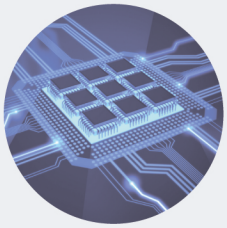
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS



Explorar comunicação V2X para reforçar a segurança em colisões de veículos autónomos

RODRIGO OLIVEIRA SANTOS MOREIRA
outubro de 2025

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Leveraging V2X Communication for Enhanced Crash Safety in Autonomous Vehicles

Rodrigo Oliveira Santos Moreira

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

Supervisor: Ricardo Severino
External Supervisor: Vasco Dias, Pedro Martins

Evaluation Committee:

President:

Luis Miguel Pinho, Instituto Superior de Engenharia do Porto

Members:

Ênio Filho, Critical TechWorks

Ricardo Severino, Instituto Superior de Engenharia do Porto

Porto, October 6, 2025

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

I declare that the work presented in this document is original and my own, and has not previously been used for any other purpose.

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

ISEP, Porto, October 6, 2025

Abstract

Ensuring safety in autonomous vehicles in complex traffic scenarios is arguably still one of the most important intelligent transport system challenges. Conventional perception systems that depend on sensors such as cameras, LiDAR, and radar are prone to line-of-sight-relevant constraints, adverse weather conditions, and occlusions that can impede threat detection in scenarios of blind turns or obstructed intersections. Vehicle-to-Everything (V2X) communication is also hailed as the hopeful add-on to enhance situational awareness outside vehicle sensor range, where cars may exchange position, velocity, brake, and intent data in real-time.

This thesis investigates the application of V2X communication to enhance crash safety by developing a simulation infrastructure that integrates Unreal Engine 5 for photo-realistic scenario simulation and the eCAL middleware for lightweight, low-latency message passing. The infrastructure was developed to simulate cooperative perception for low-visibility scenarios, aiming to establish whether early communication introduces longer reaction times and allows for earlier pre-crash safety system activation, i.e., airbags.

Even though the integration of a network simulator (OMNeT++) is incomplete as the compilers and toolchains are not compatible, the project has a simulated working environment using Unreal Engine 5 and ensures eCAL's role in passing structured data using Protobuf. Experimental results indicate seamless communication between virtual vehicles with near-zero latency, which depicts the potential as well as the limitations of shared-memory communication without real-world network simulation.

This parer results provide a clearer vision of the role played by V2X in complementing legacy perception systems on autonomous vehicles. They also herald the need for toolchain synchronizations and simulator compatibilities in follow-on efforts. While the framework is incomplete, it is structured to naturally generalize to more complicated scenarios, heterogeneous sensor fusion, and full real-time synchronization with network simulators. Lastly, this research confirms the standing of V2X as one of the foremost enablers of cooperative safety applications in the quest for safer autonomous driving.

Keywords: V2X, Cooperative Perception, Crash Safety, Unreal Engine 5, eCAL,OMNeT++, Autonomous Vehicles

Resumo

Garantir a segurança em veículos autónomos em cenários de tráfego complexos ainda é, sem dúvida, um dos desafios mais importantes dos sistemas de transporte inteligentes. Os sistemas de perceção convencionais que dependem de sensores, como câmaras, LiDAR e radar, são propensos a restrições relacionadas com a linha de visão, condições meteorológicas adversas e oclusões que podem impedir a deteção de ameaças em cenários de curvas cegas ou cruzamentos obstruídos. A comunicação Vehicle-to-Everything (V2X) também é aclamada como um complemento promissor para melhorar a consciência situacional fora do alcance dos sensores do veículo, onde os carros podem trocar dados de posição, velocidade, travagem e intenção em tempo real.

Esta tese investiga a aplicação da comunicação V2X para melhorar a segurança contra colisões, desenvolvendo uma infraestrutura de simulação que integra o Unreal Engine 5 para simulação de cenários fotorrealistas e o middleware eCAL para passagem de mensagens leve e de baixa latência. A infraestrutura foi desenvolvida para simular a perceção cooperativa para cenários de baixa visibilidade, com o objetivo de estabelecer se a comunicação precoce introduz tempos de reação mais longos e permite a ativação mais precoce do sistema de segurança pré-colisão, ou seja, os airbags.

Embora a integração de um simulador de rede (OMNeT++) esteja incompleta, pois os compiladores e as cadeias de ferramentas não são compatíveis, o projeto tem um ambiente de trabalho simulado usando o Unreal Engine 5 e garante o funcionamento do eCAL na transmissão de dados estruturados usando o Protobuf. Os resultados experimentais indicam uma comunicação perfeita entre veículos virtuais com latência quase nula, o que mostra o potencial e as limitações da comunicação de memória partilhada sem simulação de rede no mundo real.

Os resultados deste artigo fornecem uma visão mais clara do papel desempenhado pelo V2X no complemento dos sistemas de perceção legados em veículos autónomos. Eles também anunciam a necessidade de sincronizações de cadeias de ferramentas e compatibilidades de simuladores em esforços subsequentes. Embora a estrutura esteja incompleta, ela está estruturada para generalizar naturalmente para cenários mais complicados, fusão de sensores heterogéneos e sincronização em tempo real total com simuladores de rede. Por fim, esta pesquisa confirma a posição do V2X como um dos principais facilitadores de aplicações de segurança cooperativa na busca por uma condução autónoma mais segura.

Acknowledgements

This thesis would not have been possible without the support and guidance of several people to whom I am deeply grateful.

I would like to thank Eng. Vasco Dias and Eng. Pedro Branquinho for their orientation, as well as the entire Airbag team at Continental Engineering Services for welcoming me and integrating me into their work environment.

My gratitude also goes to Dr. Ricardo Severino for his guidance and valuable input throughout this stage of my academic journey.

Finally, I want to express my appreciation to my family and my girlfriend for their continuous support and encouragement.

Contents

List of Figures

List of Source Code

List of Abbreviations

CAV	Connected and Autonomous Vehicles
V2X	Vehicle-to-Everything
V2V	Vehicle-to-Vehicle
V2I	Vehicle-to-Infrastructure
V2P	Vehicle-to-Pedestrian
V2N	Vehicle-to-Network
UE5	Unreal Engine 5
eCAL	enhanced Communication Abstraction Layer
CES	Continental Engineering Services
ADAS	Advanced Driver Assistance Systems
RSU	Roadside Units
VLOS	Visual Line-Of-Sight
DSRC	Dedicated Short Range Communication
C-V2X	Cellular-V2X
CAM	Cooperative Awareness Message
DENM	Decentralized Environmental Notification Message
BSM	Basic Safety Message
BLOS	Beyond Line-Of-Sight
CP	Cooperative Perception
NLOS	Non-Line-Of-Sight
ROS2	Robotic Operating System 2
QoS	Quality of Service
NR-V2X	New-Radio Vehicle-to-Everything
URLLC	Ultra-reliable Low Latency Communication

Chapter 1

Introduction

In order to contextualise this thesis with the current status of autonomous vehicles and simulators for this purpose, this chapter provides a first glance at the project. The chapter begins by outlining the context of the thesis and how it aligns with the goals of the hosting company, problem definition, the objectives, the anticipated outcomes, finishing with the structure of this report.

1.1 Thesis Context

The growing operation of Connected Autonomous Vehicles (CAVs) in real-life settings comes with tremendous promises in mobility, efficiency, and traffic flow. Yet, one of the most vital elements still in the making is the guarantee of safety of these systems, especially in dynamic and intricate city environments where conventional perception systems are lacking. Settings such as blind bends, occluded intersections, or highly crowded areas present exclusive challenges that surpass the potentials of traditional onboard sensors, which are intrinsically constrained by line-of-sight limits, climatic factors, and physical obstructions. These constraints can drastically compromise the capacity of autonomous vehicles to identify and respond to impending hazards in a timely and dependable fashion.

Addressing these limitations, Vehicle-to-Everything (V2X) communication has appeared as an enabling technology to extend the situational awareness of autonomous systems. V2X supports real-time information exchange between vehicles (V2V), infrastructure (V2I), pedestrians (V2P), and cloud services (V2N), forming a cooperative and connected mobility ecosystem. By exchanging critical information such as position, speed, braking, and planned manoeuvres, vehicles can anticipate hazards that are out of their VLOS and take preventive action to try and avoid collisions or lessen their severity in case these are unavoidable. When vision is impaired, this is especially useful as roadside assistance or early warnings from other surrounding vehicles can greatly enhance decision-making and reaction times.

In parallel with the development of V2X technologies, there has been increasing interest in the creation of simulation tools that facilitate realistic, scalable, and controlled evaluation of

autonomous vehicle systems and cooperative communication approaches. High-fidelity simulation environments provide researchers with the ability to model and test scenarios that might consist of limited visibility, unpredictable obstacles, and multi-agent interaction without the risks and expenses of real-world testing. Such platforms provide a safe and reproducible framework to assess system performance, test decision-making algorithms, and investigate communication dynamics under diverse conditions.

In order to deploy such simulation-based investigations successfully, the interplay between high-end rendering engines and real-time middleware is crucial. With its sophisticated physics and visuals, Unreal Engine 5 (UE5) provides a strong basis for creating dynamic traffic actors, realistic road networks, and a variety of environmental scenarios. Its capacity for simulating sensor behavior, motion dynamics, and environmental interactions renders it well-suited to autonomous vehicle testing. Augmenting this, communications middleware like the Enhanced Communication Abstraction Layer (eCAL) facilitates low-latency, high-throughput messaging between parts of the simulation system. eCAL offers distributed architecture support and publish-subscribe communication patterns, which positions it well for use in simulating V2X interactions between multiple agents in the simulated world.

By integrating Unreal Engine 5 and eCAL, it is possible to create efficient simulation platforms for autonomous vehicle research. Such platforms support real-time information exchange between virtual vehicles, making it feasible to investigate cooperative behavior and communication-based safety features. V2X simulation can demonstrate how shared situational awareness between vehicles improves response times and lowers both the collision risks and the severity of crashes in scenarios like blind corners or low visibility intersections. The objective of this thesis is to develop an eCAL-based simulation framework on Unreal Engine 5 to evaluate how V2X communication would improve autonomous vehicle crash safety. The system will evaluate how proactive communication could improve safety outcomes and mitigate sensor limitations by taking into account important use cases for low-visibility scenarios and simulating real-time V2X communication. The results of this simulation research will help to contribute to the further development of our understanding of how V2X might make the greater autonomy mobility paradigm possible by offering insight into the development of more capable and cooperative vehicle systems.

1.2 Institutional Context

This thesis is being developed in collaboration with Continental Engineering Services (CES), a division of Continental AG that specializes in customized engineering solutions for automotive and mobility applications. CES operates across a wide range of domains, including powertrain, interior electronics, advanced driver assistance systems (ADAS), and vehicle dynamics. In the context of CES, the Chassis & Brakes segment is one of the driving segments for safety-critical system development that includes braking control modules, electronic stability modules, and airbag deployment modules.

The thesis is actually placed in the Airbags team, responsible for optimizing and designing passive safety technologies that will reduce the effects of collisions and help occupants in dire circumstances. While conventional airbag systems are based largely on sensor-dependent stimulus events, this paper discusses the potential of Vehicle-to-Everything (V2X) communication to add environmental awareness to current sensor systems in conditions of low visibility or high danger on the road. With threat detection based on V2X, it is possibly feasible to predict impending crashes before timely sensor stimulation, which paves the way for preceding or adaptive intervention of safety systems. By utilizing a simulation environment based on Unreal Engine 5 and run with the eCAL middleware, this thesis explores the prospect of the use of inter-vehicle communication to inform and optimize airbag firing logic and decision-making. This is one of the many activities of CES towards its goal of developing state-of-the-art and smart mobility solutions that are as much focused on performance as they are on occupant safety.

1.3 Problem Definition

Autonomous cars are ever more integrating modern transportation systems, with new promises for safer, more efficient, and smarter mobility. Still, one of the greatest challenges in the field is to guarantee the safety of these systems, especially in complicated or unforeseen driving conditions. One of the biggest problems with autonomous vehicle perception and decision-making systems is that they rely on onboard sensors, like camera, LiDAR, and radar, which are limited by line-of-sight, weather, sensor range, and environmental occlusions. In situations like blind curves or occluded intersections, these limitations can postpone the detection of hazards and enhance the chance or intensity of collisions.

The inclusion of Vehicle-to-Everything (V2X) communication technologies presents a viable way to expand environmental perception beyond physical constraints of on-board sensors. V2X enables cars to exchange important information, i.e., position, speed, braking, and intentions to maneuver, with nearby entities to realize more extensive and proactive awareness of the traffic situation. Yet, the creation and verification of such communications-based safety measures have their own set of technical issues. These are:

- Real-time communication limitations: V2X technologies like LTE-V2X and 5G-V2X have to provide very low latency and high reliability for safety-critical use cases. Nevertheless, real-world communication performance may be impacted by congestion, interference, handovers, or infrastructure constraints [1, 2].
- Perception and decision fusion: The combination of information obtained via V2X with sensor-based perception and autonomous decision-making logic is still challenging. Consistency, synchronization, and trust in external information need to be guaranteed in order to prevent unintended behavior or safety problems [3].

- Non-line-of-sight hazard detection: Although V2X provides non-line-of-sight awareness, it remains an open challenge to establish efficient logic and algorithms that are capable of fully utilizing this added layer of information, particularly in high-risk, rapidly developing situations such as blind corners or obscured intersections [4].
- Evaluation difficulties: Testing these scenarios in the field is time-consuming, expensive, and perhaps risky. Replicating unusual or extreme traffic situations safely and repeatedly is virtually impossible in the real world [5].

These difficulties pose major obstacles to creating, testing, and verifying safety features that depend on cooperative communication. Simulation thus becomes an essential tool. A high-fidelity virtual world can offer the basis to test V2X-based crash avoidance measures under controlled but realistic circumstances, allowing repeatable and low-risk experimentation.

These issues are tackled by this thesis through the suggestion of creating a simulation environment developed in Unreal Engine 5, utilizing its realistic physics and rendering to simulate intricate driving conditions. With the utilization of the eCAL middleware, it can simulate real-time V2X communication among vehicles, enabling testing of cooperative risk perception in situations with poor visibility. The method is intended to:

- Minimize the cost and risk associated with real-world testing;
- Allow for in-depth examination of the impact of V2X communication on crash detection and response time;
- Offer a basis for comparing various communication approaches and measuring their success in improving occupant safety.

Finally, this research aims to show that V2X communication, when effectively incorporated into the autonomous vehicle decision pipeline, has the potential to greatly minimize the likelihood and severity of collisions in scenarios where traditional perception systems are inadequate.

1.3.1 Objectives

This dissertation addresses the challenges mentioned. To reach this main goal it is important to define some smaller goals such as:

- Create an UE5 simulation environment that accurately replicates real-world scenarios.
- Integrate eCAL into the simulation environment for real-time control and monitoring of the vehicles and their communications in the simulation.
- Simulate communication through a communication simulator to introduce realism to the scenario.
- Verify how effective V2X communication can be in activating safety functions and analyze its performance.

1.3.2 Expected results

Addressing the objectives it is possible to expect results such as:

- Realistic simulation with control of the environment;
- Successful data exchange between the vehicles;
- Data-driven analysis of the viability of safety functions trigger via V2X in a real-world scenario.

1.4 Thesis Structure

This document is composed of eight parts. Chapter 1 presents the thesis' theme by describing and contextualizing the problem and enumerating the objectives this work aims to achieve. In chapter 2, there is a brief review of the state-of-the-art and recent works that have motivated this thesis. In chapter 3, its presented the simulation tools and other technologies that supported the thesis development, as well as some others that were considered but were not chosen in the end, making a comparison between them. Chapter 4 describes the overall system architecture and the concept, complemented by a use-case scenario. Chapter 5 describes the process of implementing this simulation framework, as well as the problems that were found on the way, blocking or non-blocking, and the solutions for these conflicts. Chapter 6 presents the experimental results of what was implemented. In chapter 7 some conclusions of the work done are taken. Finally, in chapter 8 is proposed some future work to compliment this thesis that can scale it to become a truly powerful simulation framework for V2X simulation, especially in the safety area.

Chapter 2

State of the Art

This chapter addresses the current state of the art on CAV, environmental perception, autonomous vehicle safety, V2X communication technologies, and related work to the topic of this thesis.

2.1 Concepts

In this section, the basic concepts related to the thesis are explained in order to introduce them for the topics that will be discussed ahead.

2.1.1 Connected and Autonomous Vehicles

A vehicle is considered connected when it is equipped with specific data transmission and receiving technologies which allows it to communicate with surrounding connected vehicles, infrastructure, network, and other connected devices. On the other side, a vehicle is considered autonomous when it is equipped with the same communication mechanisms, as well as a suite of sensors, such as cameras, LiDARs, radars and ultrasonic sensors. These sensors are used to perceive the surrounding environment and make decisions based on the information collected beyond basic reactive sense-act rules. The key difference between a connected and an autonomous vehicle is the ability of the latter to operate by itself, guided by safety suggestions from its decision-making algorithms.

