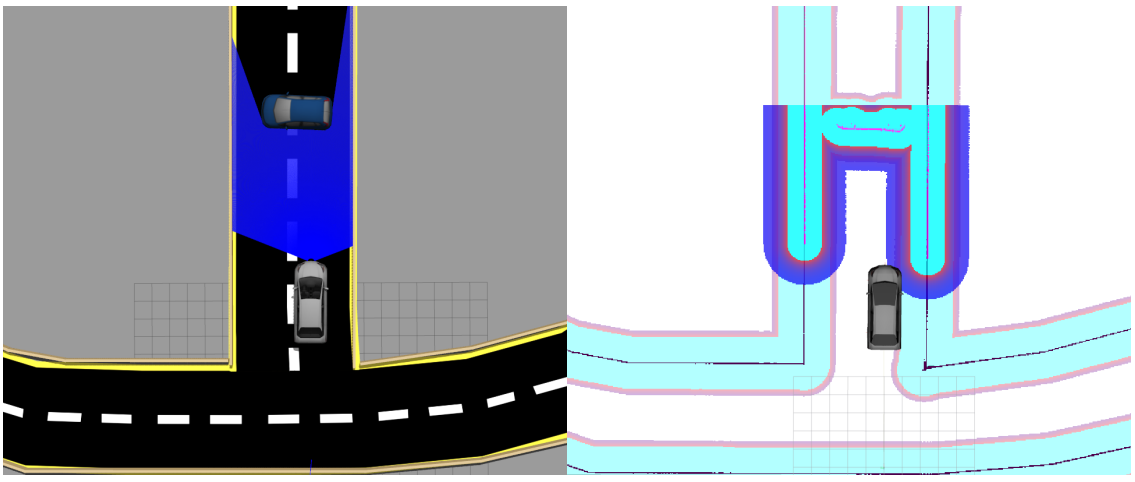


Figure 5.15: Costmap and Object Detection (Gazebo on the left, RVIZ on the right).

As depicted in figure 5.15, the global costmap has an inflation around the areas that are considered occupied, these areas are represented with a gradient from the free space color to the occupied space color. In this case, the inflation layer was configured so the robot

keeps a safe distance from the occupied areas, which prevents it from planning sharp turns around corners, and also better represents the robot size when planning a path around an object, which in the scenario that is visible in the figure, the detected object is a car that occupies the whole road making it impossible for the Prius to plan a path around it. In case the inflation layer was configured so the safe area around obstacles was smaller, the planner would plan a path around the car, which would be impossible to follow it given the size of the Prius. The darker blue area in the RVIZ visualization represents the local costmap, and, as stated previously, it convers a smaller limited area around the robot, whereas the light blue area represents the global costmap, and its size covers the whole map. Another important component for collision avoiding during planning is the robot **footprint**. The Prius footprint is represented by a green rectangle as depicted in the following figure:

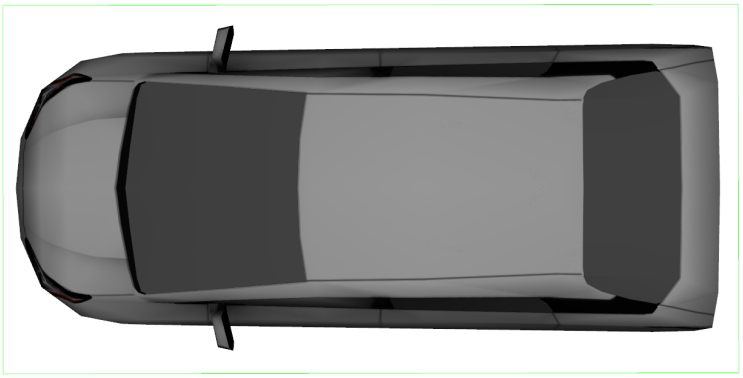


Figure 5.16: Robot Footprint Visualization in RVIZ.

The footprint outlines the robot's 2D shape when projected to the ground, this is mostly used by algorithms that make sure the robot footprint does not collide with the obstacles in the costmap while it computes the robot's paths or plans (Nav2 2020b). The footprint will also be useful for the approach used in the object detection data sharing, which will be explained later.

One of the notable advantages of the Navigation 2 framework in ROS2 is its extensive use of topic-based communication for sharing and accessing information. By employing a publish-subscribe messaging model, Navigation 2 enables easy dissemination of crucial data across the robotic system. Various components, such as the global and local planners, the costmap, and the sensor inputs, publish their relevant information on designated topics. This approach fosters a modular and flexible architecture, allowing different modules to subscribe to the topics of interest and retrieve the necessary information for their respective tasks. Additionally, this topic-based communication paradigm facilitates seamless integration with other ROS2 packages and modules, as they can easily access the published information for collaborative decision-making or data fusion. To facilitate data exchange, there are several message types used by Navigation 2 which improve accessibility to the relevant information for each navigation task.

Object Data Sharing

As previously explained in the micro-ROS section, the purpose of the sensing devices is to share the object data obtained from the laser scan, with the vehicle. This requires a method capable of sending the data using an adequate message type that contains valid obstacle

data, and it is also necessary for the vehicle to know the pose of the origin of the data so the received obstacle data is correctly positioned in relation to the perspective of the vehicle. In order to achieve this outcome, we explored two possible approaches, a distributed approach and a centralized approach.

The distributed approach consists in directly sending the LaserScan message from the sensing devices to the vehicle. This means that each device present in the environment will send a message to the vehicle containing its own obstacle data.

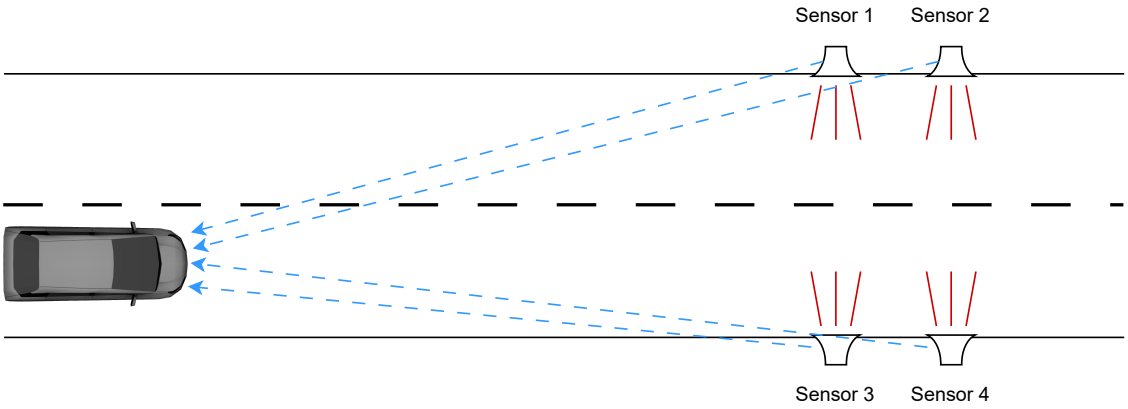


Figure 5.17: Distributed Approach Scenario.

The vehicle will then receive the LaserScan messages, and then add its data to the costmap, but there is still an issue, which is knowing the pose of the origin of the data. LaserScan messages do not contain information about the pose of the data, it only contains the frame id of origin of the data, which only works when accessing the data within the robot itself. In this case, the vehicle is accessing sensor data from another source which is technically another entity, which means that it does not know any of the frames from the other entity, resulting in the mispositioning of the obtained laser data which will be interpreted as if it is placed in front of the vehicle. The solution is applying a **transform** from the vehicle to each sensing device/entity.

In ROS2, a transform refers to the spatial relationship between two coordinate frames. It represents the translation and rotation between different frames of reference in a 3D space. Transformations are crucial for handling data from different sensors or robot components that are attached or moving relative to each other, this allows for seamless data integration and coordination between different parts of a robotic system. ROS2 provides **tf2**, which is a transform library that manages and publishes the transform information. It allows nodes to access and use the transforms relevant to their tasks, ensuring consistent spatial relationships are maintained throughout the system. The transform information is typically broadcasted over the tf2 topic, allowing nodes to subscribe and apply the appropriate transformations as needed.

In order for the Prius to correctly place the received laser data from the sensors, it was developed a node that utilizes the tf2 library to publish the transforms from the Prius to each spawned sensor. Since the transforms require the position of each sensor, the node subscribes to the message **/model_states** published by Gazebo, which contains information about all the models present in the Gazebo simulation, this information consists in position, orientation, linear and angular velocities, and other relevant attributes of each model. The

sensor data also requires orientation for it to be valid, and thus, it is necessary two transforms, one to the location of the sensor, and the other applies the rotation on the respective location.

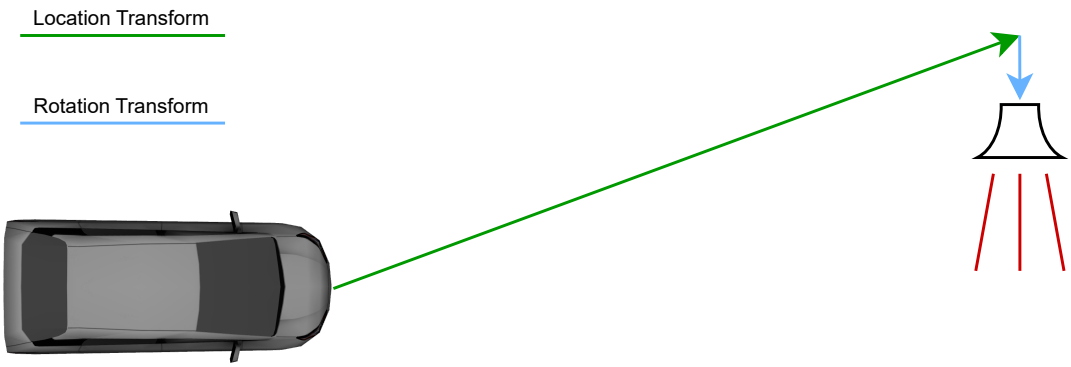


Figure 5.18: Transform from the Vehicle to the Sensor.

Thanks to the transform, the sensor data is correctly positioned so it can be used for the obstacle detection in Navigation 2. In order for the obstacle data to be added to the costmap in Nav2, it is necessary to create an obstacle layer for each sensor, and in this case, the layers were created in the global costmap since it covers a longer area compared to the local costmap. Although this approach allows for obstacle data sharing from many sensing devices with the vehicle, having the vehicle constantly receiving multiple messages is not efficient.

The focus of the second approach was to find an efficient method that shares a single message that merges all data from the sensors. This could be achieved by having another entity, working as a data aggregation device, running the navigation stack in order to merge the obstacle data in a single message that already contains the costmap data, which is an **OccupancyGrid** message. The OccupancyGrid, which is the message type used by the Costmap, is a fundamental data structure used to represent and store information about the environment's occupancy. It serves as a 2D grid-based map where each cell represents the probability of occupancy at a specific location, allowing the navigation system to reason about obstacles, free space, and unknown areas. The OccupancyGrid message type is composed as follows:

- **Map metadata** - This includes information about the map's dimensions, resolution, origin, and frame ID.
- **Grid data** - The grid data represents the occupancy probabilities of individual cells in the map. Each cell is typically represented by a single value ranging from 0 to 100, indicating the likelihood of occupancy. Common conventions include 0 for free space, 100 for occupied space, and intermediate values for uncertain or unknown regions.

With this method, the vehicle only needs to receive a single OccupancyGrid message that contains the obstacle data from all the sensors in that area. The data is added to the vehicle costmap using a static layer, which is the only default layer capable of receiving another costmap using an OccupancyGrid message.

The only inconvenience with this approach is that micro-ROS is not capable of running the Navigation 2 stack, which means that it is not possible to use one of the sensing devices to

merge the obstacle data using Nav2. The workaround is to add a device capable of running the Nav2 stack in the same area as the sensing devices. In a real scenario, this device could be a kind of a local gateway or RSU with just enough computational capacity to run ROS2 and Navigation 2. The device would be treated as a robot whose purpose is to receive the LaserScan messages from the sensors, merge them into an OccupancyGrid message using Nav2, and sending the message to the vehicle. This method would still use the same logic from the first approach, which is creating a transform from the device to each sensor and creating the obstacle layers for its costmap.

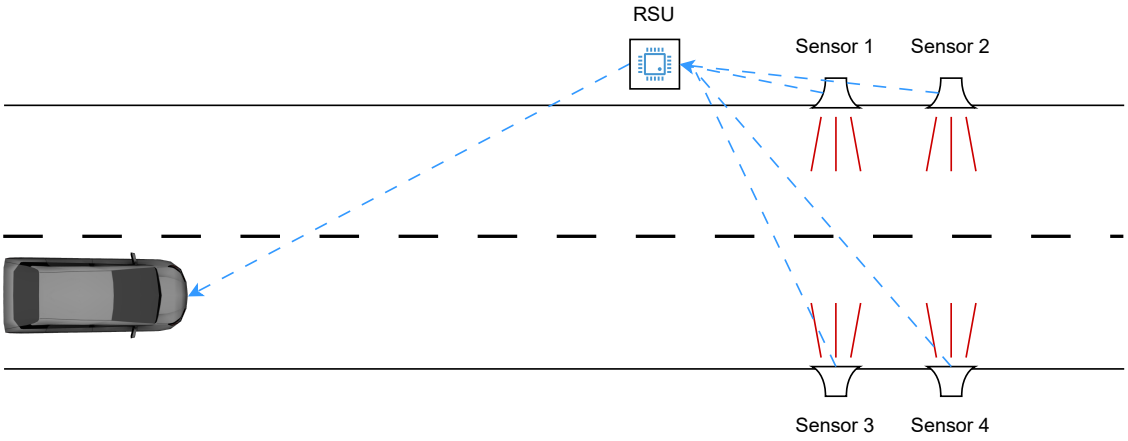


Figure 5.19: Centralized Approach Scenario.