All these sensors have strengths and limitations. Cameras provide high-quality visual information but are light-sensitive and prone to occlusions. LiDAR offers accurate 3D representations of the surroundings, but is challenging for reflective materials and is power and cost-hungry. Radar is weather-immune, but lacks resolution of the cameras and LiDAR. The sensors are merged together to create a comprehensive model of the current environment but that model is still constrained by what the sensors can "perceive", i.e., their visual line-of-sight (VLOS) and proximity of the environment.

2.1.2 Autonomous Vehicle Safety and Environmental Perception

These constraints become particularly difficult in hazardous conditions like blind curves, obstructed intersections, or multi-lane merges with reduced visibility. In these conditions, vehicle sensors may not be able to detect danger such as an incoming car, a stranded vehicle, or a pedestrian until the moment it is impossible to react safely. This leads to increased braking distances, evasive maneuvers, or even in worst-case situations, unavoidable crashes. Furthermore, weather conditions such as fog, rain, or direct sunlight degrade the sensor performance, aggravating perception challenges.

Aside from physical limitations, autonomous perception systems must also deal with the issues of unpredictability, sensor malfunction, and dynamic uncertainty. Environments around a vehicle are non-static, they consist of other cars, bicycles, and people, all of whom behave in complicated and often non-deterministic ways. To drive safely, an AV must not only know what is happening in its local surroundings, but also predict what might happen next, a task that grows increasingly difficult based solely on local sensor readings.

In addition to these technical challenges, the process of validating perception systems is itself a major hurdle. Real-world testing is expensive, time-consuming, and can be dangerous when dealing with edge cases or safety-critical scenarios. Even the most comprehensive test fleets cannot encounter every possible combination of environmental and traffic conditions, especially those that are rare but potentially catastrophic. As a result, simulation-based approaches have gained immense traction, providing a method for creating reproducible and controlled environments where AV perception and safety approaches can be researched in an intense fashion.

Yet, simulations based purely on sensor models are still plagued by the same fundamental blind spots found in real sensors. Thus, researchers and commercial developers are turning increasingly to cooperative approaches that lengthen the perception horizon beyond the horizon of individual vehicles. On this broader stage, integration of V2X communication systems holds the promise of autonomous vehicle safety. V2X enables vehicles to disclose information outside the immediate range of onboard sensors such as the approach of a vehicle behind an obstacle or the sudden braking of a vehicle out of view. By adding distributed situational awareness, V2X offers new possibilities for early detection of hazards, better decision-making, and finally, fewer and less serious car accidents.

2.1.3 Vehicle-to-Everything Communication

As the limitations of sensor-based perception in autonomous vehicles become increasingly apparent, researchers and engineers have turned to cooperative systems that can extend a vehicle's awareness beyond its physical and sensory constraints. Among these systems, V2X communication emerges as a transformative technology that enables real-time data exchange

between vehicles and other elements in the transportation ecosystem. By facilitating cooperation between road users and infrastructure, V2X has the potential to significantly improve safety, traffic efficiency, and decision-making in autonomous and connected vehicles.

V2X is an umbrella term that encompasses several communication modalities:

- V2V: direct communication between vehicles to share information such as position, velocity, acceleration, and intent.
- V2I: interaction with road infrastructure, such as traffic lights, road signs, and roadside units (RSUs), to receive information about signal phases, road conditions, or construction zones.
- V2P: communication with vulnerable road users, often via mobile devices, to improve safety in shared spaces.
- V2N: connection to cellular networks and cloud services for updates, navigation, and real-time traffic data.

Together, these communication channels establish a cooperative driving environment, in which vehicles are no longer isolated decision-makers but active participants in a distributed, context-aware system. This enables more informed and timely decisions, especially in critical scenarios where sensor-based perception alone may not suffice, such as when a vehicle is approaching a blind intersection or when an emergency vehicle is nearby but not visible.

The technological foundation of V2X communication has evolved significantly in the last decade. Two primary approaches have emerged: ETSI ITS-G5 (european standard) or Dedicated Short-Range Communications (DSRC) (american standard), based on the IEEE 802.11p standard, and Cellular-V2X (C-V2X), developed by the 3rd Generation Partnership Project (3GPP). DSRC offers low-latency communication in the 5.9 GHz band and has been widely tested in various pilot projects. However, its limitations in terms of scalability, range, and quality of service in dense environments have raised concerns about its long-term viability.

C-V2X, by contrast, leverages existing cellular infrastructure and includes two communication modes: PC5, for direct communication between vehicles or with RSUs without the need for a cellular base station, and Uu, which uses the cellular network to communicate with cloud services or distant infrastructure. C-V2X was introduced in 3GPP Release 14 and has continued to evolve, with 5G-V2X emerging as a key enabler of next-generation intelligent transportation systems.

5G-V2X brings substantial improvements in three critical dimensions: latency, reliability, and capacity. It supports Ultra-Reliable Low-Latency Communication (URLLC), which is essential for safety-critical applications, and massive Machine-Type Communication (mMTC), allowing

thousands of vehicles and infrastructure components to communicate simultaneously. Additionally, network slicing in 5G enables customized virtual networks with specific quality-of-service guarantees, which is especially relevant for differentiating between safety messages and non-critical data.

One of the most important contributions of V2X to autonomous vehicle safety is its ability to extend situational awareness beyond the vehicle's line of sight. For instance, a car approaching a blind curve may receive a message from another vehicle already within the curve, alerting it to slow down or change lanes. Similarly, infrastructure units can broadcast messages about adverse road conditions or upcoming traffic signal changes, allowing AVs to adjust their trajectories accordingly. These types of cooperative awareness messages reduce reaction time, improve decision-making under uncertainty, and provide an additional layer of redundancy to traditional perception systems.

In practice, V2X communication is structured around standardized message types, such as the Cooperative Awareness Message (CAM) and the Decentralized Environmental Notification Message (DENM) in Europe, or Basic Safety Messages (BSM) in the United States. These messages contain concise, frequently updated information about the vehicle's status and any detected hazards. The frequency and content of these messages are tightly regulated to ensure that safety-critical data is prioritized and that bandwidth is used efficiently.

Despite its potential, the widespread adoption of V2X faces several challenges. These include the interoperability between vehicles from different manufacturers, security and authentication of messages, data privacy, and the cost and complexity of deploying supporting infrastructure. Furthermore, as V2X is still under active development, regulatory frameworks and spectrum allocation policies vary across regions, complicating global harmonization efforts.

Nonetheless, V2X communication is increasingly seen as a cornerstone of future intelligent transportation systems. Its ability to supplement onboard sensors, enhance cooperative decision-making, and reduce accidents in complex environments makes it a vital component of any comprehensive autonomous driving strategy. As such, simulation environments that can accurately model V2X interactions, latency, and system response are essential for testing and validating these technologies before real-world deployment.

2.2 Related Work

This section explores relevant work that not only implements the aforementioned concepts but also takes advantage of pertinent technologies.

2.2.1 Production and Pilot Implementations of V2X by Automakers (Chronological Overview)

Over the years, several car manufacturers have slowly introduced the concept of V2X through pilot projects and eventual integration into production. Now, some context will be provided on

the integration of V2X communication in vehicles from these manufacturers, in chronological order.

2013 — Mercedes-Benz (Car-to-X via cloud; production)

Mercedes announced "Car-to-X Communication," initially using a smartphone-based solution ("Drive Kit Plus") to send/receive hazard information via a backend; later integrated into the head unit and expanded over time. Functions include early warnings (e.g., accidents, wrong-way drivers, roadworks) shown on the map and via audible prompts. This was one of the first series production Car-to-X deployments and has continued to add use-cases (e.g., pothole/speed-bump sharing in 2021) [6].

2015 — Toyota & Lexus (DSRC/ITS-G5-class @ 760 MHz; production/Japan)

Toyota's ITS Connect package (Prius, Crown, etc.) was the first mass-market OEM rollout using Japan's 760 MHz DSRC for V2V/V2I. Functions include Right-Turn Collision Caution, Signal Change Advisory/Red-Light Caution, Emergency Vehicle Notification, and Communicating Radar Cruise Control—i.e., messages from signals/roadside units and other vehicles beyond sensor line-of-sight. Toyota reported >100k DSRC-equipped vehicles on Japanese roads by March 2018 [7].

2016 — Audi (V2I over LTE backend; production/USA)

Audi launched Traffic Light Information (TLI) in Las Vegas (A4, Q7, Allroad with Audi connect). The car receives signal phase/timing data from the city's traffic management system over LTE, showing a countdown ("time-to-green") and advising the driver to avoid last-second acceleration. This was the first branded, city-scale V2I service offered to retail customers [8].

2016-2019 — Volvo (cloud V2V/V2N; production Nordics, then pan-Europe)

Volvo's Hazard Light Alert and Slippery Road Alert launched in 2016 on 90-series models in Sweden/Norway and expanded Europe-wide in 2019 (standard on MY2020; retrofittable on many SPA/CMA cars). Vehicles publish/consume anonymized hazard messages via the Volvo cloud to warn drivers beyond line-of-sight (e.g., over crests/blind curves). Volvo has since extended connected safety (e.g., 2024 Accident Ahead Alert using traffic-management center data) [9].

2017 — Cadillac (General Motors) (DSRC/5.9 GHz; production/USA)

The 2017 Cadillac CTS was the first U.S. production car with built-in V2V (DSRC). Vehicles broadcast/receive standardized safety messages to/from similarly equipped cars (e.g., hard braking, blind-intersection movement), enabling warnings even without sight or cellular coverage. GM positioned it as a step toward cooperative safety automation [10].

2017 — Groupe Renault & PSA (now Stellantis) (ITS-G5 pilots; France)

The SCOOP@F pre-deployment program equipped road corridors and fleets (e.g., 1,000 Mégane) to test C-ITS services (V2V/V2I) under real traffic. Use-cases included roadworks warnings, toll-plaza approaches, and work-zone safety. In 2024, French authorities announced the first V2X-equipped production vehicles sold to the public (PSA C4/DS4; Renault Mégane) for use on equipped networks—marking a transition from pilots to retail availability [11].

2019 — BMW (cloud V2N/V2V-like; production/EU)

BMW rolled out Local Hazard Warning across Europe, sharing anonymized safety events from connected BMW/MINI vehicles to warn others (e.g., broken-down vehicles, weather hazards). BMW also participates in multi-OEM Data for Road Safety initiatives with road authorities to turn vehicle sensor data into safety-related traffic information [12].

2.2.2 2019/2020 — Volkswagen (ITS-G5/802.11p; production/EU)

VW began Car2X as standard on the Golf 8 (Oct 2019) and then the ID. electric series, representing the largest series production of DSRC-class V2X in Europe. Features include automatic Local Hazard Warning (e.g., stationary vehicles, accidents) via direct short-range communications between vehicles and roadside units; the Golf 8 received a Euro NCAP Advanced award in 2020 for this system [13].

2020 — Buick (General Motors, China) (C-V2X; production/CN)

GM introduced C-V2X on the 2021 Buick GL8 Avenir (China), described as the first series-production vehicle with V2X in China. The system, built on GM's Global B/eCloud electrical architecture, supports V2V and V2I safety alerts and congestion information with high-rate data exchange [14].

2018–2021 — Multi-OEM & Regional Steps (pilots enabling scale)

Ford publicly committed to C-V2X (2019) and has run pilots (e.g., Wuxi, China) while U.S. spectrum policy evolved to formally allow C-V2X in the 5.9 GHz band (final rules in late-2024/2025), clearing a regulatory path to production rollouts [15].

Mercedes-Benz, BMW, Ford, Volvo ran a Europe-wide Car-to-X hazard-sharing pilot with HERE/-TomTom and transport authorities to pass acute hazard messages to drivers, complementing direct V2X [16].

2023 — Stellantis (V2X digital alerting via cloud; production/NA)

Stellantis enabled the Emergency Vehicle Alert System (EVAS) across approximately 1.8 million Chrysler/Dodge/Jeep/Ram vehicles (2018+ with Uconnect). Via HAAS Alert's Safety Cloud,

drivers receive in-vehicle warnings about nearby emergency responders and other roadway hazards, an OEM-wide, cloud-mediated V2X deployment at scale [17].

2024–2025 — Ongoing expansions

- Volvo added Accident Ahead Alert (first use of real-time traffic-center data to warn drivers directly in-car) [18].
- France (SCOOP) confirmed retail sale of V2X-equipped PSA/Renault vehicles near equipped corridors [19].
- U.S. regulators finalized C-V2X rules in the 5.9 GHz band (a key enabler for forthcoming OEM deployments) [20].

2.2.3 Vehicular Cooperative Perception

Presented in [21] is a comprehensive V2X-aided autonomous driving system aimed at significantly improving perception and planning capabilities in real-world traffic scenarios by incorporating Beyond Line-of-Sight (BLOS) information. The key motivation behind this system is to address the limitations of onboard perception, such as restricted sensor range and line-of-sight dependency by integrating cooperative awareness enabled through V2X communication technologies.

The proposed system architecture is structured into three tightly coupled subsystems that operate together to achieve robust and proactive autonomous driving behavior:

- The BLOS perception subsystem is responsible for fusing conventional onboard sensor data, such as from cameras, LiDAR, and radar with external information received via V2X communication. This includes data transmitted by other vehicles or infrastructure units (e.g., RSUs), allowing the vehicle to extend its awareness beyond its immediate surroundings;
- The extended planning subsystem dynamically adjusts the vehicle's planned trajectory and decision logic based on the combined perception data. This enables the vehicle to anticipate potential hazards or changes in the environment,such as a pedestrian suddenly crossing the road or an unexpected obstacle before these elements enter the direct field of view of its sensors;
- Finally, the control subsystem translates the high-level plan into executable actions, handling low-level commands for braking, steering, and acceleration. This ensures that vehicle behavior remains smooth and responsive, even in time-sensitive or complex traffic conditions.

To evaluate the effectiveness of this architecture, the system was deployed on a full-scale electric vehicle and tested in a series of real-world driving scenarios as part of the 2019 Hyundai Autonomous Vehicle Competition. The testbed included functioning V2X infrastructure, which

allowed for communication between the autonomous vehicle and the surrounding environment. The vehicle successfully completed a diverse set of missions that reflected common urban driving challenges. These included emergency braking in response to jaywalking pedestrians, rerouting to avoid static obstacles, obeying dynamic traffic signals, preventing collisions with cross-traffic, and yielding appropriately to emergency vehicles.

In each tested scenario, the integration of V2X-derived BLOS data enabled the vehicle to detect and respond to threats well before they entered the observable sensor range. According to the authors, perception messages were received consistently within the latency bounds required to synchronize with the onboard sensor fusion pipeline. This ensured that externally acquired data could be assimilated into the planning module in real time, without introducing delays or inconsistencies in the control loop.

The successful implementation and testing of this system offer strong validation for the use of V2X communication as a complementary source of perception in autonomous driving. The results demonstrate not only the technical feasibility of BLOS-based situational awareness, but also its practical benefits in enhancing decision-making and overall system robustness in real-world, infrastructure-supported environments.

2.2.4 V2X Cooperative Perception Survey

Presented in [22] is an in-depth and structured review of V2X Cooperative Perception (CP), positioning it as a key enabler for advancing autonomous vehicle safety and efficiency in complex and dynamic traffic environments. The paper not only consolidates a wide array of methodologies but also offers a systematic framework for understanding how cooperative perception is currently implemented, what technologies enable it, and which challenges must still be overcome to achieve large-scale deployment.

At the core of the review is a taxonomy of cooperative perception architectures, categorized along three main axes: agent collaboration, data fusion, and communication reliability. First, in terms of agent selection and collaboration strategies, the paper distinguishes between different roles in the V2X ecosystem, including ego vehicles, follower vehicles, RSUs, and cloud-based nodes, and discusses how these entities can be prioritized or dynamically assigned as information providers depending on the context. This includes considerations of proximity, sensor quality, and communication reliability, all of which affect the utility and trustworthiness of shared data.

Second, the survey provides a detailed exploration of data alignment and fusion techniques. These are broadly divided into three categories:

- Early fusion, where raw sensor data such as LiDAR point clouds or camera feeds are shared directly between agents;
- Intermediate fusion, which focuses on processed features or detected objects;

- Late fusion, where agents share high-level semantic interpretations, such as intention or maneuver predictions.

Each approach has its trade-offs. Although early fusion offers the most detailed data, it requires a high bandwidth and strict temporal synchronization. Late fusion, in contrast, is lighter in communication load but less flexible for downstream decision-making. The review also emphasizes the importance of data timestamp alignment, spatial reference consistency, and confidence estimation, especially when integrating asynchronous observations from heterogeneous sources.

Beyond technical strategies, the paper addresses several key challenges for effective V2X-CP. These include:

- Heterogeneity in sensor types, processing capabilities, and communication protocols across different vehicles and infrastructure;
- Latency and synchronization issues, which can degrade the quality of fused information or render it obsolete in highly dynamic contexts;
- Packet loss and communication unreliability, particularly in urban environments with multipath interference or high channel contention.

To address these limitations, the authors discuss enabling technologies that have recently gained traction. Notably, edge computing is presented as a scalable solution for real-time data aggregation and low-latency decision support, enabling more localized and timely fusion. The review also explores the potential of blockchain for ensuring data integrity and provenance in decentralized CP systems, digital twins for simulating sensor and communication behavior in digital replicas of the physical environment, and emerging 6G paradigms, which promise massive bandwidth, microsecond-level latency, and AI-native communication protocols tailored for autonomous systems.

Furthermore, the paper contributes to the field by proposing a generic V2X-CP framework intended to unify how cooperative perception components are organized. This includes agent roles, communication paths, fusion logic, and application layers. The authors also survey available datasets and simulation tools, identifying a lack of public resources for testing CP systems under varied, real-time, multi-agent conditions.

Importantly, the review concludes by identifying open research questions that must be addressed for widespread deployment. These include establishing trust management models, improving privacy protection during data exchange, and optimizing integration efficiency with existing AV architectures. The insights provided in this survey form a strong foundation for guiding future research and system design, including simulation frameworks like the one developed in this thesis, which aims to model cooperative hazard awareness using V2X under constrained visibility conditions.

2.2.5 Scalability and Communication Performance

In [23] there is an analysis of the limitations of LTE-V2X (3GPP Release 14) under heavy traffic scenarios and proposes several enhancements to improve performance under dense vehicle deployments. Key aspects include:

- Scalability analysis: Through comprehensive modeling and simulation, the authors show that standard LTE-V2X Mode-4 with PC5 sidelink suffers from significant congestion, packet loss, and increased latency as vehicle density grows;
- Enhanced strategies:
  - Extended frequency bandwidth: increasing channel resources to reduce contention;
  - Congestion-based transmission rate control: dynamically adapting message rates to network load;
  - One-shot transmission method: aiming to reduce message collisions by avoiding redundant repetitions.
- NR-V2X introduction: They include a survey of 3GPP Release 16 5G New Radio (NR) sidelink and its potential to address enhanced V2X (eV2X) use cases with URLLC requirements.

Thanks to these techniques, simulation results reveal marked improvements in packet delivery ratio (PDR) and latency—maintaining safety-critical performance in high-density environments. The study does not include empirical NR-V2X experiments but articulates how innovations like URLLC, advanced resource allocation, and multi-band operation in NR-V2X can resolve current bottlenecks.

[24] explores the security landscape of 5G-V2X, comparing it with LTE-V2X and proposing a novel Security Reflex Function (SRF) architecture. Their main contributions include:

- Technology comparison:
  - LTE-V2X, while a major step forward, lacks robustness for ultra-reliable, low-latency safety messaging.
  - 5G-V2X introduces URLLC and network function virtualization, addressing both latency and throughput requirements alongside stronger security foundations.
- SRF-based architecture:
  - The SRF module is proposed as an intermediary security layer within the 5G core network to dynamically verify and sanitize V2X messages before forwarding, improving upon LTE threshold-based security.
- Security requirements for UD-US:

  - The authors define criteria for "Ultra-Dense, Ultra-Secure" V2X communications, highlighting needs such as mutual authentication, key management, and privacy in V2X ecosystems.

- Open research areas:
  - Challenges identified include secure sidelink resource management, privacy-preserving message schemes, and policy-based security profiles for safety-critical contexts.

By establishing a systematic view of 5G-V2X security models, [24] illuminate both the potential and shortcomings of emerging systems, laying a foundation for secure, large-scale V2X deployment strategies.

2.2.6 Systematic Surveys on V2X in Autonomous Driving

Systematic surveys on V2X communication have played a key role in consolidating knowledge and identifying trends in its integration with autonomous driving. These works highlight how V2X enables vehicles to share information about their environment, intentions, and actions, supporting cooperative perception and decision-making, particularly in complex or low-visibility scenarios.

Recent surveys, such as [25, 26], classify cooperative perception approaches based on the type of data shared (e.g., raw sensor data, object lists, or planned trajectories) and the fusion method applied. These studies emphasize the benefits of non-line-of-sight (NLOS) awareness and improved reaction times but also point to challenges in data fusion, latency, synchronization, and trust.

From a communication technology perspective, surveys compare IEEE 802.11p (DSRC), LTE-V2X, and 5G-V2X, concluding that 5G-V2X is the most promising for safety-critical applications due to its support for ultra-reliable low-latency communication and massive device connectivity.

Another common insight is the lack of high-fidelity simulation environments that can accurately model both vehicle behavior and realistic V2X messaging. This limitation strengthens the motivation for simulation platforms,like the one proposed in this thesis, that use Unreal Engine 5 and eCAL to evaluate V2X effectiveness in scenarios where traditional perception is insufficient.

2.3 Real-Time Escalator for Event-Based Simulators

Unreal Engine 5 and OMNeT++ follow different notions of time, and that difference is the core reason why, when used together, they do not automatically form a real-time simulator. Unreal Engine is designed for interactive real time: it updates and renders the virtual world in a sequence of frames that aim to keep pace with the physical clock so that what the user sees and what the simulation computes feel “live”. OMNeT++, by contrast, is event-driven:

it advances simulation time by jumping from one scheduled event to the next, regardless of how long those events take to compute in the real world. In other words, UE5 tries to move forward in steady, clock-like steps, while OMNeT++ moves forward in logical steps that depend on when events are supposed to happen in the simulated timeline. Simply wiring these two engines together does not resolve this difference, instead, it exposes it.

This mismatch matters for any study that claims “real-time” behavior. In a real-time setting, results produced “now” should correspond to the same “now” on both sides of the system. If UE5 advances a visual/physical frame that represents the present moment, but OMNeT++ is processing a burst of events that conceptually belong to a slightly earlier or later instant in the simulated network, the combined result no longer reflects a single, consistent present. Small drifts of this kind may be acceptable for exploratory or qualitative demonstrations, but they undermine the stronger claim that the whole framework runs in real time with predictable timing and consistent causality.

Bridging this gap requires more than good intentions or careful coding, it requires a dedicated scheduler that coordinates how and when each engine advances, as shown in [27]. At a high level, such a scheduler would establish a common sense of time, decide when UE5 is allowed to take its next frame and when OMNeT++ is allowed to process the next batch of events, and ensure that both are looking at and producing outputs for the same moment. It would also need to handle the everyday realities of simulation: sometimes the network side will be momentarily “busy” with many events, sometimes the graphics and physics side will take longer than expected to render a complex scene. In those cases, the scheduler would have to choose whether to slow one side down, let the other wait, skip non-essential work, or otherwise reconcile the difference to keep both engines aligned.

Although this description sounds straightforward, designing such a scheduler in a robust and general way is a substantial piece of work. One must define the rules for alignment (what counts as “the same moment”?), the policies for dealing with delays (what gives way when time runs tight?), and the safeguards for preserving consistency (how to ensure that no side consumes information that belongs to a future it has not reached yet). These decisions also need to be implemented across two mature systems with their own update cycles and constraints, and then validated with careful testing so that the coordination holds not only in simple cases but also under heavier, more variable loads.

There is a practical dimension as well. Unreal Engine 5 can be configured to aim for steady frame pacing, but it still runs on general-purpose operating systems and standard GPUs, where background activity, resource contention, or complex scenes can occasionally make a frame take longer. OMNeT++ can simulate very quickly or quite slowly depending on how many events are scheduled at a given simulated time. The scheduler would have to cope gracefully with these fluctuations while maintaining a clear, shared sense of “now”. Getting those behaviors right so they are predictable, explainable, and repeatable, requires iteration, measurement, and refinement.

For the purposes of this thesis, the engines are integrated to enable meaningful experiments and to study how information flows between a vehicle/world simulator (UE5) and a communication/network simulator (OMNeT++) can enhance crash safety in autonomous vehicles. This provides useful insights, but it does not constitute a full real-time framework. Achieving that would require the dedicated scheduler described above, along with the design choices, implementation effort, and verification work that go with it. That is an extensive undertaking in its own right and would shift the focus away from the research goals pursued here.

Chapter 3

Technologies

This chapter introduces the key technologies that were essential for the development of this project, as well as the study of some alternatives that were considered but were not used. These technologies include simulation engines, middlewares, communication technologies, serialization formats, and communication and network simulators that allowed us to correctly decide the options that fit this project better without relying on physical hardware. The chapter concludes with a critical analysis that contrasts each of these technologies and explains why each was chosen.

3.1 Simulation Engines

This section presents an overview of the simulation engine used for the development of this thesis, as well as some others used for similar projects, and a comparison between the engines mentioned.

3.1.1 Unreal Engine 5

The simulation component of this thesis relies to a great extent on UE5, a state-of-the-art, real-time 3D creation tool from Epic Games. UE5 is a wide-brush improvement over the previous releases of the software, including new features such as Nanite (virtualized geometry) and Lumen (real-time global illumination), all while retaining the firm foundation of modular architecture, scripting capability, and extensibility that rendered its precursors welcoming to scientific simulation, architectural visualization, and digital twin applications.

UE5 offers a huge canvas to build intricate, interactive, and photorealistic simulaions to simulate driving in realistic ways in a virtual controlled setting. The reason UE5 was selected for this thesis is mainly due to its capability to produce realistic and dynamic environments required to replicate hazardous driving situations, e.g., blind corners and low-visibility intersections where V2X communication can make a tangible difference in avoiding crashes and enhancing safety.

And another strong selling point of UE5 is that it supports both visual scripting in Blueprints and in regular C++ programming, thus allowing agile and scalable development approaches based on system component complexity and performance needs.

Simulating autonomous vehicle behavior in hazardous scenarios requires a platform capable of handling several interdependent subsystems:

- Realistic vehicle physics: Realistic kinematics, collision dynamics, and actuator modeling.
- Environmental fidelity: Dynamic road geometries, occlusion conditions (e.g., blind curves),
- Environmental factors, such as weather conditions, and lighting changes,
- Message-driven control: Capacity to react to outside occasions, for example, warnings received through V2X communication.
- Data integration: Support for middleware-based communication (e.g., eCAL) and serialization frameworks (e.g., Protobuf).
- Extensibility and modularity: Support for rapid development and future research expansion.

UE5 is perfectly placed to meet these needs, namely because it inherently has both graphical interaction as well as high-performance backend logic, which are of highest importance in the integration with vehicle behavior modeling of V2X messages.

C++ in Unreal Engine 5: Precision, Performance, and Integration

Unreal Engine is written in C++ at its core, and most of its internal functionality is revealed via a robust and extensible C++ API. In the context of this thesis, C++ is the first choice for developing all performance-critical and system-relevant parts. This encompasses subscribing to V2X messages sent via the eCAL middleware, deserializing structured data formats via Protobuf, performing real-time logic, such as assessing collision threats and activating safety measures, and providing thread-safe execution for minimizing latency and preserving deterministic behavior throughout the simulation runtime.

One of the primary benefits of using C++ in UE5 is its execution speed. As C++ is a compiled language, it allows for high-performance computing that is essential in real-time systems, where delays need to be kept at a minimum in order to have proper responses from autonomous agents. In addition, C++ provides low-level control over memory management, which is especially useful when working with high-throughput data streams. Another essential benefit is its native compatibility with third-party libraries and systems. This aspect is key to the aims of this thesis since the communication interface with eCAL is managed completely in C++, mirroring the architecture of real-world embedded automotive systems. Finally, Unreal Engine's C++ environment allows for sophisticated debugging and profiling workflows using tools like Visual Studio, Unreal Insights, and logging macros that help diagnose performance bottlenecks, memory problems, and multi-threading issues.

Unreal Engine provides different types of classes to represent the various roles within a simulation. Actors are the most common type, in order to implement anything which can be added to the world, such as a barrier, traffic light, or car. Pawns are a subclass of Actor which can be owned and controlled, typically for cars or characters. Actor Components are reusable pieces of functionality that can be attached to Actors or Pawns, allowing developers to encapsulate such things as sensors, movement controllers, or communication modules. Scene Components also exist, being spatial transformations such as position, rotation, and scale, and serving as attachment points for other components. At a more general level, Game Mode class specifies the rules of the simulation and Player Controller class deals with user or AI input. Together, these types of classes provide a flexible hierarchy and allow simple world objects as well as interactive complex systems to be represented in an orderly fashion.

However, programming in C++ also has some drawbacks. The language demands a stronger knowledge of object-oriented programming concepts, in addition to a good understanding of Unreal Engine's inner workings, e.g., its actor-component model, garbage collector, and reflection macros such as UCLASS and UFUNCTION. The iterative development process is also slower by nature, as every modification to the codebase has to be recompiled and, sometimes, the engine has to be restarted, potentially delaying testing and prototyping. Furthermore, the flexibility of C++ brings with it a higher risk of runtime errors, particularly when working with manual memory management, concurrent tasks, or asynchronous callbacks. In spite of these limitations, the requirements of this project, specifically the need for low-latency communication, integration with external systems, and predictable behavior make C++ the best choice.

Blueprints: Visual Scripting for Prototyping and Interactivity

In addition to C++, UE5 has a strong visual scripting language called Blueprints. Blueprints are a node-based interface that enables developers to define behaviors, handle events, and prototype functionality without the need for handwritten code. In this project, Blueprints are consciously kept for use in those tasks that don't need direct system resource access or execution performance. Rather, they are used as a useful tool for quickly developing and experimenting with visual components and simulation flow control.

Blueprints are utilized to setup and control the simulation world, aspects like traffic lights, barriers, and scene configuration. They are also responsible for spawning vehicles with preconfigured properties and control on-screen overlays that represent internal system states, e.g., communication warnings, alerts, or the state of safety features. Due to their integration into the Unreal Editor, changes implemented via Blueprints can be applied and tested instantly, which makes them a great fit for prototyping user interface and high-level logic. Their accessibility also enables people with little programming background, e.g., designers or domain experts, to directly participate in the development process.

For all their usefulness, Blueprints have some limitations that their use in this thesis is limited to non-critical functionality. Since they are evaluated at runtime instead of being compiled, they carry a performance penalty that is unacceptable for safety-critical, real-time logic. Their scalability is also problematic: as Blueprint graphs increase in size and complexity, they become increasingly difficult to maintain, test, and refactor. In addition, not all engine-level functionality or third-party integrations are exposed to Blueprints, so they cannot be used to interface with eCAL, process Protobuf messages, or conduct low-level thread operations.

3.1.2 SUMO

SUMO (Simulation of Urban MObility) is an open-source microscopic traffic simulator developed by the Institute of Transportation Systems at the German Aerospace Center. It is widely utilized in research and industry for modeling and analyzing traffic dynamics, mobility patterns, and the effects of transportation systems and infrastructure on urban traffic flow. SUMO is particularly well-suited for studying vehicle interactions, evaluating traffic management strategies, and testing communication technologies such as V2X.

One of the defining aspects of SUMO is its microscopic simulation approach, where each vehicle is simulated individually with position, speed, acceleration, and route. This level of detail enables close study of vehicle behavior and interaction, and therefore, SUMO is an effective tool for examining the impact of cooperative driving, intelligent traffic lights, or priority for rescue vehicles.

SUMO takes many different input formats, including real-world road network data (e.g., from OpenStreetMap), road geometries custom-designed manually, and traffic demand models. The simulation runtime is highly parameterizable, with parameters controlling traffic light logic, vehicle types, driver behavior, and routing algorithms. It also includes tools for route generation, network editing, and visualization.

Especially, SUMO is typically interfaced with external systems and frameworks through its TraCI (Traffic Control Interface) protocol, which allows runtime interaction with the simulation via TCP. This facilitates dynamic vehicle control, injection of data from external sources, or coupling SUMO with higher-level systems such as traffic optimization algorithms, multi-agent platforms, or network simulators such as Veins and OMNeT++.

In V2X studies, SUMO is commonly used to model traffic situations in which connected vehicles exchange data with infrastructure or other vehicles. Although SUMO does not simulate radio communication or network-level dynamics by itself, it is frequently used in combination with tools like Veins or Eclipse MOSAIC for full-stack V2X scenario simulation, combining traffic flow simulation and network communication layers.

3.2 Middlewares

This section presents the middlewares that were considered to integrate this project, as well as comparison between them highlighting which one is the best fit and the reason it was chosen.

3.2.1 ROS2

Robot Operating System 2 (ROS2) was chosen as the middleware framework for this project at the outset due to its modular structure, community support, and aptness for distributed and real-time robotic systems. Based on the Data Distribution Service (DDS) standard, ROS2 provides major advancements over its predecessor, such as real-time support, fine-grained Quality of Service (QoS) control, and multi-platform support for embedded, desktop, and cloud platforms.