As represented in the figure 5.19, the device functions as a RSU, whose function is gathering obstacle data from the sensors assigned to a certain area, and broadcast the merged data to the approaching vehicles. In a real world scenario, it would probably be necessary a RSU for each group of sensors which could be distributed by each road, and the number of RSUs would depend on the distance and quantity of sensors.

Regarding the simulation of this approach, it was added another model in the Gazebo Environment. The 3D model representation is the same as the hokuyo model used for the sensors, as for its properties, it has the necessary entities to be able to use the Nav2 stack and the same configuration and map from the Prius so that both costmaps match in size and properties.

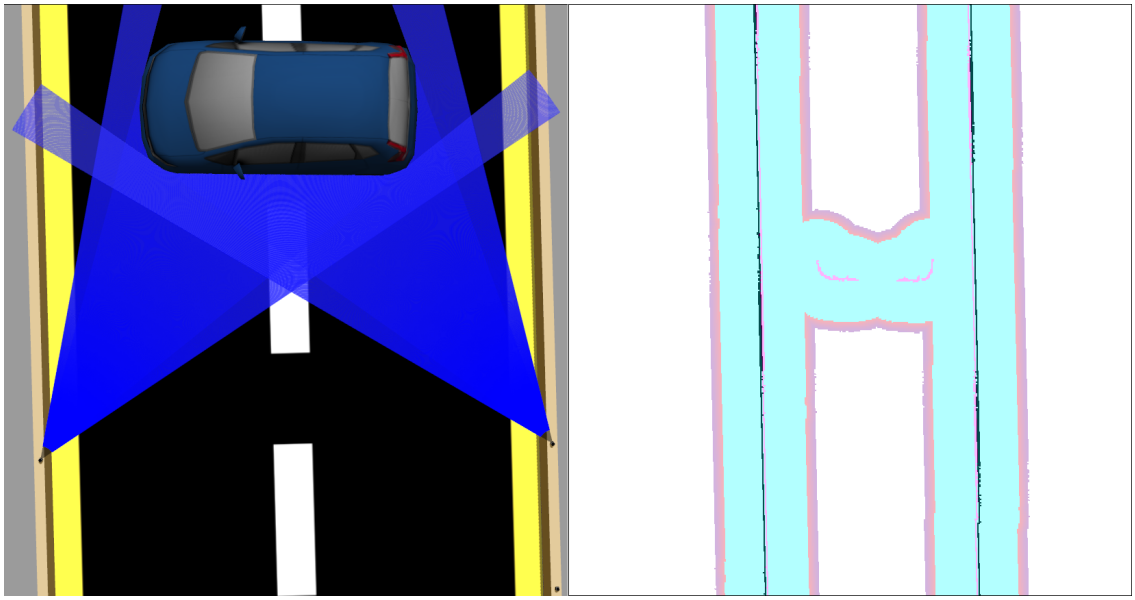


Figure 5.20: Second Approach Scenario Simulation (Gazebo on the left, RVIZ on the right).

In figure 5.20, it is depicted the simulation running in Gazebo where two sensors and a RSU which is placed in the bottom right corner of the Gazebo view, detect a vehicle that is obstructing the road. The Prius then subscribes to the global costmap topic generated by the RSU Nav2 instance, and merges it with its own costmap. And, as seen in the right of the figure, the Prius costmap contains the obstacle data provided by the sensors, which makes it aware of the obstacle without even approaching its location.

While the sensors are able to correctly detect obstacles, there is still an issue, what if the Prius itself passes through the sensors? The answer is that it will be detected as an obstacle, and therefore, it will be also added to the RSU costmap, so when the Prius receives the OccupancyGrid message, there will be an obstacle at its immediate position on the map, which will result in the navigation to be blocked without being able to find an alternate path since the surrounding area of the Prius is occupied in the costmap, as seen in figure 5.21.

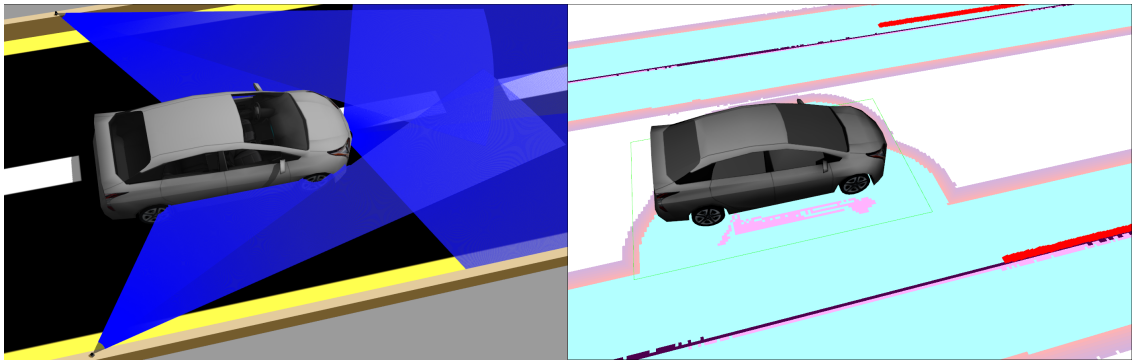


Figure 5.21: Prius Blocked by Itself (Gazebo on the left, RVIZ on the right).

In response to the aforementioned issue, a viable workaround was devised, which utilizes the robot footprint. As previously mentioned, the robot footprint represents its shape on the map, which means that the area occupied by the robot in the map is represented by the

area of the shape. So, the solution is to ignore the occupied cells that are identified within the footprint area, which will remove the Prius obstacle representation from the costmap. The information regarding the footprint is obtained by subscribing to the topic **"/prius/-global_costmap/published_footprint"**, and its message type is **PolygonStamped**.

This message consists of two main fields, header and polygon. The header field contains metadata about the message, such as the frame ID and timestamp. And the polygon field contains the actual geometric representation of the polygon, defined by its vertices. In this case, since the Prius footprint representation is a rectangle, the PolygonStamped message contains the coordinates for all 4 vertices. Since the frame ID is the map, the polygon's coordinates are expressed in the absolute world coordinates defined by the map frame. The map frame represents a fixed, global coordinate system that serves as a reference for other frames within the robotic system. Therefore, it is required to convert the footprint coordinates to the 2D grid representation of map, which is the Costmap.

In the case of the Gazebo world previously demonstrated, the map has a width of 2783 and a height of 1216, and the objective is to locate the footprint within the costmap coordinates as depicted in figure 5.22. Another aspect to consider is that the costmap data array is one-dimensional, meaning that not only it is necessary converting the footprint coordinates to the costmap, but also converting them into a 1D array which will have a size of 3384128 cells.

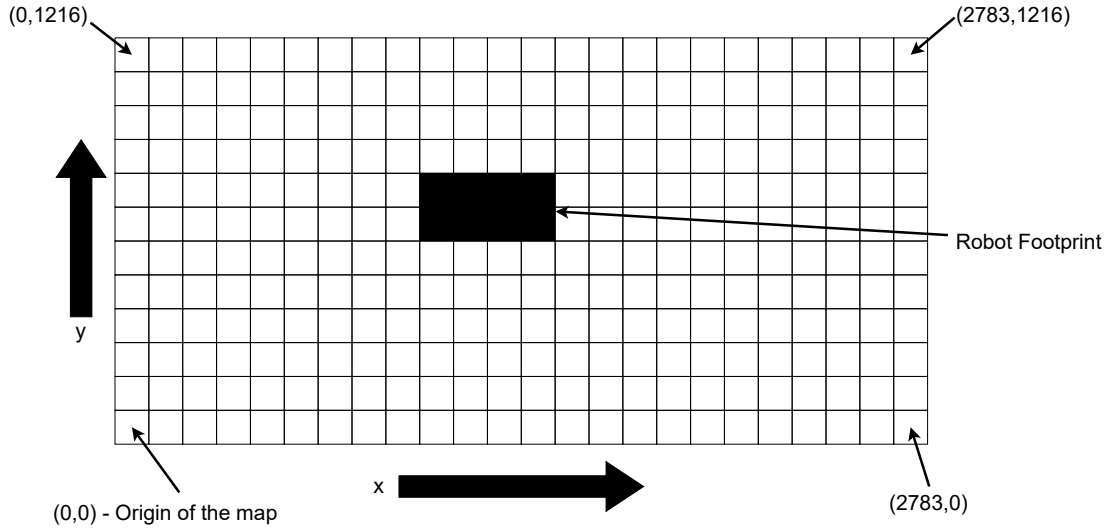


Figure 5.22: Costmap Grid.

The coordinate conversion of a certain point depends on the coordinate values, the origin of the costmap in world coordinates, and the costmap resolution, which is the size of each grid cell in the costmap. The costmap coordinate values can be calculated with the following:

$$costmap\_x = (point\_x - costmap\_origin\_x)/map\_resolution \quad (5.1)$$

$$costmap\_y = (point\_y - costmap\_origin\_y)/map\_resolution \quad (5.2)$$

Subtracting the point coordinates with the costmap origin aligns them with the costmap. Dividing the point coordinates by the map resolution scales down the world coordinates to match the costmap's grid cell size. The obtained values are then converted to integers, since the data type of the OccupancyGrid data array is Int32, and their 1D index in the data array is obtained as follows:

$$costmap\_1D\_index = costmap\_y \times costmap\_width + costmap\_x \tag{5.3}$$

The costmap_y represents the row index of the desired cell in the costmap, the width refers to the width of the costmap, which is the total number of columns in the 2D grid. Multiplying both values, gives the total number of cells in the preceding rows before the desired row. Since the costmap_x represents the column index of the desired cell, adding it to the result from the previous step gives the final index of the desired cell in the 1D array.

In the case of the footprint, as it is a rectangle, this logic needs to be applied to all cells in its area which is limited by its vertex coordinates. For this purpose, it was created a node that subscribes to both the footprint message and the OccupacyGrid message provided by the RSU, and iterates over all cells within the area occupied by the footprint to remove the occupancy from the cells, which is setting the costmap value to 0. It is then created a new OccupancyGrid message with the cleared costmap and published to a new topic that is subscribed by the Prius Nav2 static layer.

As depicted in figure 5.23, the area surrounding the footprint is now clear due to the occupied cells being located inside the rectangle, and the path is no longer blocked, as opposed to the RSU costmap which still detects the Prius as an obstacle.

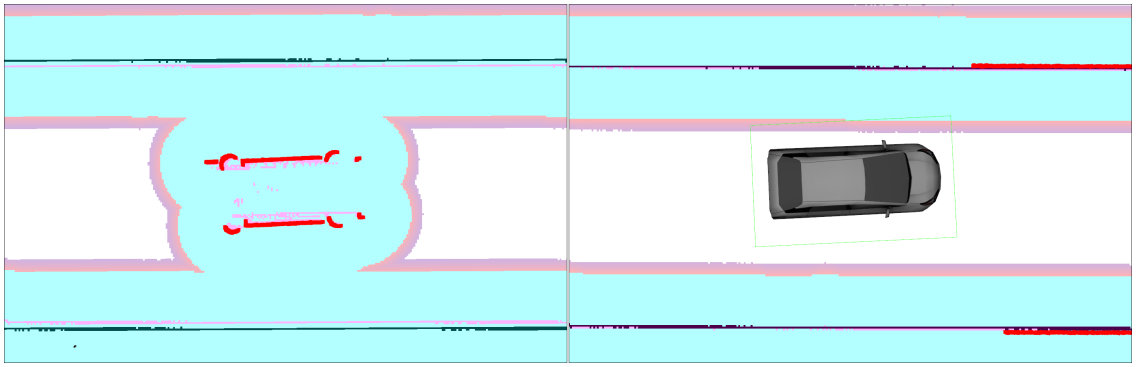


Figure 5.23: Costmap is cleared within footprint area (RSU costmap on the left, new Prius costmap on the right).

5.2.2 ITS Communications Simulation

This section deliberates on the simulation of communications within the context of our system. Effective communication is a critical aspect of any autonomous robotic system, enabling seamless coordination and information exchange between different components. In our previous setup, the communication between components was instant, which provided a simplified view of the system's behavior. However, in a real-world scenario, communication is subject to various constraints such as latency, bandwidth limitations, and signal degradation. To address these real-world challenges and gain a more realistic understanding of

our system's performance, we will employ OMNeT++, a powerful network simulation framework. By integrating OMNeT++ with our robotic system, we can emulate the behavior of communication networks and evaluate the impact of different communication parameters on the system's overall performance.