Within the fields of autonomous cars and V2X-based safety applications, ROS2 presents some very strong reasons to be chosen. The publish-subscribe architecture of ROS2 allows for easy communication among sensors, perception modules, control nodes, and external systems. ROS2's abstraction layers and broad ecosystem of tools (e.g., rviz2, rosbag2, and rqt) also speed up development and testing in simulated environments.

However, despite these strengths, significant limitations emerged during implementation, particularly related to compiling ROS2 on Windows. Although ROS2 officially supports Windows, the build process proved to be unstable and error-prone, with numerous dependencies failing to compile correctly. Maintaining ROS2 workspaces would become increasingly time-consuming as the project scaled, with frequent build failures and incompatibilities between packages. These compilation challenges would ultimately hinder the development efficiency and system stability, prompting the transition to an alternative middleware solution.

Because of these limitations, the project switched to a different middleware: eCAL. eCAL provided a lighter-weight, more stable, and more platform-independent solution more suited to this thesis's simulation requirements, especially for compatibility with UE5 and for providing stable message exchange in V2X use cases.

However, ROS2 architectural underpinnings and design philosophy, especially its focus on modularity, real-time behavior, and QoS-based communication significantly shaped the overall middleware approach taken in this study.

3.2.2 eCAL

eCAL is a high-performance middleware framework for real-time, distributed systems with strict communication demands. It was initially created by Continental for in-house automotive prototyping and embedded use. eCAL has now grown into a mature, open-source solution with C++, Python, and C# support, being actively developed and used in many fields, most notably robotics, industrial automation, and automotive software.

At its core, eCAL provides a highly efficient publish-subscribe communication model, similar in conceptual design to the middleware layer of ROS2. It supports both inter-process (local) and inter-host (distributed) communication and allows systems to exchange structured messages in a decoupled, asynchronous manner. This architecture is especially useful in modular system designs, where different components, which may be written in different languages or running on separate nodes must collaborate in real time without being directly connected.

A key advantage of eCAL is its simplicity and minimal runtime overhead. Unlike ROS2, which is built on top of the DDS standard and requires extensive configuration and dependency management, eCAL uses a lightweight internal communication stack. This design removes the need for external discovery services or complex Quality of Service (QoS) tuning, resulting in faster deployment and greater reliability, particularly on Windows platforms. eCAL supports inter-process communication via shared memory for processes running on the same host (processes in eCAL are the equivalent to nodes in ROS2). For both local and distributed systems, it also enables communication over UDP multicast and TCP, ensuring efficient message exchange regardless of physical deployment.

Both eCAL and ROS2 support structured message formats through Google Protocol Buffers (protobuf). Developers define .proto schemas that are compiled into native classes, enabling strongly typed, language-agnostic message passing. Because of this, eCAL is particularly appealing for use cases that require integration with systems written in other languages or operating systems must be seamless while maintaining data consistency and serialization efficiency. eCAL also provides handy tooling like eCAL Monitor (live inspection of topics and message flow) and eCAL Rec/Play (data recording and playback), as well as support for remote procedure calls (RPC). This makes it a good fit for both development and testing in simulation environments.

In the context of this thesis, ROS2 was initially chosen as the middleware communication between simulation components. However, due to the problems stated before, ROS2 became impractical for continued use. eCAL was adopted as an alternative thanks to its native Windows support and stability. With eCAL, it was possible to simulate V2X message exchange reliably, enabling realistic cooperative perception.

3.3 Communication Technologies

In this section, the communication technologies considered for this thesis will be discussed in depth, stating their differences, as well as every advantage and disadvantage of each them.

3.3.1 ETSI ITS-G5

ETSI ITS-G5 is the European communication standard designed to support Cooperative Intelligent Transport Systems (C-ITS). It is based on the IEEE 802.11p amendment but developed and standardized in ETSI specification, largely EN 302 663 for the access layer and EN 302

665 for the overall architecture. The system operates in Europe's reserved 5.9 GHz frequency band for road safety and traffic efficiency services and is optimized for highly dynamic vehicle environments where ultra-low latency is needed.

A prominent feature of ITS-G5 is its cooperative awareness mode of operation. Instead of relying on cellular infrastructure, it enables direct safety message exchange between vehicles (V2V) and between vehicles and roadside units (V2I). The benefit of this direct mode of operation is that the information such as sudden braking, hazardous road conditions, or the detection of vulnerable road users can be shared almost in real-time even when there is no network coverage.

The communication model is ETSI ITS protocol stack-based, with functionality being divided into access, networking transport, and facilities layers. Within the facilities layer, generic message formats such as Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) represent a shared vocabulary between infrastructure and vehicles to exchange information with one another. Security and trust are also primary design considerations, with ETSI specifications offering authentication and integrity of safety-critical data exchanges.

Even though the technology adopts the contention-based medium access of 802.11p and induces channel overload in the case of extremely dense traffic, although its applicability in actual deployments has been proven and established successful via pilot tests in Europe. Regulators continue to address coexistence issues with other 5 GHz band services, yet ITS-G5 remains the reference technology for C-ITS rollout in Europe.

3.3.2 DSRC

Dedicated Short-Range Communications is one of the firsts well-established communication technologies developed with the intention of being implemented for vehicular communication. DSRC was developed with basis on IEEE 802.11p standard, which is an amendment to the IEEE 802.11 protocol (Wi-Fi). It was designed to operate in high-speed, highly mobile environments that require low latency, operating on the 5.9 GHz frequency band, which is known as the ITS band.

The primary benefit of DSRC is that it eliminates the need for intermediary network infrastructure by enabling direct communication between vehicles and RSUs (V2V and V2I, respectively). This decentralized architecture enables communications with extremely low latencies that are critical for safety applications such as collision avoidance and emergency braking, as well as possibly for the application studied in this thesis, triggering safety mechanisms via V2X.

DSRC employs a contention-based medium access mechanism like Wi-Fi, supporting fast message propagation but potentially resulting in packet collisions in extremely dense environments. Notwithstanding this constraint, DSRC has been subject to thorough testing and standardization, especially within the United States, with the Federal Communications Commission (FCC) initially allocating spectrum for DSRC-based applications. More recently, however, portions of the DSRC spectrum have been reassigned to other uses, which has affected the momentum for DSRC adoption.

From a protocol standpoint, DSRC accommodates a full stack of communication layers for vehicular environments. This encompasses the IEEE 1609 family of standards, which offer services like security (IEEE 1609.2), networking (1609.3), and multi-channel operations (1609.4). These standards facilitate secure, authenticated sharing of information among vehicles, a key requirement for trust in safety-critical situations.

Competing technologies, particularly Cellular-V2X (C-V2X), which provide greater scalability and integration with current cellular infrastructure, have made DSRC more challenging to deploy, despite its use in pilot projects and infrastructure in some locations. When ultra-low latency and infrastructure independence are essential, DSRC remains a practical and technologically sophisticated choice for local vehicular communication.

3.3.3 Cellular-V2X

Cellular Vehicle-to-Everything is a communication technology developed by the 3rd Generation Partnership Project (3GPP) to support the needs that were arising from connected and autonomous vehicles (CAV). Unlike DSCR that relies on Wi-Fi-based communication, C-V2X relies on cellular network technologies to offer a large range of interactions, including V2V, V2I, V2P, and V2N.

One of the main features of C-V2X is that it is capable of working in two different modes. The first one is direct communication over the PC5 interface, where vehicles can exchange data directly with other vehicles or infrastructure without relying on the cellular network (base station) to do so, which is especially useful for time-critical applications like the ones mentioned previously and the one studied throughout the thesis, where extremely low-latency is required. The second one uses the Uu interface that connects the vehicles to the cellular network, which allows for cloud-based services, broader data sharing and communication over longer distances. This, of course, causes the latency in the exchanges to increase, which makes it not suitable for safety-critical applications.

One of the biggest advantages of C-V2X is that it benefits from the global mobile ecosystem, meaning that mobile network operators, chipset manufacturers and automotive companies are already aligned around the 5G roadmap, which means there is strong momentum for implementing this technology at scale. Furthermore, C-V2X is advantageous from a deployment and cost standpoint because it can be implemented using the current infrastructure.

QoS and security are also significant advantages. C-V2X can support features like message prioritisation, secure communication, and authentication right out of the box because it makes use of mobile network technologies. These features are essential for safety-realted applications.

LTE-V2X

LTE-V2X was introduced by 3GPP in Release 14 as the first generation of C-V2X communication. It was developed as an alternative to DSRC, offering both direct and and network-based communication via the existing cellular infrastructure. The objective was to support basic safety applications, like the ones already mentioned.

One of the main strengths of LTE-V2X was its improved performance in terms of range and quality in NLOS coditions when compared with DSRC. Better scalibility was also demonstrated in dense traffic environments. The use of cellular modulation and coding schemes allowed for more robust and reliable communication, especially in urban scenarios.

Nonetheless, LTE-V2X also had limitations. It was well-suited for safety alerts and awereness messages, but its latency and data throughput were not enough for more advanced cooperative driving use cases. Additionally, its resource allocation mechanism on the PC5 interface was static and could become inefficient under heavy traffic loads.

5G NR-V2X

5G NR-V2X (New Radio Vehicle-to-Everything) is the 5G-based evolution of C-V2X communication, that was introduced in 3GPP Release 16. Although it was built on the same priciples and interfaces as LTE-V2X, 5G NR-V2X is the representation of a major shift in both performance and functionality, that aims to support a broader range of connected and automated driving use cases.

Delivering ultra-reliable low-latency communication (URLLC), which is crucial for autonomous and cooperative driving, is one of the main improvements brought to 5G NR-V2X. Applications such as initiating safety mechanisms via V2X and other time-critical and safety-critical applications are made possible by its higher data throughput, improved spectral efficiency, and latencies as low as 1 millisecond.

NR-V2X offers more dynamic and flexible communication by support both periodic and event-driven messaging, in contrast to LTE-V2X that was primarily focused on BSM. Another enhancement provided by NR-V2X is that it adds new sidelink communication modes that provide support for unicast, groupcast, and broadcast transmissions. This enables vehicles to provide data to selected peers instead of broadcasting it, which reduces the channel congestion.

Even though 5G NR-V2X has a lot of potential, it also comes with some new challenges. Since it relies on 5G infrastructure and radio access technologies, its deployment is more complex

than LTE-V2X and is often limited to urban areas. Also, backward compaibility with LTE-V2X is not guarenteed at the air interface level, which complicates technology transitioning periods.

Nonetheless, the general architecture is still in line with LTE-V2X, including the utilisation of the PC5 interface for direct communication and Uu interface for network-based communication. Manufacturers can gradually switch from LTE-based to NR-based systems over time thanks to this continuity, which also makes higher-level integration easier.

3.4 Serialization Formats

In this section, the serialization formats considered for the purpose of interprocess data exchange will be analysed and compared with each other.

3.4.1 XML

XML, which stands for eXtensible Markup Language was introduced in 1998 by the World Wide Web Consortium (W3C) as a standardised format for representing structured data in a way that is independent from the platform. It was designed with flexibility and extensibility in mind, which allows for users to define their own tags and document structures.

XML's self-descriptiveness is one of its main advantages. Because tags give data semantic meaning, XML documents are easier to comprehend and validate with the use of tools like XML Schema and DTD (Document Type Definition). In early web services and enterprise systems, where stringent data validation and interoperability were crucial, this feature was especially helpful.

But the verbosity of XML soon became a noticeable disadvantage. Large message sizes brought on by the widespread use of opening and closing tags lengthen processing times and increase transmission overhead. Additionally, parsing XML can be computationally costly, particularly in systems that require quick turnaround times or environments with limited resources. Because of these drawbacks, XML is being used less frequently in applications that are performance-sensitive.

3.4.2 JSON

JSON, or JavaScript Object Notation was introduced early in 2000s by Douglas Crockford as an alternative to XML for data exchange. JSON is derived from JavaScript syntax and provides a simpler and more compact way to represent structured data using key-value pairs, arrays, and nested objects. Its minimal grammar and readability have contributed to its rapid adoption across a large range of applications and programming languages.

Unlike XML, JSON is much less verbose, which results in smaller payload sizes and less parsing time. It is especially useful for web APIs and microservices, where fast data exchange and low

bandwidth usage are crucial. However, JSON has its limtations, it lacks built-in schema enforcement, making it harder to vaidate or guarantee the structure and types of data exchanged. Additionally, as it is a text-based format, JSON is less efficient than binary serialization alternatives when it comes to efficiency, parsing speed, and message size.

3.4.3 Protocol Buffers (Protobuf)

Protocol buffers, commonly referred to as protobuf is an efficient, language-agnostic, platform-agnostic, binary data serialization mechanism developed by Google, similiar to JSON and XML but smaller, faster, and simpler. Protobuf enables developers to define structured data in a .proto file, that is then used to generate code capable of reading and writing data from different data streams. Protobuf was originally developed by Google engineers looking for efficient ways to serialize structured data across multiple internal servers, known as Proto1. Proto2 was the initial public release that quickly gained traction leading to its third and current version, Proto3, released in 2016.

Protobuf is fundamentally schema-driven, in contrast to text-based formats like XML and JSON. This means that developers define data structures using a specific language (.proto files), which are then compiled into source code in a variety of programming languages. This method provides back-and-forth compatibility, efficient serialization and deserialization, and strong typing.

Protobuf's compact binary serialization is one of its primary benefits. Compared to their JSON or XML counterparts, serialized Protobuf messages are substantially smaller, which is crucial in systems where network efficiency is an issue. Protobuf's versioning support is another asset. If developers adhere to the established field numbering guidelines, compatibility can be maintained even if new fields are added or older ones are deprecated over time.

3.5 Communication and Network Simulators

In this section, various simulators capable of simulating V2X data exchanged will be addressed, analyzing the features of each one as well as their limitations.

3.5.1 OMNeT++

Building network simulators is the main purpose of OMNeT++, an extensible, modular, component-based C++ simulation toolkit and framework. "Network" refers to a wider range of networks, such as queueing networks, on-chip networks, wired and wireless communication networks, and so forth. Developed as separate projects, model frameworks offer domain-specific functionality including support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modelling, photonic networks, etc. Among its many tools, OMNeT++ provides a graphical runtime environment and an Eclipse-based integrated development environment.

Among other things, there are extensions for database integration, SystemC integration, network emulation, and real-time simulation.

In the context of vehicular networks and V2X communication, OMNeT++ is most commonly used alongside frameworks such as Veins (Vehicles in Network Simulation), which couples OMNeT++ with the traffic simulator SUMO via the TraCI protocol. This co-simulation approach allows researchers to simulate realistic traffic behavior in parallel with wireless communication between vehicles, infrastructure, and other agents. OMNeT++ handles the communication stack, while SUMO models the mobility of each vehicle in the environment. This makes OMNeT++ a powerful tool for testing V2X protocols under dynamic traffic conditions.

Despite not being a network simulator in and of itself, OMNeT++ has become a popular platform for network simulation in both scientific and industrial contexts, and it has amassed a sizable user base. OMNeT++ provides a component architecture for models. C++ is used to program the components (modules), which are then put together using a high-level language (NED) to create larger components and models. Model reusability is free of charge. Because of OMNeT++'s modular architecture and wide range of GUI support, integrating the simulation kernel (and models) into your applications is simple.

OMNeT++'s simulation engine makes it particularly suitable for evaluating delay, throughput, message frequency, and reliability in V2X communication. Furthermore, its support for modular protocol stacks allows researchers to model both IEEE 802.11p (used in DSRC) and LTE/5G-based C-V2X protocols, depending on the simulation framework in use. This flexibility, combined with visualization tools and logging capabilities, makes OMNeT++ an essential part of many V2X research pipelines.

3.5.2 Veins

Veins (Vehicles in Network Simulation) is an open-source framework designed to enable bidirectional, real-time coupling between traffic and network simulation, specifically targeting vehicular ad hoc networks (VANETs) and broader V2X scenarios. It integrates the OMNeT++ network simulator with the SUMO traffic simulator using a runtime protocol known as TraCI (Traffic Control Interface), which allows both simulators to run in parallel and exchange data during execution.

At its core, Veins operates by synchronizing two key simulation domains:

- Mobility simulation, conducted by SUMO, calculates the movement of each vehicle on a detailed road network, incorporating realistic traffic dynamics, traffic signals, driver behaviour models, and route choices;
- The wireless communication stack, message propagation, radio channels, and protocol behaviour between automobiles, infrastructure, and other network nodes are all modelled using OMNeT++, which handles network simulation.

The synchronization between SUMO and OMNeT++ is achieved via TraCI, a TCP-based API that allows OMNeT++ to control vehicle behavior in SUMO (e.g., inserting or removing vehicles, changing routes or speeds), and conversely, for OMNeT++ to access real-time mobility data such as vehicle positions, speeds, and headings. This feedback loop enables network events to influence traffic behavior and vice versa — a core requirement in realistic V2X research.