Regarding the approach implemented in this work, it was followed the same approach utilized by AuNa, which uses **Artery**. Artery, an extension of the OMNeT++ simulation framework, offers an implementation of the ETSI ITS-G5 communication architecture. This architecture enables the simulation of communication between onboard units (OBU) in vehicles and RSU within the infrastructure. Artery comprises various components, including the physical layer and MAC layer from Veins, the transport and networking layers from Vanetza, and a configurable application layer connected through a managing middleware. By leveraging Artery, modular applications like platooning can be implemented, and other frameworks can be seamlessly integrated into OMNeT++ through the corresponding layers of Artery. In the context of C-ITS scenarios, Artery provides services in the application layers. These services implement the communication rules necessary for message transfer. Notably, Artery natively supports the Cooperative Awareness (CA) basic service defined by the ETSI ITS-G5 standard. This service facilitates the automatic broadcasting of Cooperative Awareness Messages for communication with other road users (Teper et al. 2022).

Regarding the integration of the ROS2 framework with Artery/OMNeT++, the approach used is also based in AuNa, which will be covered in the following steps (Teper et al. 2022). The first step involves establishing the connection and synchronization between OMNeT++ and other simulations using ROS2 nodes. To facilitate this, a custom scheduler is implemented within OMNeT++ that manages the execution of OMNeT++ events, synchronization between OMNeT++ and Gazebo, and updates between the ROS2 navigation system and the communication module in OMNeT++ for each robot. The scheduler processes the functions of the ROS2 node, receiving the current simulation time and updating robot states. It schedules OMNeT++ events based on their time, ensuring they occur at the appropriate moment within the robot simulation.

The second step focuses on creating communication modules for each robot. The implemented node requests and receives the current state of the Gazebo simulation. OMNeT++ checks if a corresponding module exists for each robot and spawns new modules as needed. This ensures that a suitable OMNeT++ module is dynamically created for each robot system without unnecessary duplication.

The final step involves integrating the communication modules with their respective navigation systems. The mobility module in Artery is extended to receive the estimated state of the robot, encompassing position, velocity, acceleration, heading, and yaw rate. Following the architecture of Artery, the mobility module notifies other components such as the middleware and vehicle data provider, ensuring their updates align with the received information. This integration enables seamless communication. For specific applications like the platooning scenario, an additional service is implemented in the application layer. This service forwards messages from the leading vehicle to the navigation system, facilitating coordinated actions.

In the case of this work, the scenarios being simulated are based on the communication between the RSU and the vehicle, and thus, the platooning services are not used. Since in the approach used for the RSU interprets it as a robot, its integration with Artery/OMNeT++ using the previously mentioned method was seamless, meaning that, it was only necessary

to perform slight changes to the ROS2 nodes and some of the Artery components, which were both introduced by AuNa.

On the side of the ROS2 package, the node responsible for sending the robot data to Artery/OMNeT++ is the **omnet_transmitter**. This node receives the robot odometry data to obtain the speed, acceleration, yaw rate and curvature, and also receives the robot pose. With the data obtained, the node then creates a CAM message which is composed as follows:

- **Header** - Contains the robot id, message id, and timestamp.
- **Station Type** - Contains the station type, ex. passenger car.
- **Reference Position** - Contains longitude, latitude and altitude.
- **BasicVehicleContainerHighFrequency** - Contains high-frequency basic information about the sending vehicle, providing essential details that need to be frequently updated. This information consists in:
  - **Speed** - Vehicle speed.
  - **Heading** - The orientation or direction of the vehicle.
  - **Acceleration** - The rate of change of the vehicle's velocity.
  - **Vehicle Size** - Information about the dimensions or size of the vehicle.
  - **Curvature** - The rate at which the vehicle's trajectory deviates from a straight line.
  - **Yaw Rate** - The rate of change of the vehicle's heading or orientation over time.

The CAM message is then published to a topic called **cam_out** which is subscribed by one of the Artery/OMNeT++ modules responsible for the ROS2 integration. Since, in the context of this work, the focus is to share the OccupancyGrid between the RSU and the vehicle, it was added a new field to the CAM message that contains the OccupancyGrid data array, meaning that now the entities will not only share the previous mentioned data, but also share their costmap.

On the side of Artery/OMNeT++, each created module for representing each robot now contains the OccupancyGrid data array aswell, and it was also added in the CAM message that is sent in the ETSI ITS-G5 communication simulation. During the communication simulation, when a message is received by the vehicle, it is published a new message to the topic **cam_in** so it can be accessed by ROS2.

Finally, in the ROS2 Package, there is a node belonging to the vehicle, which subscribes to the CAM message and performs the previously mentioned method of clearing the costmap with the robot footprint, and creates a new OccupancyGrid message to add to the costmap.

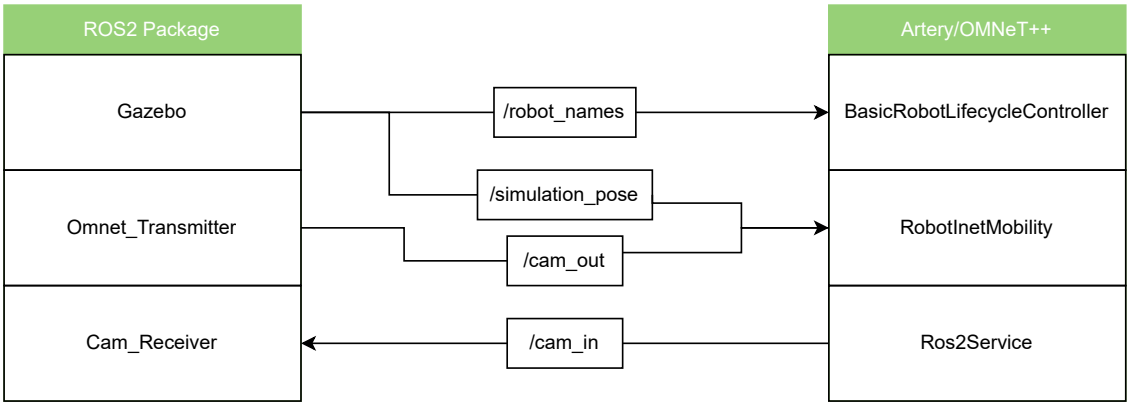


Figure 5.24: ROS2 and Artery/OMNeT++ Integration.

During the simulation runtime, with the OMNeT++ Qtenv GUI it is possible to control the simulation with actions such as start, stop, pause, resume, and reset the simulation. It is also possible to control the simulation's execution speed, step through events, and manage simulation time. Regarding network visualization, it provides visualization capabilities to view the network topology, including the nodes, connections, and links.

In figures 5.25 and 5.26 is depicted a communication simulation testing between the RSU and the Prius. As stated previously, a communication module is created for each robot and the name assigned is "robot[<id>]", which makes the module creation dynamic. In figure 5.26, it is visible 2 modules for the existing robots in the environment, robot[0] represents the Prius, and robot[1] represents the RSU. Both modules belong to the "World" module which represents the simulated environment in which the vehicular network operates, and serves as a container for various components and modules that define the characteristics and behavior of the simulated world. The **radioMedium** represents the simulated wireless communication medium within the vehicular network, it models the propagation of radio signals between each unit based on specified communication parameters and channel characteristics. The **coordinateSystem** component defines the coordinate system used in the simulation environment. It establishes a reference frame for positioning and mobility-related operations. The coordinateSystem can be configured based on real-world coordinate systems like latitude-longitude or any custom coordinate system suitable for the simulation scenario. Thanks to the information regarding each model position provided by Gazebo, the positioning of the robots within the world canvas in the GUI also approximately represents the actual position in the environment as seen in figure 5.25.

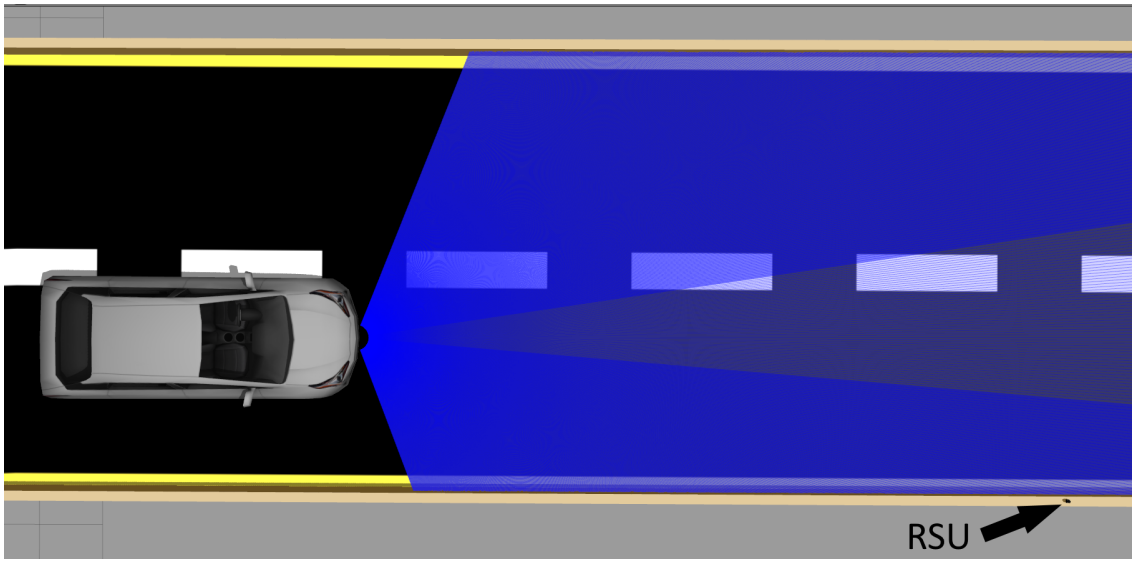


Figure 5.25: Robots Positioning in the Environment.

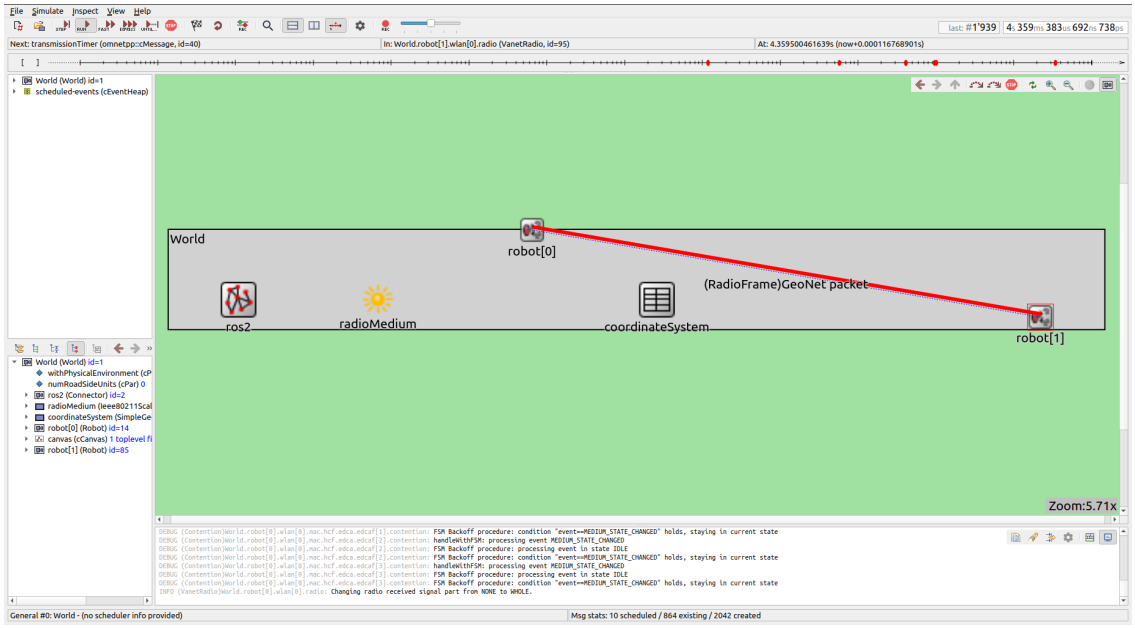


Figure 5.26: OMNeT++ Qtenv GUI.

The diagram below illustrates the process of sending a message from the RSU to the vehicle. It begins with Gazebo initiating the process by sending a LaserScan message, adapted to micro-ROS compatibility, containing obstacle data. Subsequently, micro-ROS nodes subscribe to this message and relay it to a new topic. The RSU's navigation system then utilizes the LaserScan data to populate the costmap obstacle layer. Following this, an OccupancyGrid message is transmitted to the communication node, known as omnet_transmitter. Within the omnet_transmitter node, the data is transformed into a CAM message using the OccupancyGrid information and sent to the corresponding OMNeT module representing the RSU. Inside OMNeT, the RSU transmits the CAM message to the vehicle via ITS-G5

communication. Subsequently, the vehicle module forwards the received message to the vehicle's communication node responsible for CAM message reception, known as cam_receiver. The cam_receiver processes the OccupancyGrid data, applies footprint clearing, and subsequently generates a new OccupancyGrid message. This message is then dispatched to the vehicle's navigation system, where it finally obtains the obstacle data.

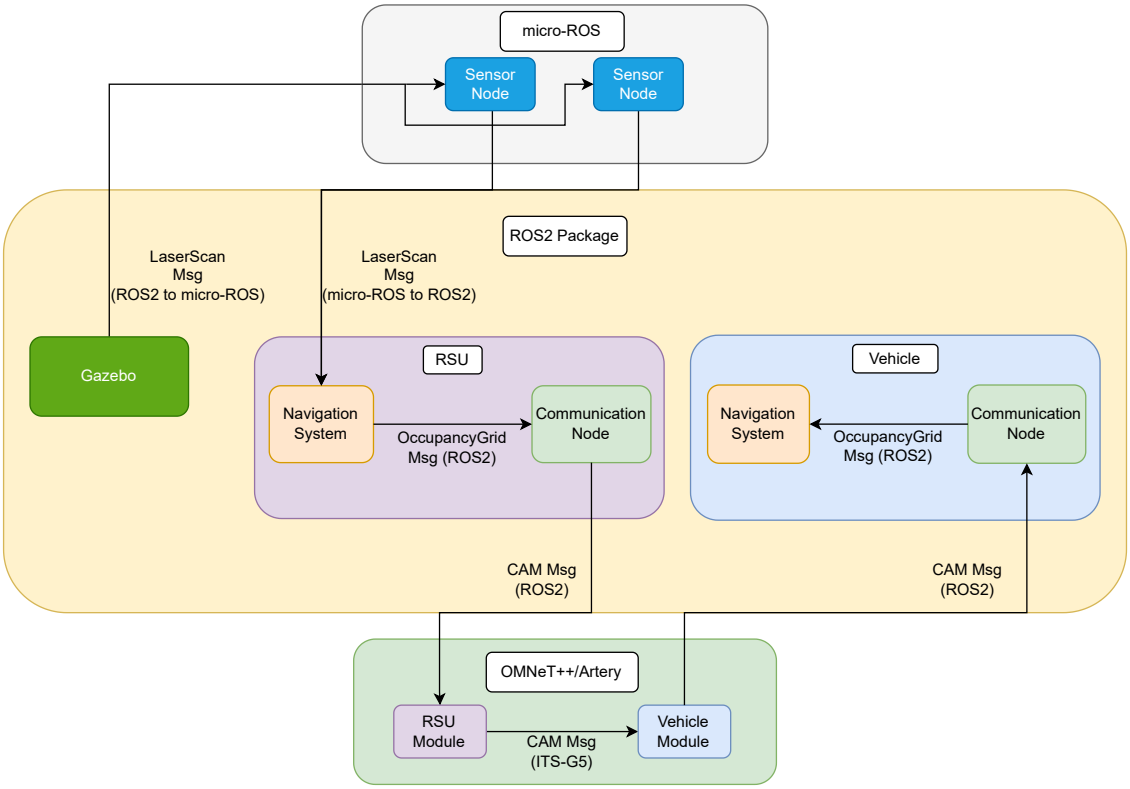


Figure 5.27: RSU to Vehicle Communications Diagram.

Chapter 6

Experimental Results

This section analyses the results of the experiments performed during the testing and validation of the scenarios. It serves to validate and evaluate the implemented methodologies and solutions, assessing their performance, accuracy, and practical implications. By examining the results, we gain insights into the effectiveness, reliability, and scalability of the research, enabling informed conclusions and further discussions.

All the testings and obtained timing values are performed using Gazebo simulation time. The Gazebo simulation time, which is distinct from the system time provided by the operating system, refers to the virtual time that progresses within the Gazebo simulation environment, it represents the passage of time in the simulated world, allowing the simulation to progress at a specified rate. As previously mentioned, the used approach for the OMNeT++/Artery and ROS2 integration synchronizes the OMNeT++ and Gazebo simulation time using the Gazebo clock. This allows for consistent time references across both simulation environments, enabling accurate analysis of communication delays between simulated entities. The tests were conducted utilizing a computer equipped with an NVIDIA GTX 1050 GPU and an Intel I7-7700HQ CPU.

6.1 Implemented Scenarios

6.1.1 Road Block Scenario

The first scenario is the previously introduced preliminary scenario which consists in the vehicle receiving obstacle information about a blocked road that belongs to its planned path so it can calculate a new path to perform a detour without having to approach the obstacle.

To facilitate the setup of a scenario, the ROS2 package has a scenarios folder which contains the launch files used for starting the simulation. For each scenario, there is a launch file for each component which are the Prius, the RSU, and the sensors. The launch file contains the namespace of the robot, its spawning position in the Gazebo world, and in the case of the Prius and RSU, it also contains parameters for launching Nav2, such as the used map file and configurations file. The namespace allows for the dynamic creation of the robots in the simulation environment, it is used for naming each robot frame and topics created by the robot, which allows the creation of multiple robots while using the same URDF files. Regarding micro-ROS, since the developed micro-ROS app is located in the micro-ROS workspace, it is launched separately from the ROS2 package, but thanks to the previously discussed method of using environment variables, it is simple to create a micro-ROS node for each spawned sensor in the environment by using their IDs.

The representation of the scenario in Gazebo is depicted in the following figure.

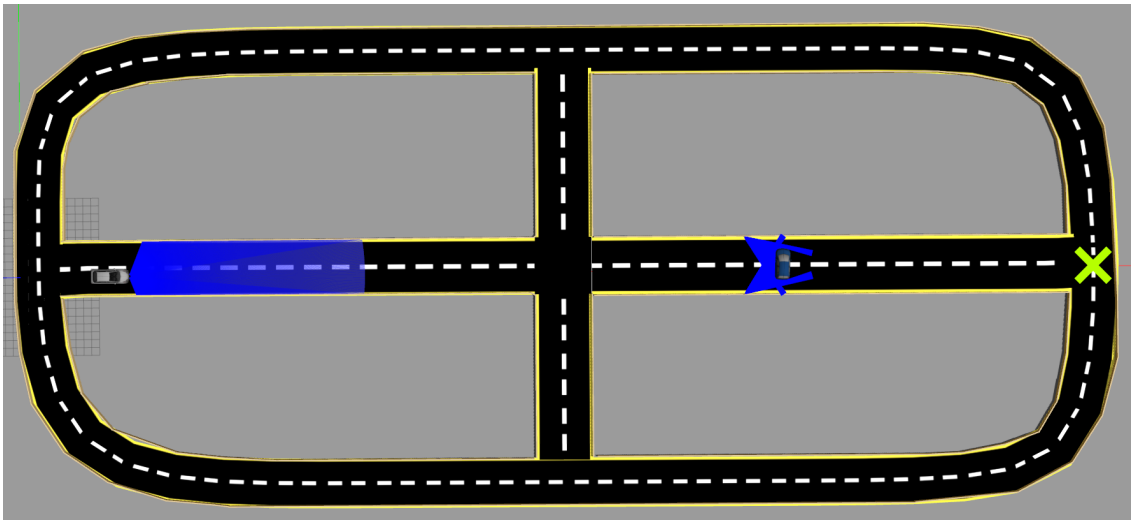


Figure 6.1: Gazebo Representation of the First Scenario.

As displayed in the figure, each robot is located in its spawning position, and there is a car blocking the road, which is in the line of sight of the 2 sensors present in the scenario. The goal is for the Prius to autonomously drive to the **"X"** marked on the figure using the Nav2 navigate to waypoint function, and the shortest path for the Prius to follow is simply going straight. Without receiving the data from the RSU, the vehicle navigation system will generate a path to make it go straight to the goal, but as soon as its sensor detects the obstacle, it will generate a new path in an attempt to make a u-turn which will be impossible to achieve given that there is not enough space for the Prius to perform such maneuver, and also the Nav2 algorithms used for the path planning and following do not allow for the robot to go in reverse.

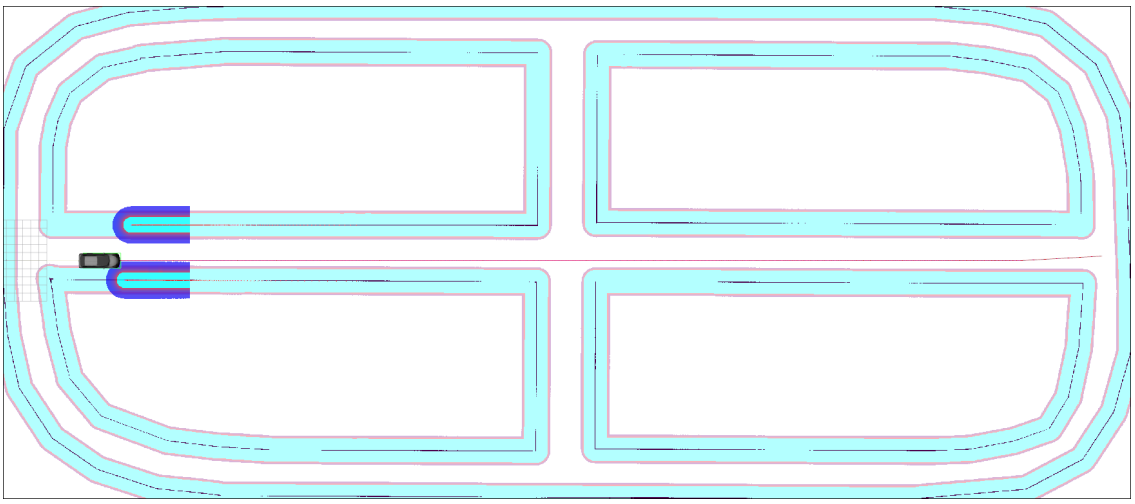


Figure 6.2: Path Generated without RSU.

With the aid of the RSU and its sensors, the expected outcome is for the vehicle to obtain the awareness of the position of the obstacle, which will allow the generation of a new path in advance, and thus, it will be able to take a detour to the waypoint that will avoid the blocked road.

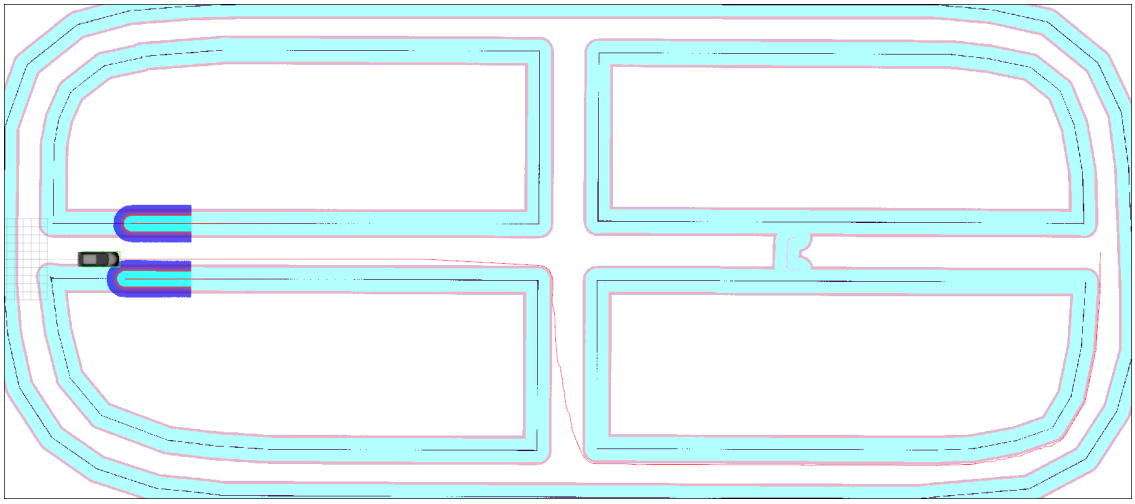


Figure 6.3: Path Generated with RSU.

As illustrated in figure 6.3, thanks to the RSU and the sensors cooperation, the vehicle costmap now contains the car that is blocking the road, allowing it to create a new path as represented by the line in the figure, which achieves the expected outcome.

This scenario showcases the usefulness of having the perception closer to the environment, as it proves that without the help of the sensors, the vehicle would not have been able to react to such scenario.

6.1.2 Dynamic Obstacle Scenario

For this scenario it was experimented a new Gazebo world that represents a small city. The generation of its map was done with the same procedure using the SLAM toolbox as covered earlier.

Figure 6.4: Second Scenario World in Gazebo and the Created Map Respectively.

As visualized in figure 6.4, the new Gazebo world is significantly bigger and contains several models to portray a city. The only downside of this world is being computationally more demanding, which might affect performance during the simulation.

In this scenario it was explored the detection of a moving obstacle in the environment in order to test the detection ability of the sensors, and analyse whether the vehicle is able to react in time.

In order to enable moving obstacles in the Gazebo environment it will be necessary to use one of its features for creating animated models. These entities are called **Actors**, they extend common models by adding animation capabilities, and are able to follow predefined paths in simulation without being affected by the physics engine. Actors in Gazebo are designed with unique properties that distinguish them from other objects in the simulation. Unlike

conventional objects, Actors are not affected by gravity or subject to collisions with other objects. However, they have 3D visualization, making them visible to RGB cameras, and they also possess 3D meshes that can be detected by depth sensors relying on GPU-based technology (Open Source Robotics Foundation 2014).

In the case of this scenario it was used an Actor that represents a pedestrian which was configured to cross the road. With the configured speed of the animation, the pedestrian takes approximately 5 seconds to cross the road, and after another 5 seconds the animation is reset so it can cross again.

Figure 6.5: Pedestrian Actor Crossing the Road.