Veins uses OMNeT++'s modular architecture to define vehicles as networked nodes, where each vehicle is typically modeled as a compound module containing layers for the application logic, communication protocols (e.g., IEEE 802.11p), MAC and PHY layers, mobility, and antenna models. Veins also includes models for realistic radio wave propagation, such as path loss, fading, and interference, as well as support for event-triggered message transmission based on vehicle actions or external stimuli.

Because Veins is agnostic to the underlying communication technology, researchers can extend or modify it to test various V2X technologies — from DSRC-based protocols to experimental LTE/5G-V2X implementations (though cellular support is often added through frameworks like SimuLTE or Artery). Vehicles can generate periodic beacons (CAMs, BSMs), event-driven messages (DENMs), or more complex application-layer traffic, depending on the study.

Veins is widely used in academic research for studying:

- Cooperative awareness and perception,
- Congestion control and channel access strategies,
- Infrastructure-assisted driving (V2I),
- Security and trust models in VANETs,
- The impact of communication delays or failures on traffic behavior.

One of Veins' key strengths is its realism — combining real-time, fine-grained traffic dynamics from SUMO with highly configurable communication models in OMNeT++. This makes it especially suitable for analyzing the interplay between traffic flow and network performance, which is critical in evaluating the effectiveness of V2X systems under realistic conditions.

3.5.3 Artery

Artery is an open-source framework designed to enable cooperative ITS simulations by coupling the traffic simulator SUMO with the OMNeT++ network simulator. Built on top of the Veins framework, Artery extends the simulation capabilities by providing a complete ETSI ITS-G5 protocol stack and additional application layers, thereby allowing realistic modeling of V2X communication scenarios. In practice, Artery enables vehicles simulated in SUMO to be represented as network nodes in OMNeT++, where they can generate, transmit, and receive CAMs and DENMs according to the ETSI standards. This simulation engine allows the study of both vehicular mobility and communication aspects in a consistent simulation environment, supporting research in cooperative awareness, traffic efficiency, and safety applications. Because

Artery leverages standardized protocol implementations, it is frequently used in academic and industrial studies that require a close-to-reality representation of vehicular networking behavior.

3.5.4 ns-3

ns-3 (Network Simulator 3) is a widely used open-source discrete-event network simulator that is primarily aimed at research and educational use. It is a potent tool for simulating computer network behavior, allowing researchers to model complex network topologies, protocols, and traffic patterns with high fidelity.

Unlike its predecessor ns-2, ns-3 is written using contemporary C++ and Python bindings with improved modularity, scalability, and performance. It is an extensible framework that enables users to insert custom modules or edit existing modules to meet specific research needs. This renders ns-3 particularly handy for wireless network, Internet protocol, and emerging technology such as 5G and IoT simulations.

One of the key strengths of ns-3 is realism. It supports integration with real network stacks and real-world emulation tools, allowing simulations to interact with live systems. This capability bridges the gap between theoretical modeling and actual implementation, making ns-3 a top choice for validating protocol designs and performance analyses.

Chapter 4

Architecture and Concept

This chapter presents the overall concept developed as well as the global architecture of the intended system. It describes the proposed use-case scenario that supports the development of the simulation and explains how the system components interact and communicate with each other.

As stated before, this thesis focuses on the design and development of a proof-of-concept simulation system to recreate dynamic urban environments where it is possible to test whether vehicle intercommunication can improve the response time of pre-crash safety actions and airbag deployment.

4.1 Concept and Use Case

To better understand the main idea behind this work, a preliminary use-case scenario was defined. This scenario consists of the cooperation between two vehicles on a collision route, sending their data (position, velocity, and intention) to one another in order to evaluate if the communication latencies allow the activation of safety measures via V2X.

A common real-world example is a blind curve or a road intersection with no visibility. In these situations, it is impossible to detect another vehicle approaching solely through traditional sensors such as LiDAR, radar, or cameras, since all of them are restricted by their visual line of sight (VLOS).

In practice, the vehicles use their communication interfaces to transmit relevant data directly to each other or to the infrastructure, allowing nearby vehicles to obtain information that might prevent a collision. In cases where an impact is unavoidable, this communication may still enable faster crash detection and trigger pre-crash measures earlier, such as airbag deployment.

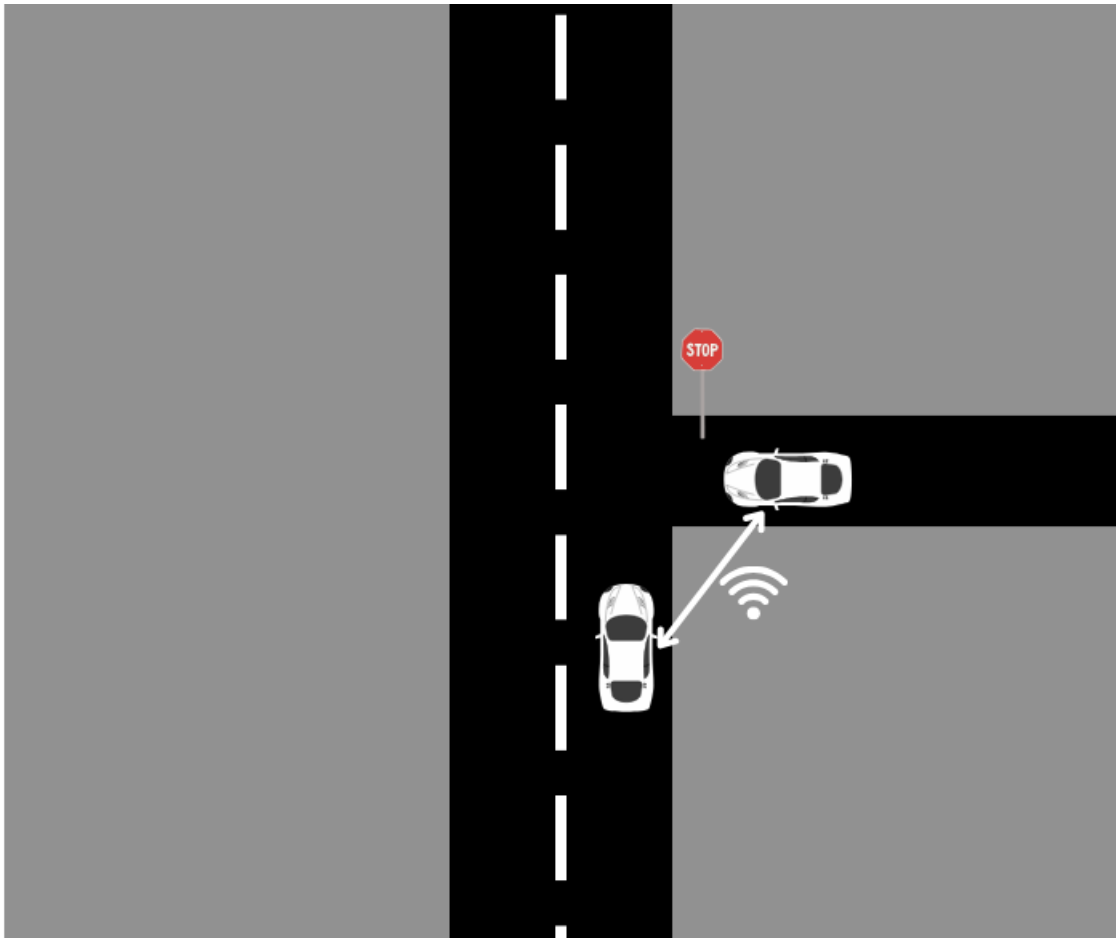


Figure 4.1: Project use case.

In conclusion, the developed simulation aims to reproduce low-visibility or no-visibility scenarios where inter-vehicular communication and cooperation can enhance safety. This approach involves using vehicle communication units to exchange information, evaluate latency, and verify whether V2X-based actions can effectively support safety mechanisms.

4.2 System Architecture

The proposed concept is implemented through a modular simulation architecture that integrates three main technologies: UE5, OMNeT++, and eCAL. These technologies play the following roles:

- **UE5**: Simulates the scenario and handles the physical aspects of the simulation;
- **OMNET++**: Responsible for simulating the communications between vehicles, infrastructure, and other equipment capable of this type of communication;

- **eCAL**: Establishes communication between the different components of the simulation via shared memory.

Each component interacts as depicted in Figure 4.2, representing the global system architecture. Vehicle data transmission is achieved using 5G communication either through the network (Uu interface) or directly (PC5 interface) between vehicles. The RSU gathers data from passing vehicles and shares it with other nearby equipment accordingly. For more critical use cases, direct communication is preferred to achieve lower latencies.

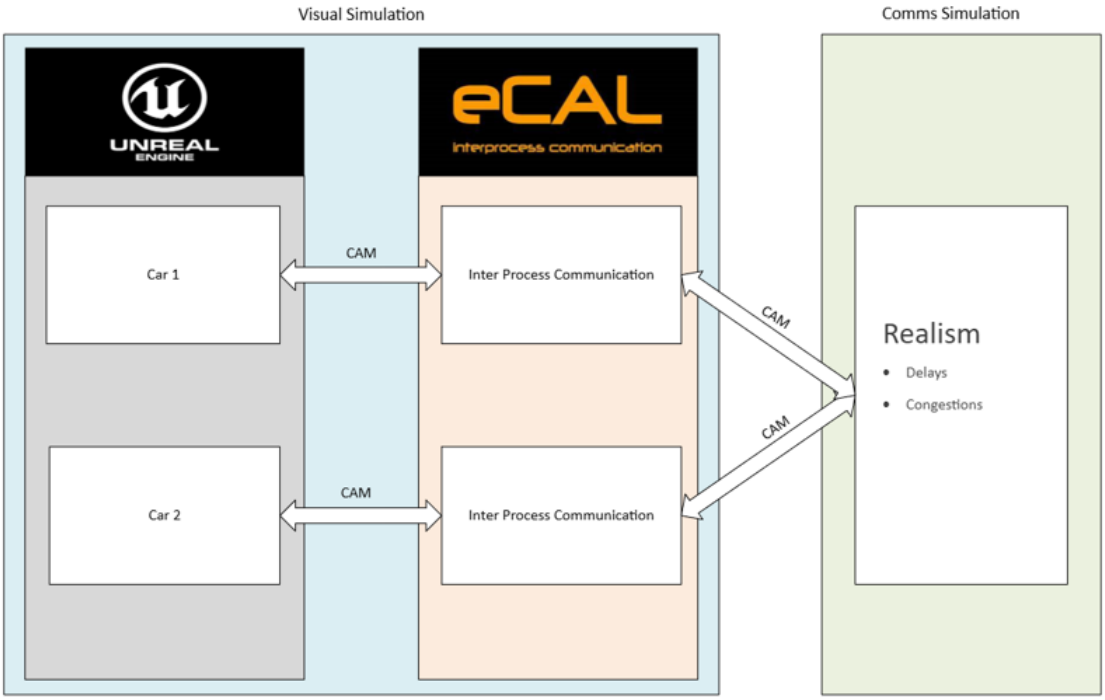


Figure 4.2: System architecture.

This simulation framework enables the study of cooperation scenarios focusing on:

- High-quality, high-resolution scenario creation;
- V2X, especially V2V communication.

These aspects will be further detailed in the following chapters. The presented use-case scenario illustrates how inter-vehicular communication can enhance vehicle safety and extend perception beyond the vehicle's VLOS. Furthermore, they demonstrate the utility of the simulation framework in evaluating whether the communication systems meet the stringent time constraints associated with specific safety-critical scenarios.

Chapter 5

Development

This chapter focuses on the development of the simulation framework using UE5, eCAL, and OMNeT++. Firstly, the UE5 simulation development is described, followed by the integration of eCAL in UE5 and OMNeT++. As some of this development was not finished due to some problems and conflicts, there will be further ahead another chapter describing these in detail as well as some possible ways to work around them.

5.1 UE5 Simulation Scenario

This section overviews the development of the simulation in UE5, as well as every component and plug-in used and how everything works together.

To develop this project, the it was decided to use version 5.5 of Unreal Engine. This version comes with templates already prepared that we can start prototyping from. For the purpose of this thesis it was used the "Vehicle Template", which already features a car track and a wheeled vehicle that is input-controlled. Since we could start from there, the next steps were identified, which were as follows:

- Spawn a second vehicle;
- Create a way for the vehicles to follow a path;
- Make the vehicle follow a path instead of being controlled by an input.

5.1.1 Spawn Vehicles

The template that was utilized in the project already contained a fully operating vehicle blueprint created with the Chaos Vehicles plug-in, which is the most widely used plug-in for the creation and definition of wheeled vehicles in Unreal Engine. The default vehicle was not only visually modeled but also came with a well-defined and highly detailed physics setup, making it an ideal starting point for our own use. Aspects such as the chassis layout, suspension, torque profiles, motion of the wheels, and engine characteristics had already been optimized, offering realistic dynamics without needing to manually tweak them on a grand scale. By leveraging

this pre-made asset, we avoided the hassle of importing a community resource custom vehicle skeleton and then painstakingly creating and tuning all of its accompanying elements, including collision hitboxes, drivetrain properties, and joint limits.

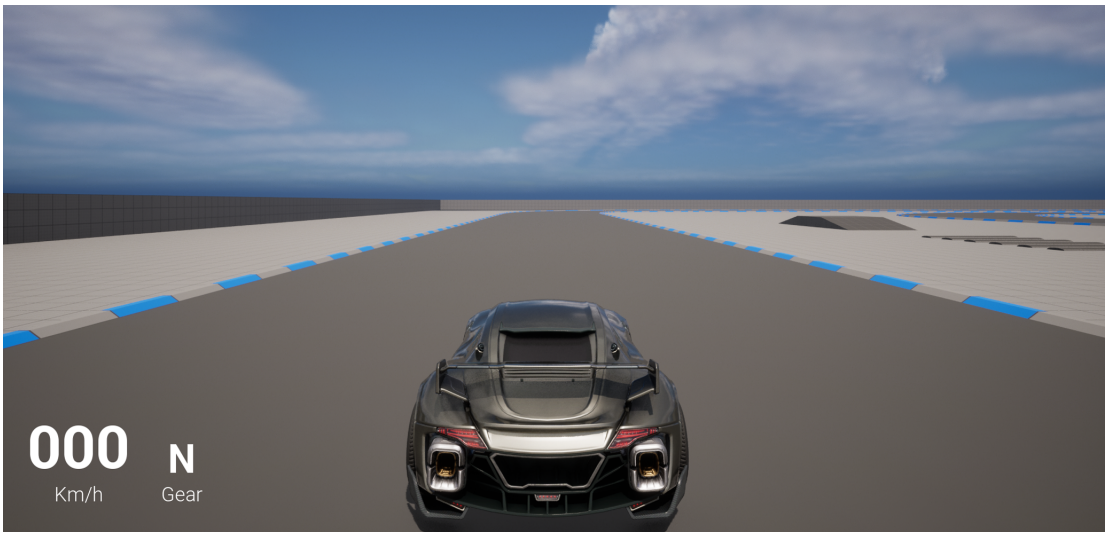


Figure 5.1: Vehicle from the template.

With the template already including a solid and realistic reference vehicle, extending the simulation with a second vehicle was simple. Instead of having to start from scratch, we simply duplicated the original blueprint, adjusted its properties as needed, and dropped it at the precise location where it needs to be when the simulation is initialized. This significantly sped up development, allowing us to focus on more theoretical elements of the simulation, such as vehicle interaction, communication protocols, and scenario creation, rather than squandering valuable time over low-level vehicle configuration.

5.1.2 Path Design

In order to make the vehicles in the simulation follow a predefined path, it is first necessary to find a way to trace that path so that the actors (vehicles) can interact with it.

The most commonly used method in UE5, as well as in other simulation engines, is to trace the path using splines. In Unreal Engine 5, a spline is an editable curve defined by a group of control points, used to create smooth paths or shapes in a 3D world. Splines are especially useful for guiding the movement of objects, such as vehicles following a road, or for generating geometry like roads, pipes, or fences that conform to a curved path. They can be manipulated directly in the editor by adjusting their control points and tangents, allowing developers to design complex paths visually. Splines can also be accessed in code or Blueprints to retrieve positions, rotations, or directions along the curve.