Since Actors are only detected by GPU based sensors, it was necessary to edit the sensors URDF file to change the laser sensor type to "gpu_ray", which also makes it unable to detect the collision model of an object. This also means that it would also be necessary to change the laser sensor type of the Prius for it to be able to detect the pedestrian, but changing its type would affect the precision of the vehicle localization on the map which uses AMCL. On the other hand, it will also be useful for the scenario, given that it will be possible to test the detection of the pedestrian while relying only in the road sensors.

Regarding the layout and composition of the scenario, it was used 2 sensors which were placed in each side of the road and in common position of a crosswalk which is right next to a crossroad. The idea is to test the accuracy of the received data in case the vehicle were to turn left on the crossroad, which would require avoiding hitting the pedestrian that is crossing the road, by using only the obstacle data received from the sensors and the RSU. This will allow us to analyse if the communication speed is adequate for the vehicle to react in time in these types of scenarios.

The following figures 6.6 and 6.7 depict the scenario layout in Gazebo.

Figure 6.6: Second Scenario Pedestrian and Sensors Layout.

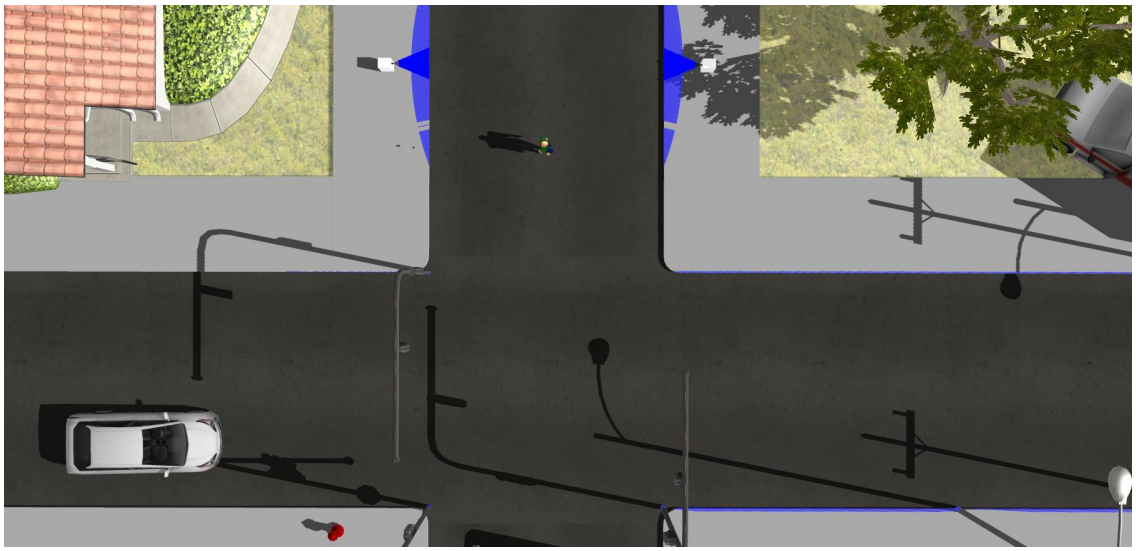


Figure 6.7: Second Scenario Overview.

With the sensors and RSU up and running, it was first tested if the costmap is able to dynamically add and remove obstacles from the map when they are no longer being detected, and thus, it was only necessary to use the RSU costmap. In figure 6.8 it is visible the RSU costmap reaction to the pedestrian crossing the road, which is using the LaserScan info from the micro-ROS nodes to fill the obstacle layer. The red dots represent the actual LaserScan data by subscribing directly to the topic provided by the sensors. This comparison allows us to conclude that obtaining the data from the micro-ROS nodes introduces a delay, which affects the accuracy of the data, meaning that the costmap occupied obstacle cells will not accurately match the actual position of the pedestrian.

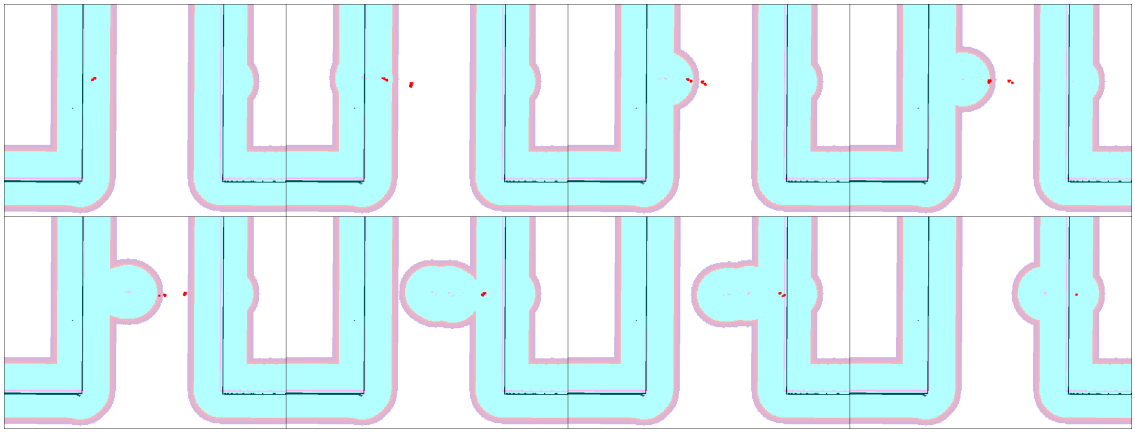


Figure 6.8: RSU Detecting the Pedestrian Using micro-ROS.

To validate the previous conclusion, it was also tested the same scenario without relying on the data provided by the micro-ROS nodes, i.e, using instead the LaserScan data directly from the sensors to fill the obstacle layer. As illustrated in figure 6.9, it is confirmed that the LaserScan positioning is now more accurate regarding its representation in the costmap.

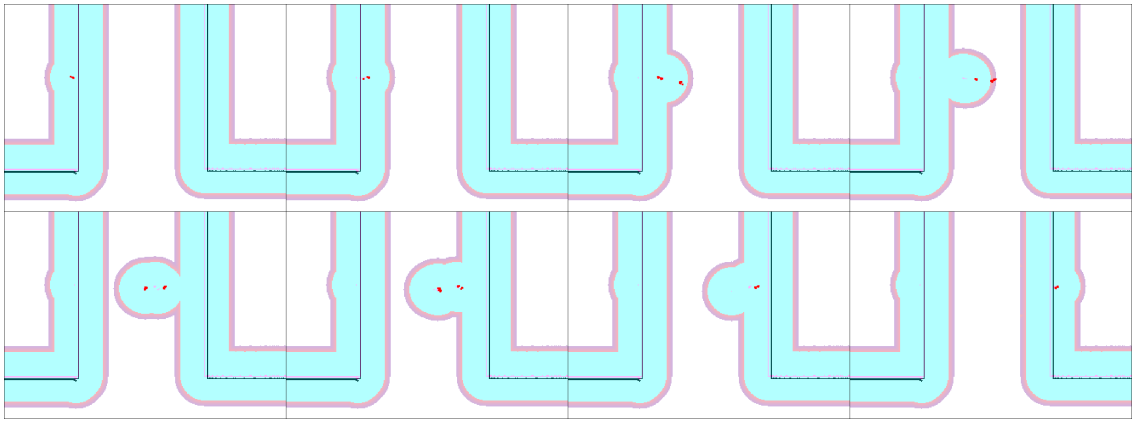


Figure 6.9: RSU Detecting the Pedestrian Without micro-ROS.

The reason for this delay is due to the previous explained issue in section 5.1, which is the usage of a Reliable QoS stream for micro-ROS LaserScan messages, which in turn increases latency. The need of using a Reliable QoS stream is due to the inability for micro-ROS to perform fragmentation on Best-Effort message streams, meaning that it is not possible to serialize the LaserScan data in a single stream due to the MTU size.

Under normal circumstances, the previous mentioned workaround which consists in reducing the message size by reducing the number of samples and resolution of the laser sensor, should be a adequate solution, given that in this case it is not required to accurately obtain the shape of the object to add it to the costmap so it can be avoided. But unfortunately, the costmap obstacle layer is not capable of clearing the occupied cells by the pedestrian when it is no longer detected.

In figure 6.10, it is depicted the RSU costmap using the micro-ROS data with a Best-Effort stream that required reducing the message size. It is visible the previous mentioned issue where the path behind the red dots that represent the pedestrian is still occupied.

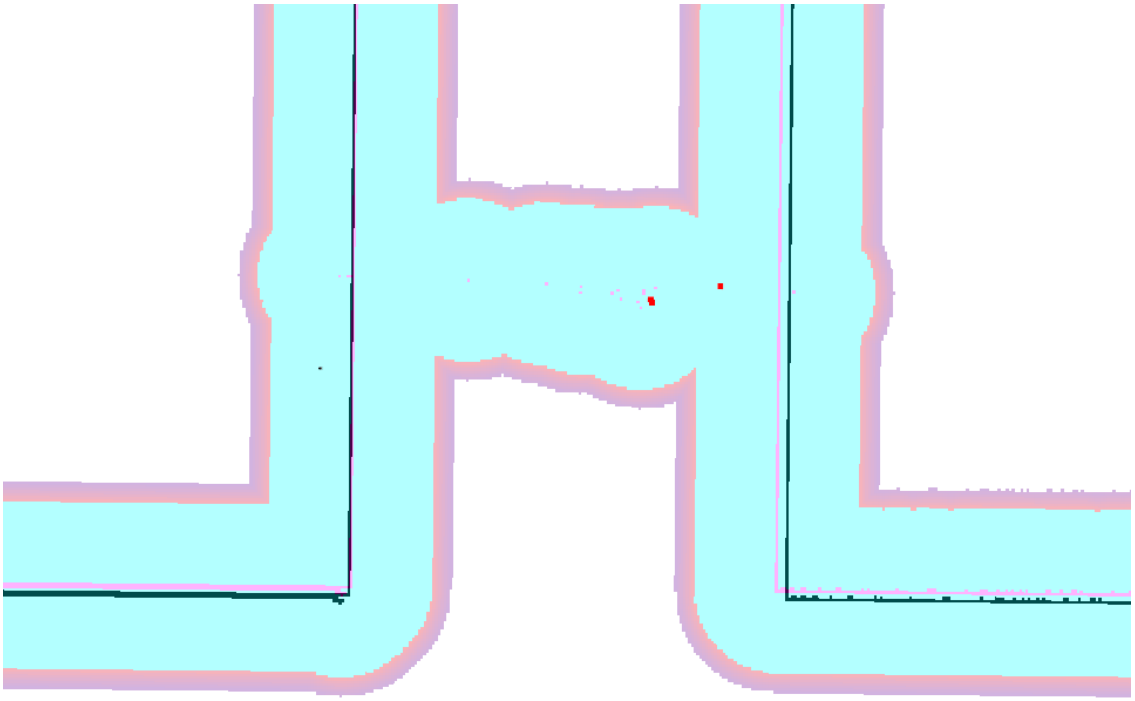


Figure 6.10: RSU Costmap using micro-ROS node and Best-Effort QoS.

Since the previous tests were performed without the communications simulation, it was also tested the obstacle data sharing with the vehicle using Artery/OMNeT++ and using a Reliable message stream for micro-ROS.

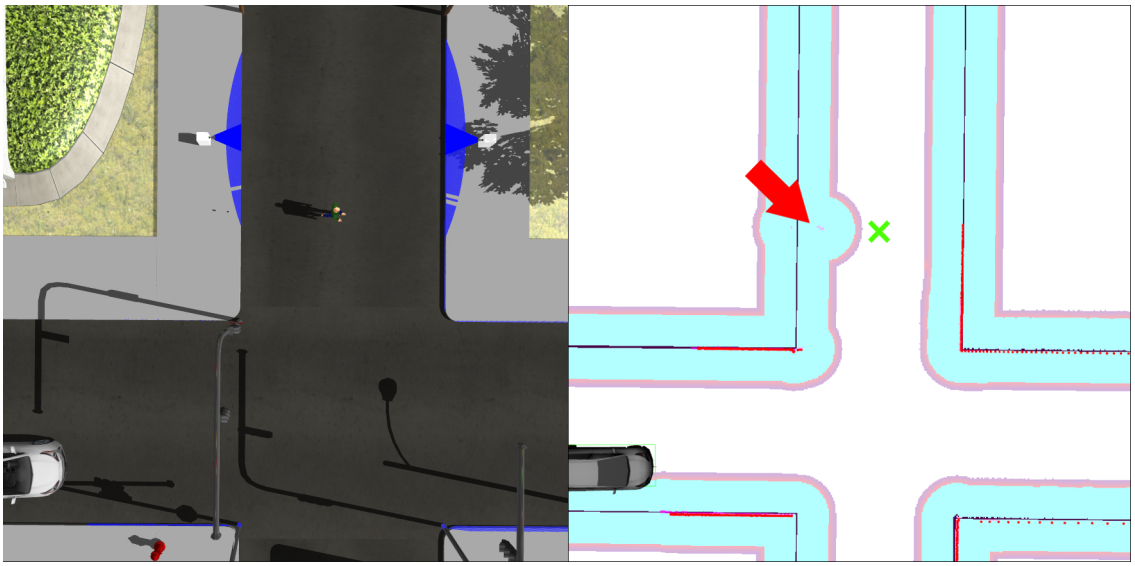


Figure 6.11: Prius Costmap and Gazebo Comparison.