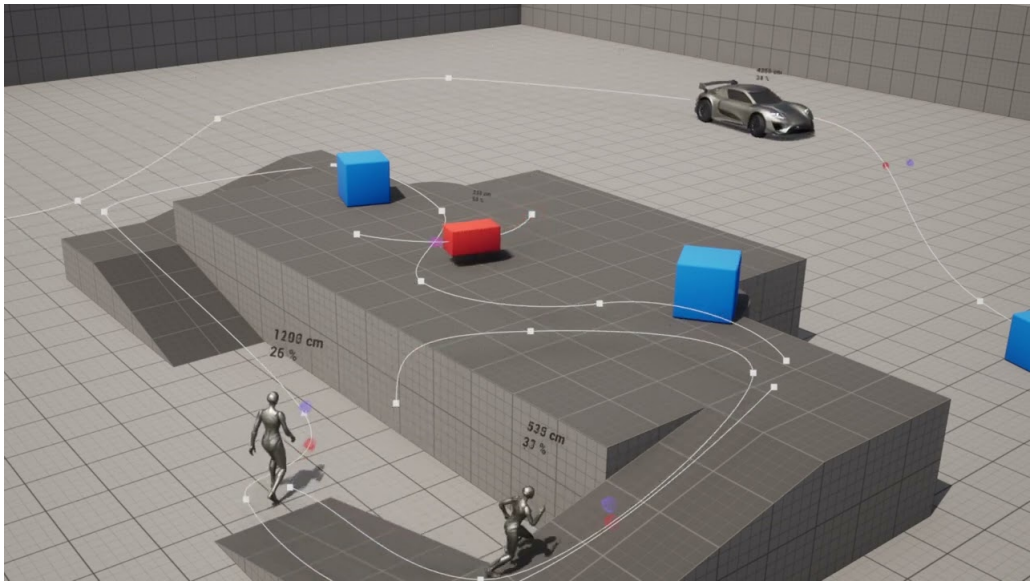


Figure 5.2: Example of a Spline [28].

To build a spline path in UE5, we first declare a new Actor class that owns a USplineComponent. In the header file (.h), we expose the spline as a UPROPERTY(VisibleAnywhere) so it is editable in the level editor. In the source file (.cpp), the constructor craetes and attaches this USplineComponent to the actor's root and sets sensible defaults, such as open/closed loop, debug drawing, point types), and in OnConstruction, which is the class that is called when the world is being constructed and rendered, we ensure the spline has valid control points and call UpdateSpline() so its cached length, points, tangents, and interpolation are correct.

If this is done correctly, there is now a new actor in the Content Browser, with whichever name was given, that can be dragged and placed in the simulation world. This component represents the editable path curve in the world, allowing control points and tangents to be adjusted either in the source file or through the visual editor, which was the preferred approach. Once placed in the scene, the spline provides a continuous path along which actors can be positioned or moved.

5.1.3 Path Following

With the vehicle actors already spawned and the spline path drawn, the only thing missing is making these actors follow said spline. As stated before, the Vehicle Template from UE5 comes with input-controlled vehicles, which means that the first step was to find the input map that was binding each action to the key on the user's keyboard.

These bindings were found on ProjectNamePawn.cpp (with ProjectName meaning the name of the current project) which already revealed which were the functions needed to control vehicle functions, such as throttle, steering, braking, and handbraking. Then there were two ways to control the vehicle, with C++ code, or with Blueprints. The first attempts were made in

C++, which unfortunately was deemed not viable, especially due to the lack of documentation from Epic Games regarding the Vehicle Template, and also from the community, as the majority opts for Blueprints. After attempting to backtrack each function through the template it was consuming too much time, which was becoming a critical parameter to consider at this point.

The documentation regarding Blueprints at that time was much more extensive than C++, from both Epic Games and also its community, which led to a second attempt of making the vehicle follow the spline path, but this time using the Blueprints.

In order to develop a reusable component to follow the spline, a new Actor Component was created and built using Blueprints, as shown in Figure 5.3. To generate steering inputs that allow the vehicle to follow a spline, the Blueprint first queries the spline component associated with the active path and retrieves the current position of the vehicle in world space. Using this position, it determines the tangent of the spline at the closest point, which represents the local driving direction of the path. The tangent vector is normalized and scaled by a look-ahead distance which is a parameter (in this case 500 cm) to define a target position down the road. The offset from the current position of the car is added and projected back onto the spline to ensure that the target is always along the desired path, even when momentarily off it. A desired orientation is then computed by calculating the "look-at" rotation from the vehicle's position to the projected point on the spline. The difference between this desired orientation and the current rotation of the vehicle is a yaw error, which quantifies how far off the path the vehicle is. Finally, this angular error is scaled from a range (-45° to +45°) to a normalized scale (-1 to +1), producing a steering input that can be directly used on the vehicle control system. In practice, this gives continuous and smooth steering corrections, with small angular displacements giving proportional inputs and large errors saturating to full steering angles, giving strong path following behavior.

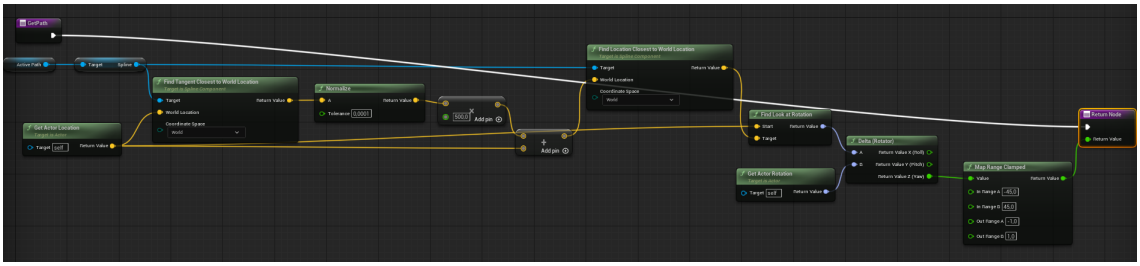


Figure 5.3: Blueprint to follow the spline.

5.2 eCAL Integration

In this section we will explain the process that led to a successful integration of eCAL in Unreal Engine 5, composed of two subchapters, creating the publisher and the subscriber.

5.2.1 eCAL Publisher in UE5

As stated in 3.2.2, eCAL is a high-performance, lightweight, publish-subscribe middleware framework for real-time distributed systems. In order to establish communication and send the vehicle data to other Actors inside the simulation or to external components, eCAL needs to be integrated into the UE5 project.

The first step to integrate this middleware into the project was to include its libraries and source files in the project dependencies (.Build.cs file). To do this we needed a folder for all the external dependencies, in this project it was named "include". Navigating to the eCAL root folder we copied all the dependencies from there to the "include" folder in the project, where it is now accessible to the developer.

Having this, the next step was to define the path to the "include" folder inside the project and using PublicAdditionalLibraries.Add() and PublicDelayLoadDLLs.Add() to include all the necessary dependencies, combining the path with the name of the dependency, for example:

- Static libraries: PublicAdditionalLibraries.Add(Path.Combine(LibPath, "ecal_core.lib"));
- Dynamic libraries: PublicDelayLoadDLLs.Add(Path.Combine(BinPath, "Continental.eCAL.Core.dll")).

With all the necessary dependencies included in the project, we could start developing the code for the publishers and subscribers to test data exchange between different processes. Given that eCAL installation comes with the applications already mentioned, including eCAL Monitor, it made sense to start by developing a publisher for protobuf messages, as we could verify its correct functioning with the monitor.

To achieve this, we created a new C++ Actor, so we could place it in the simulation world to gather data from there and bring it to the outside. The first step to do this was to create a .proto file specifying the message the would be sent, for this thesis purposes, it included the following twelve parameters:

- float pos_x: Position on the X axis;
- float pos_y: Position on the Y axis;
- float pos_z: Position on the Z axis;
- float vel_x: Velocity along the X axis;
- float vel_y: Velocity along the Y axis;
- float vel_z: Velocity along the Z axis;
- float speed: Overall scalar speed of the vehicle;
- int32 hours: Timestamp hours component;
- int32 minutes: Timestamp minutes component;
- int32 seconds: Timestamp seconds component;

- int32 milliseconds: Timestamp milliseconds component;
- int32 microseconds: Timestamp microseconds component.

Now compiling this file like this: protoc −−cpp_out=.VehicleState.proto, the output is two different files, VehicleState.pb.cc and VehicleState.pb.h. In order for the different processes to understand each other, they all need both these files in their directory.

Starting with the header file, we had to declare a timer in order to send periodic messages, and declare a Protobuf message publisher (eCAL::protobuf::CPublisher<vehicle::VehicleState> VehiclePublisher), where the message type is also specified, referring to the .proto file explained above.

Then on the source file, the first step was to initialize eCAL with eCAL::Initialize(0, NULL, "UE5Publisher"), which returns 0 if the initialization failed and 1 if it succeeded. After this we needed to create a protobuf publisher VehiclePublisher = eCAL::protobuf::CPublisher<vehicle::VehicleState>("vehicle1"), which also specifies that it will communicate following the format specified in "VehicleState" message type from the "vehicle" package, and publish it to the "vehicle1" topic. We then create a method to gather the data from the desired vehicle, create a message that follows the specified format, and publish it to the topic in question. Having this, the timer was set up in order to publish a message each 0.05 seconds with the data from the vehicle inside the simulation. The source code for the timer can be seen in Listing 5.1 and the one for the method that collects and publishes data can be seen in Listing 5.2.

```
GetWorld()->GetTimerManager().SetTimer(
    MessageTimerHandle,
    this,
    &APublisherActor::PublishVehicleState,
    0.05f,
    true
);
```

Listing 5.1: Timer to call the publisher method.

With the publisher ready, the only thing left to do was drag the actor from the Actors tab to the simulation world and then build the simulation and run it. Once the simulation is running and the car going around the track following the spline path designed previously, it was possible to see on eCAL Monitor the process "UE5Publisher" which included the topic "vehicle1" with one active publisher, posting the data from the vehicle along the timestamp at which the message was sent, shown in Figure 5.4. The timestamp will be relevant in the future when we want to know the latency of the messages.

```
for (TActorIterator<APawn> It(GetWorld()); It; ++It)
  {
    APawn* Pawn = *It;
    if (!Pawn) continue;

    FString Name = Pawn->GetName();

    FVector Position = Pawn->GetActorLocation();
    FVector Velocity = Pawn->GetVelocity();
    float SpeedKmh = Velocity.Size() * 0.036f;

    vehicle::VehicleState state;
    state.set_pos_x(Position.X);
    state.set_pos_y(Position.Y);
    state.set_pos_z(Position.Z);
    state.set_vel_x(Velocity.X * 0.036f);
    state.set_vel_y(Velocity.Y * 0.036f);
    state.set_vel_z(Velocity.Z * 0.036f);
    state.set_speed(SpeedKmh);

    if (Name == "AI_Car_C_0")
    {
      double TimeSeconds = FPlatformTime::Seconds();
      int64 TotalMicroseconds = static_cast<int64>(TimeSeconds * 1'000'
    000);

      int32 Hours = (TotalMicroseconds / (60 * 60 * 1'000'000)) % 24;
      int32 Minutes = (TotalMicroseconds / (60 * 1'000'000)) % 60;
      int32 Seconds = (TotalMicroseconds / 1'000'000) % 60;
      int32 Milliseconds = (TotalMicroseconds / 1'000) % 1000;
      int32 Microseconds = TotalMicroseconds % 1000;

      state.set_hours(Hours);
      state.set_minutes(Minutes);
      state.set_seconds(Seconds);
      state.set_milliseconds(Milliseconds);
      state.set_microseconds(Microseconds);

      VehiclePublisher.Send(state);
    }
  }
```

Listing 5.2: Method to publish the vehicle state.

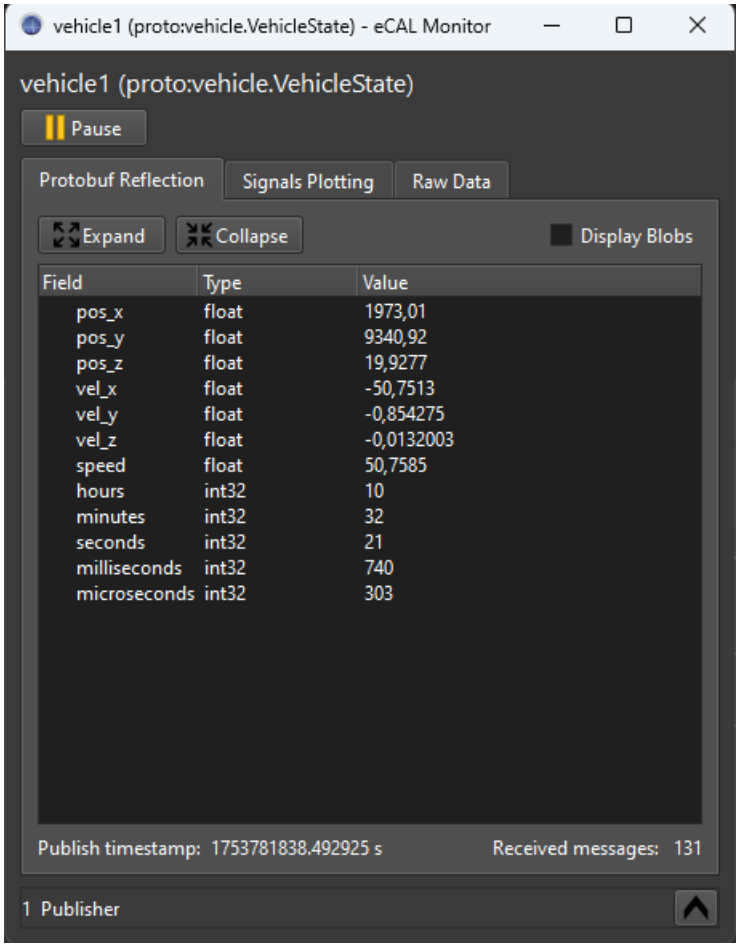


Figure 5.4: Vehicle data being published from UE5.

5.2.2 eCAL Subscriber in Unreal Engine 5

With the publisher functioning correctly, the next step is to implement a new UE5 actor that will act as a subscriber to the "vehicle1" topic. This actor will listen for incoming messages and process the data transmitted from the vehicle, ensuring that the state information published earlier can now be received and utilized within the simulation or outside of it.

After a few attempts at creating a subscriber we quickly realized that it was better to run it on a separate thread, as blocking message reception would otherwise interfere with the main game loop, causing stutters and preventing the simulation from updating smoothly. Considering this, the thread started being written in the header file, where we declared the constructor "WorkerTask()" and its corresponding destructor " WorkerTask()", the callback function that will be executed whenever a new "VehicleState" message arrives from the publisher, this callback function also receives the topic name and the message itself. Then we declare a pointer to an eCAL subscriber object that listens for messages of type vehicle::VehicleState, which is how the thread receives vehicle state updates. On the private section we declare a pointer to the actual Unreal Engine thread object (FRunnableThread) that runs this worker. With this,

we can start, stop, or manage the worker thread (e.g., Kill(), Suspend()). All this can be seen in Listing 5.3 below.

```
public:

  WorkerTask();

  virtual ~WorkerTask() override;

  bool bRunThread = true;

  bool Init() override;
  uint32 Run() override;
  void Stop() override;

  void OnReceiveVehicleState(const char* topic_name_, const vehicle::
    VehicleState& msg_, const long long time_, const long long clock_);

  eCAL::protobuf::CSubscriber<vehicle::VehicleState>* VehicleSubscriber;

private:

  FRunnableThread* Thread;
```

Listing 5.3: Header file where the thread and subscriber are declared.

Moving on to the source file of the actor,we start by creating a variable called Worker that can point to an object of type "WorkerTask". Then on the BeginPlay method we allocate a new worker task object on the heap and assign its address to the pointer "Worker". When the simulation is stopped, the EndPlay function runs so we use it to invoke the WorkerTask destructor, which by itself does not free the memory allocated, so inside the destructor we have to first stop the thread, so that after we can kill it safely and then delete it, freeing up all the memory that was allocated to it.

```
WorkerTask* Worker;

void ASubscriberActor::BeginPlay()
{
  Super::BeginPlay();

  Worker = new WorkerTask();
}

void ASubscriberActor::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
  Super::EndPlay(EndPlayReason);

  if (Worker)
  {
    Worker->~WorkerTask();
  }
}

WorkerTask::~WorkerTask()
{
  if (Thread)
  {
    Stop();

    //Kill() is a blocking call
    Thread->Kill();

    delete Thread;
  }
}
```

Listing 5.4: Lifecycle management of a worker thread.

With the management of the thread completed, a callback function was implemented to receive and parse the data coming from the publisher. Inside this function, the timestamp included in the sent message is compared against the time at which it was received, allowing us to analyze and measure the latency of the messages. This process is shown in Listing 5.5.

```
void WorkerTask::OnReceiveVehicleState(const char* topic_name_, const
    vehicle::VehicleState& msg_, const long long time_, const long long
    clock_)
{
  posX = msg_.pos_x();
  posY = msg_.pos_y();
  posZ = msg_.pos_z();

  velX = msg_.vel_x();
  velY = msg_.vel_y();
  velZ = msg_.vel_z();

  speed = msg_.speed();

  double TimeSeconds = FPlatformTime::Seconds();
  int64 TotalMicroseconds = static_cast<int64>(TimeSeconds * 1'000'000);

  int32 subHours = (TotalMicroseconds / (60 * 60 * 1'000'000)) % 24;
  int32 subMinutes = (TotalMicroseconds / (60 * 1'000'000)) % 60;
  int32 subSeconds = (TotalMicroseconds / 1'000'000) % 60;
  int32 subMilliseconds = (TotalMicroseconds / 1'000) % 1000;
  int32 subMicroseconds = TotalMicroseconds % 1000;

  hours = msg_.hours();
  minutes = msg_.minutes();
  seconds = msg_.seconds();
  milliseconds = msg_.milliseconds();
  microseconds = msg_.microseconds();

  UE_LOG(LogTemp, Warning, TEXT("Publisher Timestamp: %d.%d:%d.%d.%d"),
    hours, minutes, seconds, milliseconds, microseconds);

  UE_LOG(LogTemp, Warning, TEXT("Subscriber Timestamp: %d.%d:%d.%d.%d"),
    subHours, subMinutes, subSeconds, subMilliseconds, subMicroseconds);

}
```

Listing 5.5: Message parsing and timestamp comparison.

After building the UE5 project with both actors (publisher and subscriber), the subscriber was not receiving any messages, even though the publisher was working correctly and all the parameters in the subscriber were set correctly. It turned out that Unreal Engine compiles its projects as one single executable, which causes the simulation to be one single process. eCAL has the loopback setting disabled by default, which did not allow the subscriber to subscribe to a publisher on the same process. In order to avoid this problem, in the subscriber source code we had to change this setting by doing eCAL::Util::EnableLoopback(true). As a result, the data exchange between processes occurred seamlessly and continuously.

5.3 OMNeT++ Integration

This section details our attempt at development of the simulation in OMNeT++, but focusing more on the problems encountered that prevented the proper development of the simulation as well as the integration in the framework.

5.3.1 Problems in the integration

After studying and comparing various network simulators, we selected OMNeT++ as it proved to be the most flexible, offering the best documentation and strong community support, including existing V2X simulation projects. The installation and setup of OMNeT++ presented no obstacles, allowing us to complete it quickly and move on to exploring its functionality and programming through some simple projects.

The difficulties began when we attempted to integrate eCAL into OMNeT++, as library conflicts quickly emerged. Since the computer in use was running a Windows operating system (as previously mentioned), the eCAL libraries had been compiled with MSVC, whereas OMNeT++/INET on Windows is typically built using MSYS2/MinGW with the GCC toolchain. As a result, the two libraries were incompatible, preventing us from linking eCAL directly within an OMNeT++ module. Furthermore, because the IT policy also restricted the use of TCP/UDP in "network mode," we were constrained to local mode (shared memory), leaving no straightforward way to bridge the gap through sockets.

As the project timeline was already approaching its limits, it became clear that there was no straightforward way to overcome this blocking issue. Consequently, we were unable to move forward with the planned integration, properly test the framework, or design and implement a concrete use-case for further study.

5.3.2 Possible Alternatives

Although there was no time left to successfully develop an alternative to overcome the problem stated above, we made a study of possible alternatives to explore in order to have a fully functional simulation framework. This study resulted in four possibilities.

Emulate the network inside eCAL

It was possible to create a lightweight intermediary process (a "shim") that subscribes to the original eCAL topic(s), applies configurable network effects (fixed/variable latency, jitter, loss, duplication, reordering), and republishes on a new topic. Parameters (e.g., mean latency, standard deviation, drop rate, burstiness) can be supplied via CLI and loaded at runtime. Timestamps should be captured on ingress and egress to compute the exact end-to-end delay introduced by the shim. The shim remains entirely in local mode (shared memory), thus bypassing IT policies that blocked TCP/UDP, and avoids any ABI/toolchain conflicts because it links only against the same Windows/MSVC eCAL libraries as the rest of the application.

While this approach fails to simulate a full protocol stack, it produces reproducible and controlled delay and loss conditions close to LTE/5G/V2X ranges and enables more meaningful performance assertions than with raw shared-memory latencies. It also introduces simplicity and portability, as it runs on a single toolchain and operating system, enabling simple deployment in the corporate environment. But it has its drawbacks since it is not link, MAC, and PHY-level realistic—i.e., it does not model congestion, channel contention, scheduling, propagation, and interference—and network effects are statistically emulated rather than derived from actual network behavior.

Cross-boundary bridge without external sockets

One solution would have been to run OMNeT++/INET inside WSL2 (Linux) and keep eCAL on Windows with the two worlds being bridged without employing raw TCP/UDP sockets, which were being prohibited by IT policy. Other non-network IPC mechanisms could be employed instead. For example, a Windows mini-bridge process could read from eCAL and output frames on a named pipe, and a corresponding WSL2 helper would read from the pipe (relayed or through wsl$ integration) and feed the events into OMNeT++ through a special input module, such as reading from stdin or a FIFO. Or, shared directories (wsl$) could provide memory-mapped files or append-only binary logs, with the Windows bridge prepending framed messages and timestamps, and the OMNeT++ side appending to the files, retrieving the data, and scheduling events appropriately. Time synchronization could be guaranteed by timestamping each message with wall-clock and simulation-time hints, which the OMNeT++ module would then use to preserve ordering and compute effective queuing delays.

This method would make the integration feasible under the given policy constraints, as it requires no external sockets and still allows real eCAL traffic to feed into OMNeT++. Compared to a simple shim, it would also improve model fidelity by enabling INET/Veins to simulate aspects such as contention, routing, and per-hop behavior once the messages are inside the simulator, while the bridge handles only the ingress and egress. However, the trade-off would be added complexity and fragility, since multiple runtimes and IPC mechanisms introduce more potential failure points. Care would need to be taken with buffering, back-pressure, and event ordering to avoid artificial artifacts that could compromise the accuracy of the results.

Align toolchains

The ABI mismatch could have been removed by recompiling eCAL and its dependencies (e.g., Protobuf, Asio) with MSYS2/MinGW (GCC) so they link directly into OMNeT++/INET on Windows. Practically, this means configuring CMake with the MinGW generator, producing .a/.dll artifacts, and ensuring correct symbol export annotations (e.g., __declspec(dllexport) where required). After producing the MinGW-compatible libraries, they would be validated with unit tests and then linked into a custom OMNeT++ module. A theoretical alternative—compiling OMNeT++ with MSVC—exists, but it is generally unsupported or fragile in recent stacks and is therefore not recommended.

This approach yields the cleanest architecture: eCAL becomes a first-class dependency inside an OMNeT++ module, meaning no bridges are required, no IPC—enabling direct function calls and shared message definitions. It also unlocks the richest evaluation pathway, allowing full use of INET/Veins for MAC/PHY/channel models, mobility, and protocol behavior while interacting with live eCAL logic. The trade-off is engineering effort and risk: rebuilding and maintaining the entire dependency chain under MinGW is non-trivial, and success depends on internal IT permitting the required toolchain installation and build workflow.

Commercial version of OMNeT++

OMNeT++ can be connected with Microsoft Visual C++ (MSVC), but it is only supported officially in the commercial version of the framework, which is called OMNEST. While the academic version of OMNeT++ relies on MSYS2/MinGW or Clang as its toolchain in Windows, OMNEST offers full MSVC support and even pre-compiled libraries for recent versions of Visual Studio such that it is possible to develop and run simulation models within the MSVC environment completely. This creates a gap where MSVC compatibility is not present in the free academic version but is accessible through the licensed commercially available version.

On this project, building OMNEST with MSVC would have aligned the toolchain with eCAL, which itself had been built with MSVC. This alignment would have eliminated the ABI incompatibility that had prevented direct integration, so that eCAL could be linked directly as part of an OMNeT++ module without needing intermediate bridges or IPC. Nonetheless, while the solution here satisfies the compiler and ABI issue, it does not touch on the IT policy restrictions against TCP/UDP in "network mode," which would have continued to limit the networking options available. Additionally, for the deployment of OMNEST, there would have been a need to confirm that all subsequent dependencies (such as INET, Veins, and Protobuf) were successfully compiled with MSVC for full binary compatibility.

Chapter 6

Experimental Results

In this chapter, the experimental findings during the course of development of this thesis are presented and described. Although the final outcome wasn't as planned when expectations were initially set, whatever work has been done is still of certain utility. Since the project wasn't completed in full, it isn't possible to evaluate a complete end-to-end output or to derive any conclusions from the overall setup. Instead, focus has been placed on this chapter to examine the result of each individual part as it was being made, as well as to investigate the behavior of the system upon putting these parts together. Not only does this highlight the progress made introduced in different project phases but it also helps to understand the constraints and problems that failed to allow the full desired result to be produced.

6.1 Initial Goal

The primary objective of this thesis was to design and implement a simulation framework that integrates Unreal Engine, eCAL, and OMNeT++ to evaluate how V2X communication can contribute to safer autonomous driving in crash and pre-crash scenarios. The framework was envisioned as a tool to recreate cooperative driving scenarios where vehicles are able to exchange information about themselves, their environment, share hazard warnings, and adapt their behavior accordingly. A particular emphasis was placed on limited-visibility conditions, such as blind curves or obstructed intersections, where traditional onboard perception technologies may fail to anticipate hidden dangers.

The importance of addressing such scenarios lies in the fact that they represent some of the most critical risk factors in road traffic. By extending the perception horizon beyond what is physically visible, V2X communication offers the potential to significantly reduce accident likelihood and enhance overall safety. Thus, the proposed framework aimed not only to simulate these challenging environments but also to serve as a foundation for assessing latency, reliability, and overall feasibility of cooperative safety functions.

Although the integration of all components into a fully operational end-to-end framework was not achieved, the work carried out throughout this project provides valuable insights. Each

component—simulation in Unreal Engine, communication via eCAL, and attempts at network-level modeling with OMNeT++ contributed to a deeper understanding of both the opportunities and the challenges of building such a complex system.

6.2 Project Constraints

Despite the progress achieved, the project faced significant challenges that ultimately prevented the completion of the full end-to-end framework. The main limitation was the integration of OMNeT++ with the rest of the system. This step was essential to simulate realistic network conditions, particularly to model communication latencies and packet loss in V2X exchanges. However, the integration process encountered persistent technical obstacles, mainly related to incompatibilities between build environments, compilers, and external dependencies. These issues proved time-consuming and, within the scope of this thesis, could not be fully resolved.

Nevertheless, it is important to emphasize that the other components of the framework were successfully developed and validated. The Unreal Engine environment was set up to simulate vehicles and complex driving scenarios, while eCAL was configured to enable efficient inter-process communication. Together, these elements already provide a functional basis for simulating cooperative perception and safety-related message exchanges, even if network-level realism remains limited.

6.3 Results of Each Component

In this section, the reults achieved for each individual component will be discussed, as well as their integration with each other. Since the complete integration of all modules was not possible, the focus will be on analyzing the contributions of the main building blocks separately.

6.3.1 Unreal Engine 5

In the UE5 environment, the main achievements were the integration of vehicles and the use of splines to define their trajectories, enabling controlled and repeatable motion within the simulated scenarios, which unfortunately has no video demonstration. Additionally, it was verified that importing community-developed asset libraries, which already provide modular elements such as roads, intersections, and urban infrastructures, is straightforward. This means that complex and realistic environments can be built efficiently by simply placing these pre-modeled components where needed.

Figure 6.1 shows how the spline for a vehicle in the Vehicle Template track, which was the one we were building up from, looked like:

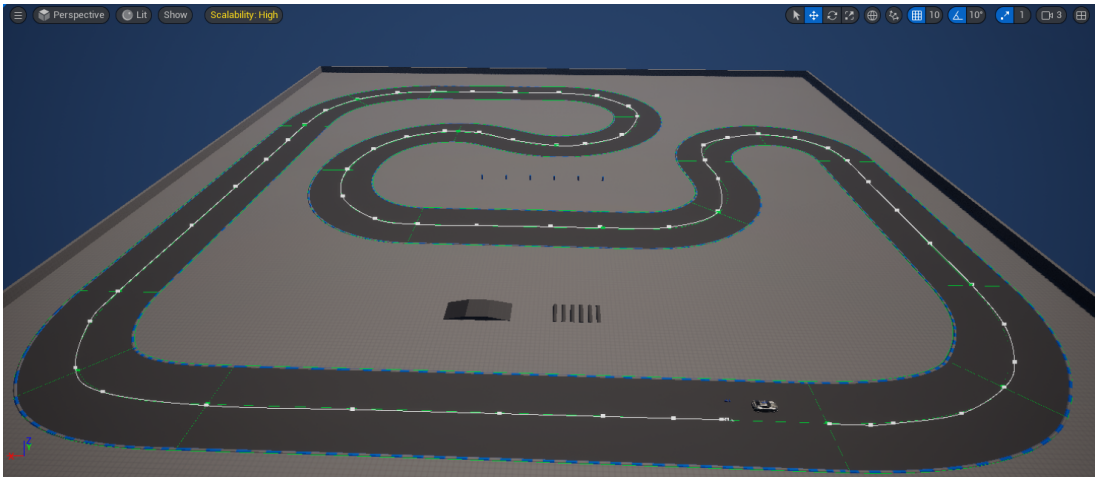


Figure 6.1: Spline path for the template used for the project.

Another key conclusion from this stage is that the level of realism inherently provided by Unreal Engine 5 is considerably higher than that of most commonly available simulators. In particular, when compared with the originally considered Gazebo environment, Unreal offers a much richer visual and physical representation, making it more suitable for the study of scenarios where driver perception and environmental detail play a critical role.

6.3.2 eCAL

The second major component of the framework was the eCAL middleware, which was successfully configured and tested as the backbone for inter-process communication. eCAL allowed for the seamless exchange of data between different modules of the system through a publish–subscribe mechanism, supporting both local shared memory communication and network-based transmission (even if not available for this project).

During the development, the middleware was validated by sending and receiving simulated V2X messages between independent processes, confirming its suitability for handling vehicular communication scenarios. The ease of integration with external applications and the availability of monitoring tools within eCAL further simplified the debugging and verification of message flows.

As stated in 3.2.2, eCAL is capable of establishing communication between nodeseither through shared memory (local mode) or TCP/UDP (network mode). To make the measured latencies more realistic and closer to real-world delays, the preferable option would have been to use TCP/UDP. However, this approach was quickly discarded due to company IT firewall policies that did not allow such communication. As a result, we were restricted to using local mode, which inherently provides much lower delays. This limitation explains why the latency values presented are so low, as shown in Figure 5.6, and should be kept in mind when interpreting the results.

As we were using local mode, which was the best choice for inter-process communication, the latencies still needed to be measured. To achieve this, the messages were timestamped on sending and on receiving. The result is as shown in Figure 6.2.

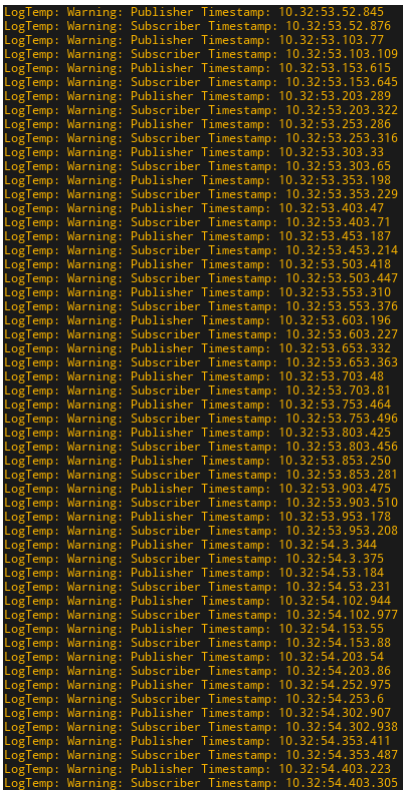


Figure 6.2: Timestamp comparison between publisher and subscriber.

From the messages on the image above it is possible to extract that the minimum latency encountered was of 24 microseconds, the maximum was of 82 microseconds, and the average latency was 36 microseconds. This shows how incredibly fast local mode is and how it can help communication between different modules on a simulation framework.

Although the initial plan was to operate eCAL in network mode (TCP/UDP) to emulate realistic transmission delays, restrictions in the development environment, particularly firewall constraints, made this option impractical. As a result, the evaluation relied primarily on local communication mode, which ensured reliability but lacked the realism of network-induced latencies. This limitation highlights the importance of complementing eCAL with a dedicated network simulator to capture more accurate communication behavior.

6.3.3 OMNeT++

The final component planned for the framework was the integration of OMNeT++ in order to simulate realistic network conditions, such as latency, jitter, and packet loss, which are crucial

to evaluate the reliability of V2X communication. However, this step proved to be the most challenging and ultimately prevented the completion of the full framework.

The integration process faced persistent technical difficulties, primarily caused by ABI mismatches and compiler incompatibilities between OMNeT++ (built with MSYS2/MinGW on Windows) and the precompiled libraries used by eCAL (built with MSVC). These conflicts made it impossible to link the two systems directly, despite multiple attempts that included exploring alternative build configurations, recompiling dependencies, and even investigating the feasibility of compiling OMNeT++ with MSVC, which was deemed to only be possible with the commercial version of OMNeT++, OMNEST.