As demonstrated by figure 6.11, the costmap representation of the pedestrian does not accurately match its position in the environment, the red arrow is pointing to the costmap cells occupied by the pedestrian, and the green "X" represents its actual position using Gazebo as a reference. This outcome was expected given the previous result, but it is important to

note that the simulation of the communication only introduced a small delay, negligible, to the costmap representation, conveying that the main factor that is affecting the accuracy of pedestrian's positioning in this scenario is the delay introduced by the Reliable stream used for the micro-ROS nodes.

6.1.3 Conclusions

After examining the outcomes of the aforementioned scenarios, it is evident that the desired outcome was only accomplished in the road block scenario. This success can be attributed to the static nature of the detected object and its considerable distance from the vehicle. Consequently, the time constraints associated with message delivery are less stringent, allowing for a generous time interval within which the vehicle can receive the message and execute the detour in a timely manner. Conversely, in the second scenario, time constraints were introduced, necessitating the rapid delivery of the message to retain the pedestrian's actual position at the moment of transmission. Unfortunately, the employed approach utilizing micro-ROS nodes and the Reliable message stream intended to allow fragmentation, introduced a substantial delay that hinders the accuracy of the pedestrian's positioning. Nevertheless, the utility of this approach for the vehicle may vary depending on the scenario's demands and how the received data is leveraged. For instance, it's conceivable to augment the inflation cost of the cells to expand the safety margin. In effect, this modification would cause the cells representing the pedestrian's area in the costmap to span the entire road. Such an adjustment should suffice to alert the vehicle to the existence of a dynamic obstacle within its path.

6.2 CAM Communications Delay Analysis

The analysis of communication delay holds significant importance, particularly in the context of obstacle data transmission. Obstacle data, due to its time constraints, necessitates prompt and accurate delivery to ensure timely and reliable decision-making processes. Understanding the delay incurred during the communication of obstacle data is crucial for evaluating the system's ability to meet these time constraints. By examining the delay, we gain valuable insights into the responsiveness of the communication framework and its impact on the overall system performance.

It is also important to mention that the performed tests only analyse the communication from the RSU to the vehicle, which means that the CAM communication delay tests do not take into account the possible delay that could be introduced by the micro-ROS nodes, given that this communication simulation is yet to be implemented.

6.2.1 OMNeT++ Comunication Analysis

As explained previously, the usage of OMNeT++/Artery is what enables the simulation of the communication between the RSU and the vehicle using ETSI ITS-G5, which simulates the real-world constraints regarding this technology. In order to verify if OMNeT++/Artery properly simulates the communication, its was analysed the delay between the sharing of the OccupancyGrid between the RSU and the vehicle ROS2 node that receives it. It was also analysed, for reference, the communication without the usage of OMNeT++.

For the obtaining of the delay values, it was calculated the time difference in the ROS2 node that receives the CAM message from OMNeT++. In ROS2, the CAM messages

are generated with an approximate rate of 135hz, and it is added a timestamp to one of its fields that contains the current time in seconds and nanoseconds in the message creation instant. This value serves as a measure of the time elapsed from the instant the message is generated within the RSU to the instant it reaches the CAM receiving node. This encompasses the complete sequence of actions involved in receiving the message within OMNeT++, forwarding a CAM message to the vehicle's representative module, and subsequently publishing the message to ROS2. Regarding the delay values of the direct communication between the RSU and the vehicle ROS2 node, it was used the same method but in this case, the ROS2 node subscribes directly to the CAM message generated by the RSU that is supposed to be sent to OMNeT++.

In figure 6.12, it is displayed a graph that compares the delay values with and without the usage of OMNeT++ for the message simulation. The graph was obtained using 50 samples from the obtained values calculated in the CAM receiver node.

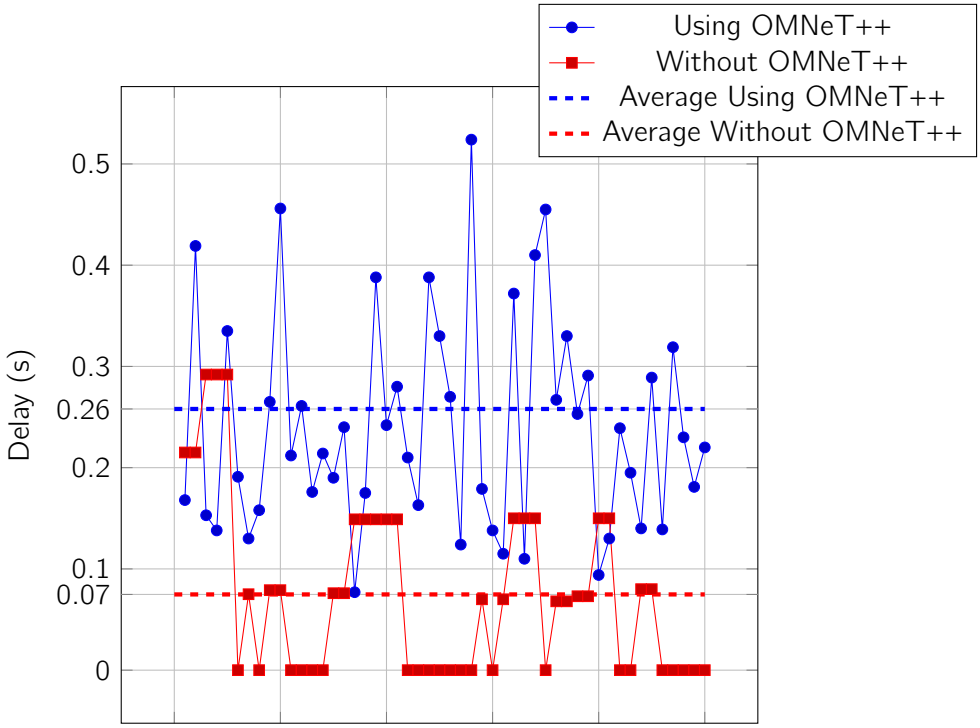


Figure 6.12: Message Sharing Delay Comparison.

The results obtained demonstrate that OMNeT++ effectively introduces a simulated communication delay that reflects real-world constraints. In the case of the direct communication, the measured values align with the expected results, with the majority falling below 0.1s. With the usage of OMNeT++, the recorded values range from 0.05s to 0.52s, which require further analysis for its validation. For instance, if the RSU were to transmit a message to the vehicle with a delay of 0.52s, it means that the received position information is only 0.52s old, which upon initial examination, it appears to be relatively insignificant. But, in the second scenario, this time interval could be significant enough to observe substantial changes in the pedestrian's position, which will be analysed later. However, the obtained values could be further increased if the micro-ROS communication was also being simulated, meaning that the pedestrian's position could be further outdated.

As mentioned earlier, the Gazebo world used for the second scenario, depicting a small city, poses higher computational demands compared to the first scenario. This increased computational load can be attributed to the larger number of models present in the world. Therefore, it was replicated in the initial testing world, the same scenario using also a Gazebo actor to represent a pedestrian. Figure 6.13 depicts a graph comparing the delays of both Gazebo worlds.

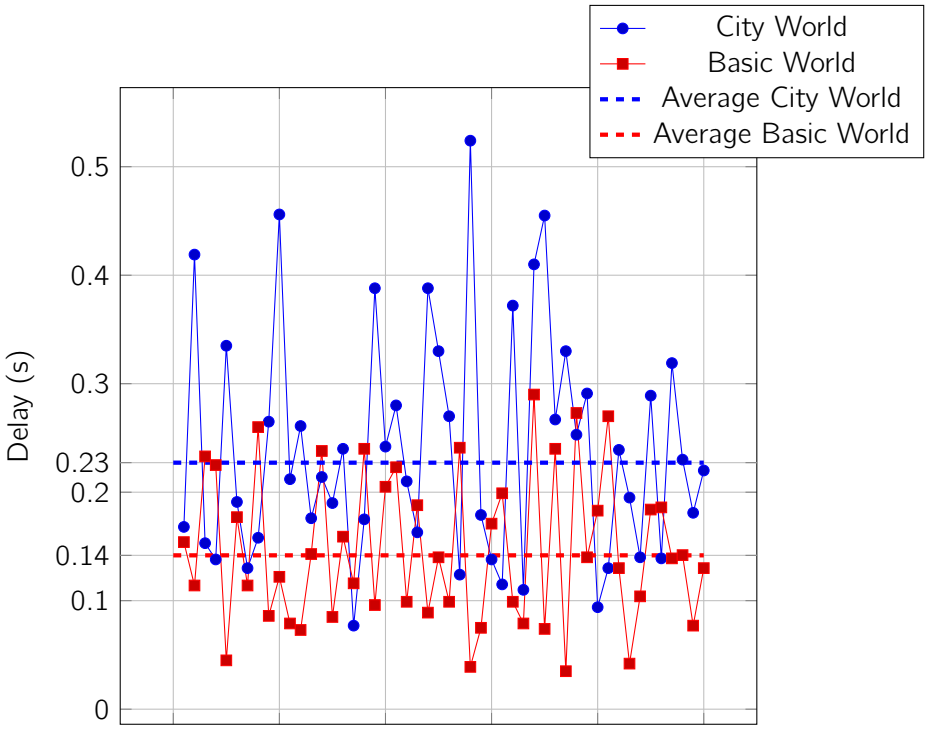


Figure 6.13: Gazebo World Message Delay Comparison.

Upon analyzing the graph, it can be observed that the more computationally demanding world utilized in the second scenario exhibits a slight increase in delay compared to the basic world used for testing. Significantly, the basic world demonstrated recorded values ranging from 0.03s as the lowest to 0.28s as the highest. In contrast, the city world exhibited a wider range, with recorded values of 0.05s as the lowest and 0.52s as the highest. Based on the obtained results, we will consider the values recorded in the basic world as our reference for the upcoming tests.

6.2.2 CAM Sending Frequency Analysis

As previously discussed, ensuring timely message delivery is crucial in the context of obstacle data transmission to meet the time constraints. While the message delivery time is the primary concern in the demonstrated static obstacle scenario, the second scenario presents an additional constraint. In the case of a dynamic obstacle, the focus also involves the message sending frequency as a critical factor.

When detecting a moving obstacle, it is crucial to have a sufficiently high frequency of data updates, the reason being that moving objects can exhibit rapid changes in direction, speed, or other attributes. A higher frequency allows the system to capture these changes more accurately and respond promptly to adapt to the obstacle's behavior. With a low frequency

of data updates, there will be a delay in receiving updates about the obstacle's current state. This delay can hinder vehicle's ability to react quickly and appropriately to changes in the obstacle's behavior.

To gain insights into determining the optimal frequency value for a scenario like the pedestrian crossing the road, an in-depth analysis of pedestrian behavior was conducted. In section 6.1.2, it was introduced the method used for the simulation of the pedestrian which is a Gazebo Actor. In this scenario, the pedestrian simply moves forward, at a constant velocity, from its starting point to its destiny point in the other side of the road, taking approximately 5 seconds. By subscribing to the Gazebo "/model_states" topic, we can obtain the current position of the pedestrian in the environment, which will allow us to analyse the pedestrian positioning throughout the timespan. In Gazebo, the metric system follows the International System of Units (SI) conventions, meaning that its Cartesian coordinate system unit of measurement is meters. By using meters as the unit of measurement, Gazebo enables better referencing to real-world measurements, enhancing the ability to relate simulated objects and their behaviors to their real-world counterparts.

Considering the utilization of ETSI ITS-G5 technology for communication, the frequency of CAM message generation must adhere to specific requirements outlined by ETSI. These requirements dictate the time interval between CAM generations, with a maximum limit of 1 second and a minimum limit of 0.1 seconds. Consequently, it is necessary to analyze the pedestrian's distance between each point after a time step of 0.1 seconds, until it reaches 1 second. This analysis allows us to determine the optimal frequency value that satisfies these requirements.

Frequency of 1 and 10 Messages per Second

To evaluate the impact of a message frequency of 1 message per second, we analyzed the positional difference of the pedestrian after this time interval. This was accomplished by utilizing Gazebo's clock advancement feature, which allows for the precise progression of time according to the desired time step. We set a timestep of 0.1 seconds and continuously analysed the pedestrian's position throughout the duration of 1 second. With the help of the "/model_states" topic, we can extract the x-axis values to calculate the distance between each point. This enables us to determine the average distance traveled within a time interval of 0.1 seconds, yielding an approximate value of 0.22 meters, which means that after 1 second, the difference of the pedestrian's position is 2.2 meters.

Figure 6.14 presents a time chart that provides a visual representation of the concept of having a frequency of 1 message per second. The red area in the figure highlights the positional changes of the pedestrian that are disregarded until the 1-second time mark is reached.

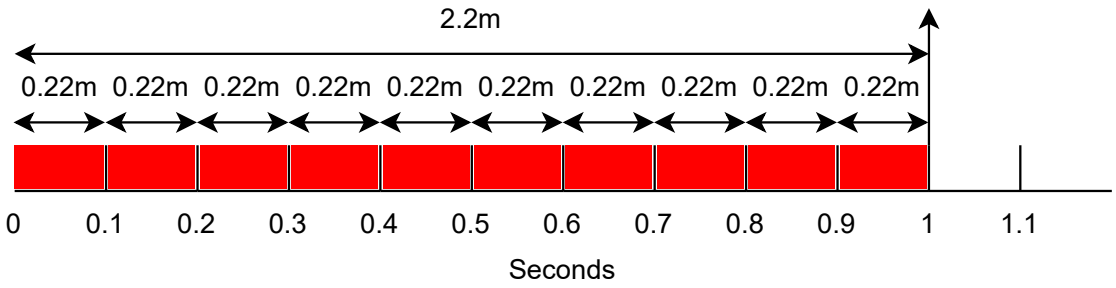


Figure 6.14: 1 Message per Second Time Chart.

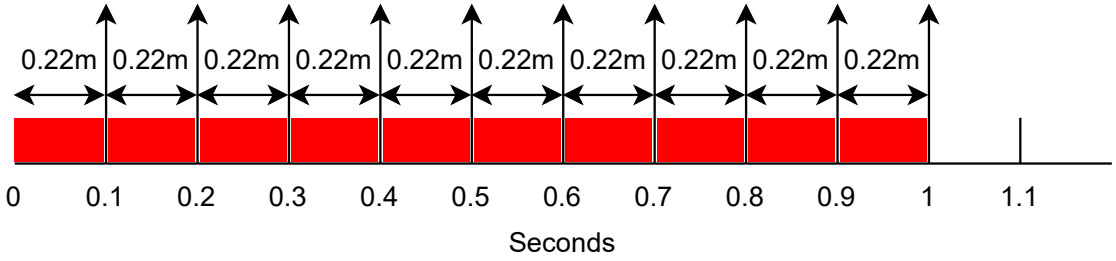


Figure 6.15: 10 Messages per Second Time Chart.

In Figure 6.16, it is illustrated the pedestrian's positions in the environment in a time interval of 1 second. The x-axis coordinates are 42.97 and 45.24 respectively, which results in a distance of 2.27 meters, and, as demonstrated in the figure, the positional discrepancy after a single second is notably substantial.

Figure 6.16: Pedestrian's Positioning After 1 Second.

Based on the obtained results, it can be concluded that a frequency value of 1 message per second is inadequate for the vehicle to effectively track and capture significant changes in the pedestrian's position.

With a frequency value of 10, the messages are sent every 0.1 seconds, which in this case, represents every 0.22 meters traveled by the pedestrian. This higher frequency enables a more precise and accurate tracking of the pedestrian's position, thereby facilitating a more responsive and timely response from the vehicle. Additionally, it enhances the ability to capture rapid changes in the pedestrian's movement, ensuring better situational awareness and improving overall safety measures.

After 0.1 seconds, as illustrated in Figure 6.17, the change in the pedestrian's position is significantly less pronounced compared to the previous example. This indicates that the frequency value of 10, corresponding to a message sent every 0.1 seconds, captures the pedestrian's movements with a satisfactory level of precision. Therefore, we can conclude that this frequency value represents the lower acceptable limit, striking a balance between capturing meaningful positional changes and minimizing the computational and communication overhead.

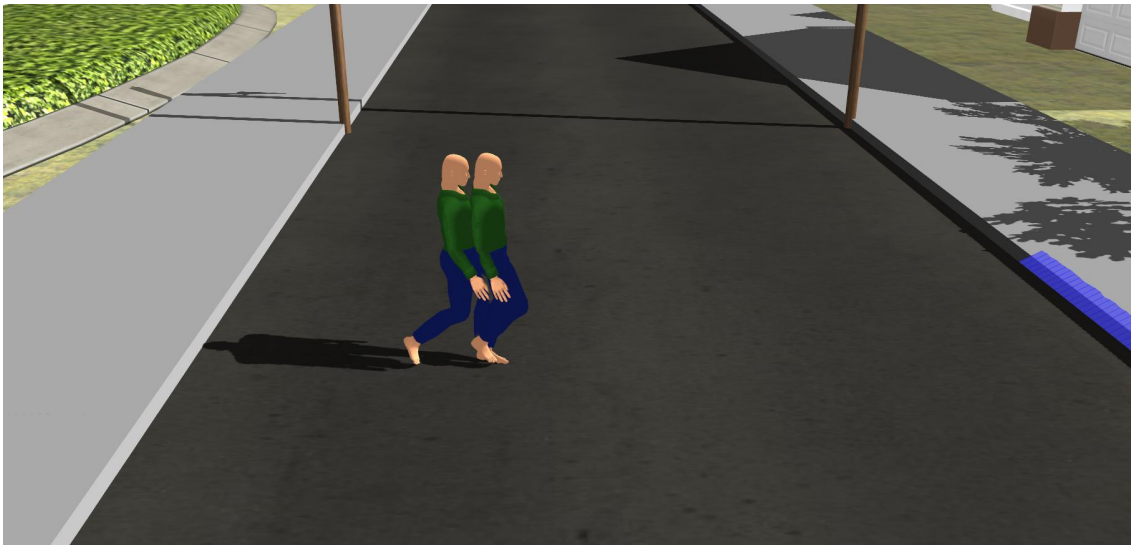


Figure 6.17: Pedestrian's Positioning After 0.1 Seconds.

Still, there are other factors that need to be considered when determining the suitability of this frequency value. As mentioned previously, the time it takes to send a message from the RSU to the vehicle can introduce a significant delay, which can be as high as 0.28 seconds. Moreover, there may be additional delays caused by the micro-ROS communication layer. Taking these factors into account, it is crucial to assess whether the chosen frequency value allows for timely and accurate information exchange between the RSU and the vehicle. If the total communication delay exceeds the desired level of complying with the time constraints, it may compromise the effectiveness of using a frequency value of 10.

In order to study the optimal frequency value, it is also necessary to analyse if it is worth using a high frequency value. As previously seen, a frequency value of 10 is suitable enough to accurately trace a moving obstacle, but if we also take into account the message sending latency, it is possible that each message is not delivered before a new message is sent.

We observe that a frequency value of 10 leads to a higher overhead due to the increased number of messages processed per second. However, even with a latency of 0.3 seconds, there remains a time interval of 0.1 seconds between the reception of each message, which is not a significant overload. Therefore, for scenarios like the one involving a pedestrian

crossing the road, a frequency of 10 is deemed more suitable, as it ensures the necessary accuracy in tracking and monitoring movements.

6.3 micro-ROS Communications Analysis

This section analyzes the delay described earlier in relation to micro-ROS Reliable stream messaging. This analysis aims to gain a better understanding of whether this delay is the sole obstacle preventing the validation of this approach.

In section 6.1.2, a notable disparity in delay was observed between the Reliable and Best-Effort QoS. While the simulation of micro-ROS communications via OMNeT++ was not yet implemented, we conducted an analysis of the delay introduced by utilizing the Reliable QoS. This analysis involved examining the timestamp difference between the transmission of the LaserScan message from the micro-ROS node to the RSU node and the corresponding moment of its reception by the RSU.

Figure 6.18 depicts a comparison of the message delays between the two QoS streams. The disparity between the delays is quite significant, with the usage of Reliable QoS introducing a delay that is one second higher than the delay associated with the Best-Effort QoS. The analysis conducted previously highlighted the significant variation in the pedestrian's position within a single second. However, when factoring in the delay introduced during the transmission of the message from the RSU to the vehicle, the total time difference can far exceed one second.

Given the results, we can conclude that due to the introduced delay by the Reliable QoS, the used micro-ROS communication approach notably decreases the accuracy in detecting dynamic obstacles.

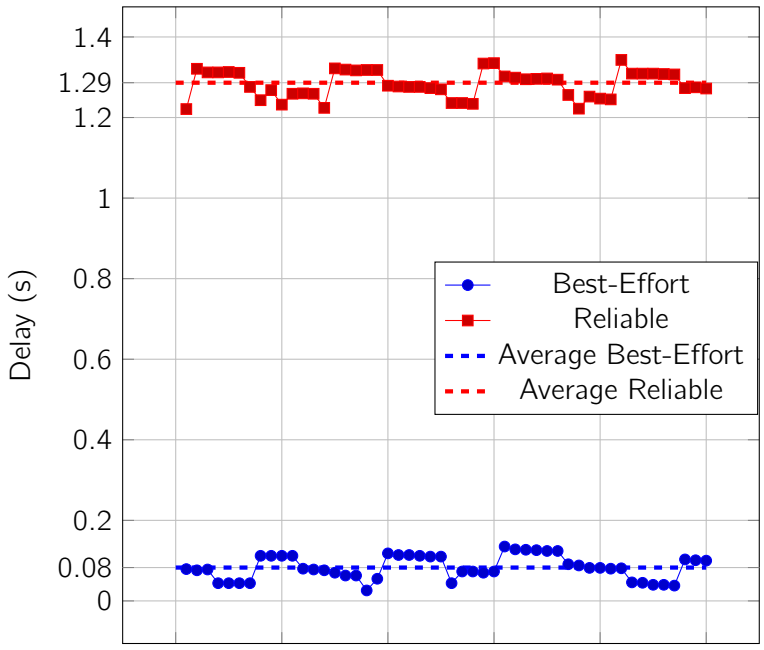


Figure 6.18: micro-ROS QoS Delay Comparison.

On the contrary, the Best-Effort QoS exhibits notably lower delay values, with the highest recorded value being just 0.13 seconds. This suggests that, without the aforementioned limitation in the Navigation system, this approach could serve as a valid solution for dynamic object detection. However, the communication between the sensing devices and the RSU might introduce a delay, regardless of the communication technology used for the micro-ROS connection to the agent.

6.4 Implemented Scenarios Evaluation

Based on the previously analyzed results, this section evaluates the implemented scenarios to determine whether their outcomes align with the expected results. This evaluation aims to demonstrate the suitability of the introduced approach for handling these type of scenarios. As the developed work revolves around the simulation of real scenarios, this evaluation will also play a significant role in assessing the overall effectiveness and validity of the work.

In terms of the evaluation methodology, this study will comprehensively assess the achievement of expected outcomes in both scenarios. This evaluation serves as a critical measure of the effectiveness of the proposed cooperative perception approach. Furthermore, an analysis of the simulation framework's role in the implemented scenarios will be undertaken. This evaluation aims to gauge the framework's utility and potential in facilitating the study of real-world scenarios, thereby enriching our understanding of its applicability and contribution.

6.4.1 Road Block Scenario

The static obstacle scenario, involving a stationary obstacle necessitating vehicle detouring, successfully realized its expected outcome, as demonstrated in Section 6.1.1.

With the simulation of the communications, we were able to analyse external factors that could affect the effectiveness of the cooperation, such as latencies and communication distance. Leveraging OMNeT++, the research determined that the communication range between the vehicle and RSU ensures timely vehicle response, even under worst-case latency scenarios.

Overall, the static obstacle scenario attests to the viability of the cooperative perception approach within scenarios possessing lenient time constraints.

6.4.2 Dynamic Obstacle Scenario

The dynamic obstacle scenario imposed more stringent time constraints compared to the static obstacle scenario. Within this context, involving a pedestrian crossing a road, simulation tools proved instrumental in dissecting the pedestrian's behavior and the ensuing impact on vehicle response time.

As demonstrated earlier, the analysis encompassed the pedestrian's temporal position variation, leading to the determination of an optimal frequency value and the potential influence of communication latency on data precision upon the vehicle's receipt. From the outcomes, it can be deduced that while there is a discernible degree of imprecision concerning the vehicle's knowledge of the pedestrian's position, the data's utility in evading collisions remains feasible. For instance, the utilization of elevated cell cost values in the costmap can augment the safety distance to objects, mitigating this imprecision.

In summation, this scenario underscores the simulation framework's adeptness in validating cooperative perception strategies, especially within the temporal limitations introduced by the dynamic obstacle detection.

6.5 Conclusions

The outcomes of the conducted tests underscore the considerable potential of the simulation framework for scrutinizing and dissecting cooperative scenarios. Despite the earlier delineated constraints such as the unavailability of simulating the micro-ROS underlying communication protocol and the influence of computational capacity, evidenced by the comparison of the utilized Gazebo worlds, the study effectively realized a proof-of-concept cooperative perception approach. The tools employed facilitated an in-depth exploration of external variables critical for validating the proposed approach. Consequently, the obtained results robustly affirm and validate the practicality and efficacy of the co-simulation framework.

Chapter 7

Conclusions

This chapter serves as the concluding section of this thesis, where it is discussed the limitations encountered during the research and development process. Additionally, it explores potential future work to propose further enhancements for the developed framework. Addressing these limitations and considering potential improvements is essential for advancing the application of the framework in various scenarios. Finally, we provide a comprehensive conclusion, summarizing the key findings and contributions of this work.