Although the integration was not achieved within the scope of this thesis, the process yielded valuable insights into the complexity of coupling heterogeneous simulation tools. It highlighted the importance of aligning toolchains and build environments from the outset, and it also provided a clearer view of possible alternative paths, with the best one being the acquisition of OMNEST which would allow eCAL to integrate seamlessly with it.

The findings from this thesis are that the projected simulation framework not only is feasible but also is on the verge of rapid maturity under the right setup and/or licensing agreements. The study sets a distinct practical integration path between high-fidelity scenario and V2X communications simulation, presenting replicable building blocks that others can utilize. In addition, the state-of-the-art review identifies that, despite numerous V2X deployments already being implemented, there remains various untapped potential, in which cooperative perception and pre-crash response can yield notable gains in road safety, rcash safety, traffic efficiency, cooperative mobility, energy sustainability, and more. Through the development of this scalable methodology that generalizes across one use case, this work lays the foundation for systematic experimentation, relative benchmarking across communication stacks, and rapid prototyping of new V2X approaches. In doing so, it provides academia and industry with a realistic, extensible environment in which to try out, refine, and accelerate the next generation of V2X-based safety solutions.

Chapter 7

Conclusions

The work developed throughout this thesis set out to design and implement a framework capable of combining simulation, communication, and network modeling to study the role of V2X communication in road safety. While the complete integration of all components into a fully operational system was not achieved, the project nevertheless provided valuable insights and contributions that extend beyond the original scope.

The most significant challenge encountered was the integration of heterogeneous tools, in particular the coupling of OMNeT++ with Unreal Engine and eCAL. Technical barriers related to compiler incompatibilities and build environments ultimately prevented the realization of the envisioned end-to-end solution. However, the effort invested in this attempt highlighted the importance of toolchain alignment and offered a deeper understanding of the complexity involved in bridging different ecosystems for simulation purposes. These lessons are highly relevant for any future research aiming to achieve similar goals.

Despite this limitation, the project demonstrated that the remaining components could be developed effectively and that the chosen technologies were well-suited to the task. Unreal Engine 5 proved to be a powerful and realistic platform for modeling driving environments and vehicle dynamics, while eCAL established itself as a reliable middleware for modular and scalable communication. Together, these elements form a strong foundation upon which a complete framework can be built.

Beyond the technical outcomes, the thesis also contributed to a broader perspective on the role of V2X communication in limited-visibility scenarios, underlining its potential to extend situational awareness and improve road safety. Even in its partial form, the framework illustrates how simulation can be used to assess latency, reliability, and cooperative safety functions, paving the way for future research to expand these capabilities.

In conclusion, while the original vision of a fully integrated framework was not fully realized, the project succeeded in establishing the building blocks, methodologies, and insights necessary for its eventual completion. The work highlights both the opportunities and the challenges of developing such systems, providing a valuable reference for future studies and reinforcing the importance of V2X communication as a key enabler of safer autonomous driving.

Chapter 8

Future Work

Although this thesis has provided valuable insights into the potential of V2X communication for enhancing crash safety, especially in scenarios with limited visibility, several opportunities for improvement and extension remain as some limitation were found in 5.3. These directions could significantly enrich both the realism of the simulation environment and the applicability of the proposed framework to real-world deployments.

8.1 Expansion of Simulation Scenarios

The current implementation focused primarily on blind-curve scenarios with two vehicles exchanging safety messages. Future work should extend this to more complex traffic situations such as multi-vehicle interactions at intersections, pedestrian crossings, and highway merging scenarios. Including mixed traffic conditions—where some vehicles are V2X-enabled and others are not—would also provide a more realistic evaluation of deployment challenges.

8.2 Integration of Heterogeneous Sensors

While the simulation abstracted certain perception aspects, real autonomous vehicles rely on multiple sensing modalities such as LiDAR, radar, and camera systems. A promising line of research would be to integrate these sensor models in Unreal Engine and study how cooperative perception via V2X can complement, or even compensate for, sensor blind spots and failures. This could also open opportunities to evaluate sensor fusion algorithms in V2X-augmented environments.

8.3 Full Integration of OMNeT++ within the Framework

In the present work, the integration between Unreal Engine 5 and OMNeT++ was only partially explored due to technical and compatibility constraints. As a result, the framework relied primarily on middleware-based message exchange rather than a seamless coupling of the two simulators. Future research should focus on achieving a complete integration of OMNeT++

within the framework, where vehicle dynamics, sensor data, and traffic events in UE5 directly interact with packet-level network simulations in OMNeT++.

This integration would enable highly detailed studies of communication reliability, congestion, and protocol behavior under realistic mobility patterns generated in real time by the Unreal simulation. Furthermore, it would allow researchers to test advanced vehicular networking concepts—such as resource allocation, multi-hop communication, or beamforming in 5G NR-V2X—while observing their direct impact on vehicle safety maneuvers.

Achieving such an integration would likely require addressing ABI mismatches, aligning time models between the simulators, and possibly extending the middleware to serve as a synchronization layer. Once completed, the framework could evolve into a unique end-to-end tool, bridging photorealistic traffic simulation with packet-level network modeling, thus offering a comprehensive environment for testing cooperative intelligent transport systems.

8.4 Real-Time Synchronization Between Simulation Frameworks

One limitation of the current framework lies in the fundamental differences between the simulation engines employed: Unreal Engine 5 operates as a real-time simulator, continuously advancing its physics and rendering at fixed time steps, whereas OMNeT++ and other network simulators are event-driven, progressing in discrete jumps whenever events are scheduled. This mismatch complicates accurate end-to-end latency measurement and coordinated execution across both domains.

A promising direction for future work would be the development of a custom scheduler capable of harmonizing these two paradigms into a unified, real-time framework. Such a scheduler would ensure that network events produced by OMNeT++ (e.g., packet transmission, channel congestion, delays) are injected into the real-time flow of UE5 at precisely the right intervals, while also allowing UE5-generated actions (e.g., vehicle maneuvers, sensor updates) to influence the network simulation without desynchronization.

Achieving this would not only improve the fidelity of latency-sensitive safety evaluations but would also open the door to hardware-in-the-loop experiments, where physical devices require strict real-time guarantees. By aligning event-based network simulation with real-time vehicle dynamics, the framework could evolve into a powerful testbed for cooperative perception and safety-critical V2X applications under realistic timing constraints.

Although this thesis has provided valuable insights into the potential of V2X communication for enhancing crash safety, especially in scenarios with limited visibility, several opportunities for improvement and extension remain as some limitation were found in 5.3. These directions could significantly enrich both the realism of the simulation environment and the applicability of the proposed framework to real-world deployments.

The current implementation focused primarily on blind-curve scenarios with two vehicles exchanging safety messages. Future work should extend this to more complex traffic situations such as multi-vehicle interactions at intersections, pedestrian crossings, and highway merging scenarios. Including mixed traffic conditions—where some vehicles are V2X-enabled and others are not—would also provide a more realistic evaluation of deployment challenges.

While the simulation abstracted certain perception aspects, real autonomous vehicles rely on multiple sensing modalities such as LiDAR, radar, and camera systems. A promising line of research would be to integrate these sensor models in Unreal Engine and study how cooperative perception via V2X can complement, or even compensate for, sensor blind spots and failures. This could also open opportunities to evaluate sensor fusion algorithms in V2X-augmented environments.

In the present work, the integration between Unreal Engine 5 and OMNeT++ was only partially explored due to technical and compatibility constraints. As a result, the framework relied primarily on middleware-based message exchange rather than a seamless coupling of the two simulators. Future research should focus on achieving a complete integration of OMNeT++ within the framework, where vehicle dynamics, sensor data, and traffic events in UE5 directly interact with packet-level network simulations in OMNeT++.

This integration would enable highly detailed studies of communication reliability, congestion, and protocol behavior under realistic mobility patterns generated in real time by the Unreal simulation. Furthermore, it would allow researchers to test advanced vehicular networking concepts—such as resource allocation, multi-hop communication, or beamforming in 5G NR-V2X—while observing their direct impact on vehicle safety maneuvers.

Achieving such an integration would likely require addressing ABI mismatches, aligning time models between the simulators, and possibly extending the middleware to serve as a synchronization layer. Once completed, the framework could evolve into a unique end-to-end tool, bridging photorealistic traffic simulation with packet-level network modeling, thus offering a comprehensive environment for testing cooperative intelligent transport systems.

One limitation of the current framework lies in the fundamental differences between the simulation engines employed: Unreal Engine 5 operates as a real-time simulator, continuously advancing its physics and rendering at fixed time steps, whereas OMNeT++ and other network simulators are event-driven, progressing in discrete jumps whenever events are scheduled. This mismatch complicates accurate end-to-end latency measurement and coordinated execution across both domains.

A promising direction for future work would be the development of a custom scheduler capable of harmonizing these two paradigms into a unified, real-time framework. Such a scheduler would ensure that network events produced by OMNeT++ (e.g., packet transmission, channel congestion, delays) are injected into the real-time flow of UE5 at precisely the right intervals,

while also allowing UE5-generated actions (e.g., vehicle maneuvers, sensor updates) to influence the network simulation without desynchronization.

Achieving this would not only improve the fidelity of latency-sensitive safety evaluations but would also open the door to hardware-in-the-loop experiments, where physical devices require strict real-time guarantees. By aligning event-based network simulation with real-time vehicle dynamics, the framework could evolve into a powerful testbed for cooperative perception and safety-critical V2X applications under realistic timing constraints.

Bibliography

[1] Shimaa A. Abdel Hakeem, Anar A. Hady, and HyungWon Kim. "5G-V2X: standardization, architecture, use cases, network-slicing, and edge-computing". In: *Wireless Networks* 26.8 (Nov. 2020), pp. 6015–6041. issn: 1572-8196. doi: 10.1007/s11276-020-02419-8. url: https://doi.org/10.1007/s11276-020-02419-8.

[2] Dinh-Thuan Do et al. "Two-Way Transmission for Low-Latency and High-Reliability 5G Cellular V2X Communications". In: *Sensors* 20.2 (2020). issn: 1424-8220. doi: 10.3390/s20020386. url: https://www.mdpi.com/1424-8220/20/2/386.

[3] Mao Shan et al. *A Novel Probabilistic V2X Data Fusion Framework for Cooperative Perception*. 2022. arXiv: 2203.16964 [cs.RO]. url: https://arxiv.org/abs/2203.16964.

[4] H. Alzu'bi et al. *C-V2X LiDAR-Based Non-Line of Sight Object Detection and Localization for Valet Parking Applications*. Tech. rep. 2024-01-2040. SAE Technical Paper, 2024. doi: 10.4271/2024-01-2040. url: https://doi.org/10.4271/2024-01-2040.

[5] Jian Wang et al. "A Survey of Vehicle to Everything (V2X) Testing". In: *Sensors* 19 (Jan. 2019), p. 334. doi: 10.3390/s19020334.

[6] Automotive World. *Next step on the road to accident-free driving: Next-generation intelligent networking – Mercedes-Benz brings Car-to-X technology to the roads*. June 2013. url: https://www.automotiveworld.com/news-releases/next-step-on-the-road-to-accident-free-driving-next-generation-intelligent-networking-mercedes-benz-brings-car-to-x-technology-to-the-roads/.

[7] Toyota. *Toyota Bringing Advanced ITS Technology to Mass-market Models*. Sept. 2015. url: https://global.toyota/en/detail/9676551.

[8] Audi Media Center. *Audi networks with traffic lights in the USA*. July 2016. url: https://www.audi-mediacenter.com/en/press-releases/audi-networks-with-traffic-lights-in-the-usa-7147.

[9] Vehicle Dynamics International. *A safety first: Volvo Cars is launching connected safety technology*. Apr. 2019. url: https://www.vehicledynamicsinternational.com/news/industry-news/a-safety-first-volvo-cars-is-launching-connected-safety-technology.html.

[10] Sam Abuelsamid. *New Cadillac Infotainment System Brings V2V Communications Along For The Ride*. Forbes. Feb. 2017. url: www.forbes.com/sites/samabuelsamid/2017/02/22/new-cadillac-infotainment-system-brings-v2v-communications-along-for-the-ride/.

[11] Renault Group. *Renault works with SCOOP to prepare infrastructure for tomorrow's autonomous, connected cars*. Renault Media. Dec. 2017. url: https://media.renault.

com/renault-works-with-scoop-to-prepare-infrastructure-for-tomorrows-autonomous-connected-cars.

[12] BMW Group. *BMW Group enhances road safety by sharing anonymised traffic data.* June 2019. url: https://www.press.bmwgroup.com/global/article/detail/T0296690EN/bmw-group-enhances-road-safety-by-sharing-anonymised-traffic-data?language=en.

[13] Huanyu Gu. *NXP, Volkswagen and Partners Continue to Accelerate the V2X Rollout*. NXP Semiconductors. July 2021. url: https://www.nxp.com/company/about-nxp/smarter-world-blog/BL-THE-V2X-ROLLOUT.

[14] Green Car Congress. *Buick debuts V2X technology in China; first V2X vehicle to roll out by year end*. Nov. 2020. url: https://www.greencarcongress.com/2020/11/20201124-buickv2xchina.html.

[15] Sam Abuelsamid. *Ford Breaks With Auto Rivals By Committing To C-V2X Vehicle Communications Tech*. Forbes. Jan. 2019. url: www.forbes.com/sites/samabuelsamid/2019/01/07/ford-becomes-first-automaker-to-commit-production-c-v2x-communications.

[16] Group Mercedes-Benz. *Europe-wide cooperation project.* June 2019. url: https://group.mercedes-benz.com/innovation/case/connectivity/europe-wide-cooperation-car-to-x.html.

[17] PR Newswire. *Safely Aware: Industry-leading V2X Activation Equips 1.8 Million Stellantis Vehicles With Emergency Vehicle Alert System*. May 2023. url: www.prnewswire.com/news-releases/safely-aware-industry-leading-v2x-activation-equips-1-8-million-stellantis-vehicles-with-emergency-vehicle-alert-system-301832503.html.

[18] Volvo Cars Global Newsroom. *Safely Aware: Industry-leading V2X Activation Equips 1.8 Million Stellantis Vehicles With Emergency Vehicle Alert System*. Feb. 2024. url: https://www.media.volvocars.com/global/en-gb/media/pressreleases/324696/volvo-cars-industry-first-connected-safety-technology-can-now-alert-drivers-of-accidents-ahead.

[19] C-ITS. *French automakers have sold their first connected vehicles!* July 2024. url: https://c-its.developpement-durable.gouv.fr/en/french-automakers-have-sold-their-first-connected-vehicles.

[20] Umar Shakir. *FCC passes auto safety spectrum rules - Cellular Vehicle-to-Everying technology will operate in the 5.9 GHz band.* The Verge. Nov. 2024. url: https://www.theverge.com/2024/11/21/24302733/fcc-cv2x-cellular-vehicle-everything-spectrum-rules-final.

[21] C. Jung et al. "V2X-Communication-Aided Autonomous Driving: System Design and Experimental Validation". In: *Sensors* 20.10 (May 2020), p. 2903. doi: 10.3390/s20102903. url: https://doi.org/10.3390/s20102903.

[22] Tao Huang et al. *V2X Cooperative Perception for Autonomous Driving: Recent Advances and Challenges*. 2024. arXiv: 2310.03525 [cs.CV]. url: https://arxiv.org/abs/2310.03525.

[23] Ghayoor Shah et al. “Scalable Cellular V2X Solutions: Large-Scale Deployment Challenges of Connected Vehicle Safety Networks”. In: *Automotive Innovation* 7.3 (Aug. 2024), pp. 373–382. issn: 2522-8765. doi: 10.1007/s42154-023-00277-6. url: https://doi.org/10.1007/s42154-023-00277-6.

[24] Vishal Sharma, Ilsun You, and Nadra Guizani. *Security of 5G-V2X: Technologies, Standardization and Research Directions*. 2019. arXiv: 1905.09555 [cs.NI]. url: https://arxiv.org/abs/1905.09555.

[25] Zhiying Song et al. *Wireless Communication as an Information Sensor for Multi-agent Cooperative Perception: A Survey*. 2025. arXiv: 2505.00747 [cs.OH]. url: https://arxiv.org/abs/2505.00747.

[26] Antoine Caillot et al. “Survey on Cooperative Perception in an Automotive Context”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2022), pp. 14204–14223. doi: 10.1109/TITS.2022.3153815.

[27] Sabur Baidya, Zoheb Shaikh, and Marco Levorato. *FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot*. 2018. arXiv: 1808.04967 [cs.NI]. url: https://arxiv.org/abs/1808.04967.

[28] Stephen Philips. *Follow a Spline and Report Distance Along It Using an Actor Component.* June 2022. url: https://dev.epicgames.com/community/learning/tutorials/ryL5/unreal-engine-follow-a-spline-and-report-distance-along-it-using-an-actor-component.