7.1 Limitations

The limitations encountered during the development of the simulation framework will allow us to understand what needs to be improved to further optimize the framework for future work. The main limitations are the following:

- **Costmap 2D Map Clearing** - The implemented approach hinges on the capability of Costmap 2D to incorporate obstacles to the obstacle layer using the LaserScan data obtained from sensors. However, to ensure compatibility with the Best-Effort QoS provided by micro-ROS, it is essential to maintain a MTU size not exceeding 512 Bytes. Consequently, the LaserScan data must be condensed to fit within the message, which necessitates a reduction in the number of laser sensor samples. During testing, it was discovered that when utilizing 60 laser samples in Gazebo, the Costmap 2D obstacle layer failed to remove obstacles from the map when they were no longer detected. As a result, we had to resort to employing a Reliable QoS, which unfortunately introduced significant communication delays, rendering it unsuitable for accurately detecting and perceiving moving obstacles. This limitation poses a challenge in achieving real-time obstacle tracking and underscores the need for further investigations to improve the performance and accuracy of the approach.
- **micro-ROS underlying communications simulation** - Regarding the communications simulation, we utilized the OMNeT++/Artery distribution introduced by AuNa, which already had ROS2 integration. This approach enabled us to simulate ITS-G5 communications between the RSU and the vehicle with minimal modifications to suit our desired approach. However, when attempting to simulate the communication between the micro-ROS nodes and the RSU using 6LoWPAN, we encountered challenges. Although we explored an OMNeT++ 6LoWPAN model available on the official website, it was outdated and incompatible with the version of OMNeT++ we were using. Additionally, experimenting with an older version introduced other issues due to dependencies compatibility. To achieve the simulation of 6LoWPAN communication, it is required further investigation into the connection between the micro-ROS client

and the Agent, as well as its integration with OMNeT++. This aspect remains an area of interest for future research and improvements.

7.2 Future Work and Improvements

The aforementioned limitations provide valuable insights and open new avenues for future research and improvements. Enhancing the framework requires addressing the following aspects:

- **Optimization of Costmap 2D** - To utilize the Best-Effort QoS effectively, exploring ways to optimize the Costmap 2D obstacle layer is essential. Investigating the feasibility of dynamically adjusting the number of laser samples or devising a mechanism to remove obstacles when they are no longer detected, enabling smoother communication with micro-ROS.
- **Exploring other sensors** - Incorporating various sensors in addition to the LiDAR could enhance the perceptual capabilities of the sensing devices. For instance, the inclusion of cameras could enable object identification, however, this would necessitate further research and development due to the limited computational capacities of the microcontrollers.
- **Vehicle autonomous driving and navigation improvements** - The autonomous driving and navigation system of the vehicle, powered by Nav2, would benefit from enhancements in its configuration to achieve optimized driving performance. Furthermore, the localization method, occasionally exhibiting substantial deviations from the vehicle's true location, warrants exploration of alternative techniques beyond AMCL. Another solution would be the usage of another stack, e.g. Autoware, given that it is an open-source software stack designed for self-driving vehicles built on ROS, it presents an intriguing opportunity to explore its integration with the simulation framework, especially considering its primary focus on autonomous driving. This synergy could provide valuable insights and advancements in the development of autonomous driving capabilities within the context of the simulation environment.
- **Simulation of the micro-ROS wireless connection** - For a comprehensive evaluation of the framework, implementing micro-ROS communication simulation in OMNeT++ is crucial. Further investigation into integrating micro-ROS client-to-agent communication with OMNeT++ and resolving compatibility issues will be beneficial. It will also be necessary to study a suitable communication technology, e.g. 6LoWPAN.
- **Adding new scenarios** - Extending the framework to handle complex and dynamic environments with varying traffic patterns, diverse sensor types, and multiple vehicles will provide a more realistic assessment of its capabilities.
- **Performance Optimization** - Exploring opportunities to enhance the overall performance of the co-simulation framework, such as reducing latency and minimizing overhead, to achieve near-real-time simulation results. For example, investigating alternative approaches for sharing obstacle data without sending the entire costmap in a message could significantly improve performance. This could be achieved using a custom message type that only contains the occupied cells that were filled by the obstacle layer.

By addressing these aspects, the framework can be refined and extended, making it a more robust and versatile tool for simulating dynamic environments with embedded devices and enhancing its applicability in real-world scenarios.

7.3 Conclusion and Final Considerations

This thesis has explored and developed a co-simulation framework, by extending previous developed work and technologies, enabling the simulation of dynamic environments using embedded devices for cooperative perception. By leveraging micro-ROS, the framework efficiently integrates recent ROS toolsets into embedded systems, facilitating seamless communication and interaction between the various subsystems.

Throughout this work, we have successfully demonstrated the feasibility of the framework by implementing two scenarios: one involving a static obstacle and another with a pedestrian crossing the road. The results of these simulations partially align with our expectations, given that they serve as a proof-of-concept for cooperative perception utilizing the previously mentioned technologies, which demonstrate the potential of our approach in these type of scenarios. The developed approach effectively tackled several challenges in cooperative perception as mentioned in 2.1.2. It notably leveraged the OccupancyGrid message type to consolidate obstacle data from multiple sensors into a unified message. This approach promotes multi-modality, ensuring that regardless of the sensor employed, data is consistently conveyed through a single message type. The utilization of micro-ROS further streamlines this capability by enabling seamless communication between the sensing devices and the RSU. The chosen centralized approach for vehicle communication effectively tackles calibration and multiple point-of-view challenges. This approach relies on stationary sensors, eliminating the need for frequent transformation matrix adjustments, and consolidates various viewpoints into a single message for the vehicle. Additionally, the incorporation of a navigation system with localization capabilities in the RSU enables precise placement of detected obstacles based on the map's coordinates. Since this map is shared with the vehicle, it ensures perception matching, ensuring that both costmap obstacles align accurately in terms of positioning.

By addressing the previously introduced aspects and conducting further research on performance optimization, the co-simulation framework can achieve higher accuracy, responsiveness, and efficiency, making it a valuable tool for simulating dynamic environments and studying other approaches for cooperative perception.

Bibliography

Altinel, Berk, Frank Wollenschläger, and Matthias A. Hein (2019). "Interference Tests of ITS-G5 Vehicle-to-Vehicle Communication Networks with Virtual Drive Tests". In: *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 1–5. doi: 10.1109/ICCVE45908.2019.8965140.

Amanatidis, Petros et al. (2023). "Cooperative Task Execution for Object Detection in Edge Computing: An Internet of Things Application". In: *Applied Sciences* 13.8. issn: 2076-3417. doi: 10.3390/app13084982. url: https://www.mdpi.com/2076-3417/13/8/4982.

Caillot, Antoine et al. (2022). "Survey on Cooperative Perception in an Automotive Context". In: *IEEE Transactions on Intelligent Transportation Systems*. doi: 10.1109/TITS.2022.3153815. url: https://hal.science/hal-03608119.

CARLA Team (2023). *CARLA*. url: https://carla.org/.

Dosovitskiy, Alexey et al. (2017). "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

eProsima (2018). *eProsima Micro XRCE-DDS*. url: https://micro-xrce-dds.docs.eprosima.com/.

European Telecommunications Standards Institute (2010). *ETSI EN 302 665 V1.1.1 - Intelligent Transport Systems (ITS); Communications Architecture*.

– (2014). *ETSI EN 302 637-2 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*.

Flores Muñoz, Juan and Iñigo Muguruza Goenaga (June 2019). *micro-ROS default simulation environment release Initial Y1*. url: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5c54fa2ce&appId=PPGMS.

Flores Muñoz, Juan and Borja Outerelo Gamarra (Dec. 2019). *micro-ROS default simulation environment release Revised*. url: http://www.ofera.eu/storage/deliverables/M24/OFERA_13_D26_Micro-ROS_default_simulation_environments_release_revised__PU.pdf.

FS Studio (Jan. 2021). *Ros Gazebo: Everything you need to know*. url: https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/.

Gazebo (Oct. 2017). *Blog : Vehicle and city simulation*. url: https://classic.gazebosim.org/blog/car_sim.

Gazis, A (2021). "What is IoT? The Internet of Things explained". In: *Academia Letters*, p. 2.

Goenaga, Iñigo, Juan Muñoz, and Victor Vilches (June 2019). *Sensor micro-ROS use-case in Modular Arm Release - Initial*. url: http://www.ofera.eu/storage/deliverables/M18/OFERA_60_D63_Sensor_micro-ROS_use-case_initial_PU.pdf.

Ikrissi, Ghizlane and Tomader Mazri (2021). "IOT-based Smart Environments: State of the art, security threats and solutions". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 46, pp. 279–286.

Kołcon, Tomasz, Mateusz Maciaś, and Alexandre Malki (June 2019). *Smart warehouse micro-ROS use-case - initial*. url: http://www.ofera.eu/storage/deliverables/M18/OFERA_64_D67_Smart_warehouse_micro-ROS_use-case_release_Initial.pdf.

Kosuru, Venkata Satya Rahul and Ashwin Kavasseri Venkitaraman (2023). "Advancements and challenges in achieving fully autonomous self-driving vehicles". In: *World Journal of Advanced Research and Reviews* 18.1, pp. 161–167.

Li, Yiming et al. (2022). "V2X-Sim: Multi-Agent Collaborative Perception Dataset and Benchmark for Autonomous Driving". In: *IEEE Robotics and Automation Letters* 7.4, pp. 10914–10921. doi: 10.1109/LRA.2022.3192802.

Macenski, Steve and Ivona Jambrecic (2021). "SLAM Toolbox: SLAM for the dynamic world". In: *Journal of Open Source Software* 6.61, p. 2783. doi: 10.21105/joss.02783. url: https://doi.org/10.21105/joss.02783.

Macenski, Steve, Francisco Martin, et al. (Oct. 2020). "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. doi: 10.1109/iros45743.2020.9341207. url: https://doi.org/10.1109%5C%2Firos45743.2020.9341207.

Macenski, Steve, Shrijit Singh, et al. (2023). *Regulated Pure Pursuit for Robot Path Tracking*. arXiv: 2305.20026 [cs.RO].

Macenski, Steven et al. (2022). "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66, eabm6074. doi: 10.1126/scirobotics.abm6074. url: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

micro-ROS (2023a). url: https://micro.ros.org/.

– (2023b). *Comparison to related approaches*. url: https://micro.ros.org/docs/overview/comparison/.

– (2023c). *Handling messages memory in micro-ROS*. url: https://micro.ros.org/docs/tutorials/advanced/handling_type_memory/.

– (2023d). *micro-ROS Features and Architecture*. url: https://micro.ros.org/docs/overview/features/.

– (2023e). *micro-ROS Memory Profiling*. url: https://micro.ros.org/docs/concepts/benchmarking/memo_prof/.

– (2023f). *micro-ROS Zephyr Emulator*. url: https://micro.ros.org/docs/tutorials/core/zephyr_emulator/.

– (2023g). *ROS 2 Feature Comparison*. url: https://micro.ros.org/docs/overview/ROS_2_feature_comparison/.

– (2023h). *Supported Hardware*. url: https://micro.ros.org/docs/overview/hardware/.

Nav2 (2020a). *Nav2*. url: https://navigation.ros.org/.

– (2020b). *Setting Up the Robot's Footprint*. url: https://navigation.ros.org/setup_guides/footprint/setup_footprint.html.

nsnam (2023). *About ns-3*. url: https://www.nsnam.org/about/.

Object Management Group et al. (2020). *DDS For Extremely Resource Constrained Environments - DDS-XRCE*.

OFERA (2020). *OFERA project: Open Framework for Embedded Robot Applications*. url: http://www.ofera.eu/.

Olsson, Jonas (2014). "6LoWPAN demystified". In: *Texas Instruments* 13, pp. 1–13.

Open Robotics (2022). *Understanding nodes*. url: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html.

– (2023). *Concepts*. url: https://docs.ros.org/en/galactic/Concepts.html.

Open Source Robotics Foundation (2014). *Make an animated model (actor)*. url: https://classic.gazebosim.org/tutorials?tut=actor&cat=build_robot.

OpenSim Ltd. (2019). *What is OMNeT++?* url: https://omnetpp.org/intro/.

Robotics, Open (2021). *The ros ecosystem*. url: https://www.ros.org/blog/ecosystem/.

ROS (Feb. 2022). *LaserScan Message*. url: http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html.

Sanchez, Pablo et al. (Dec. 2021). *Dronecode micro-ROS use-case release - Extension*. url: http://www.ofera.eu/storage/deliverables/M48/OFERA_95_D610_Drone_micro-ROS_use-case_release_Extension.pdf.

Teper, Harun et al. (2022). "AuNa: Modularly Integrated Simulation Framework for Cooperative Autonomous Navigation". In: *arXiv preprint arXiv:2207.05544*.

Thandavarayan, Gokulnath, Miguel Sepulcre, and Javier Gozalvez (2020). "Cooperative Perception for Connected and Automated Vehicles: Evaluation and Impact of Congestion Control". In: *IEEE Access* 8, pp. 197665–197683. doi: 10.1109/ACCESS.2020.3035119.

The QEMU Project Developers (2022). *About QEMU*. url: https://www.qemu.org/docs/master/about/index.html.

Tsukada, Manabu et al. (2020). "AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles". In: *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pp. 1–6. doi: 10.1109/VTC2020-Fall49728.2020.9348525.

Vieira, Bruno et al. (2019). "COPADRIVe - A Realistic Simulation Framework for Cooperative Autonomous Driving Applications". In: *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 1–6. doi: 10.1109/ICCVE45908.2019.8965161.

Wolter, Diedrich and Alexandra Kirsch (Aug. 2017). "Smart Environments: What is it and Why Should We Care?" In: *KI - Künstliche Intelligenz* 31.3, pp. 231–237. issn: 1610-1987. doi: 10.1007/s13218-017-0498-4. url: https://doi.org/10.1007/s13218-017-0498-4.

Zephyr Project (Sept. 2023). *Peripheral and Hardware Emulators*. url: https://docs.zephyrproject.org/latest/hardware/emulator/index.html.