Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Christian Moryah Contiero Miranda

DISSERTAÇÃO DE MESTRADO

PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS MECATRÔNICOS

Brasília

2025

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Simulating sensor data collection and shared perception for Autonomous Vehicles

Christian Moryah Contiero Miranda

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica da Universidade Brasília como parte dos requisitos necessários para a obtenção do grau de Mestre

Orientador: Prof. Dr. João Paulo Javidi da Costa

Brasília
2025

Miranda, Christian Moryah Contiero.
M769 / Christian Moryah Contiero Miranda; orientador João Paulo Javidi da Costa. -- Brasília, 2025.
72 p.

Dissertação de Mestrado (Programa de Pós-Graduação em Sistemas Mecatrônicos) -- Universidade de Brasília, 2025.

1. Autonomous vehicles. 2. Traffic simulation. 3. 5G Radio Access Network. 4. Intelligent Transportation Systems. I. da Costa, João Paulo Javidi,

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Christian Moryah Contiero Miranda

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica da Universidade Brasília como parte dos requisitos necessários para a obtenção do grau de Mestre

Trabalho aprovado. Brasília, 30 de março de 2025:

Prof. Dr. João Paulo Javidi da Costa,
Dr.-Ing., ENE/UnB, Hamm-Lippstadt University of Applied Sciences
Orientador

Prof. Dr. José Alfredo Ruiz Vargas,
ENE/UnB
Examinador interno

Prof. Dr. Edison Pignaton de Freitas,
Universidade Federal do Rio Grande do Sul, Instituto de Informatica
Examinador externo

Prof. Dr. Giovanni Almeida Santos,
Hamm-Lippstadt University of Applied Sciences
Suplente

Brasília
2025

To my loving family, whose support and belief in me have been my driving force.
To my friends and mentors who shared wisdom and encouragement.

Acknowledgements

I would like to express my gratitude to the University of Brasília for providing me with the opportunity to pursue my master's degree. The academic environment and resources helped me to "learn how to learn" and shaped my academic journey, enabling me to undertake this research.

I am profoundly thankful to my thesis advisor, Professor Dr. João Paulo Javidi da Costa and my colleagues Dr. Antonio Silva, and Dr. Daniel Alves da Silva for their support, guidance, and insights throughout the research process. Their expertise and dedication to their respective fields have been a constant source of inspiration for me. Thank you for the constructive feedback and encouragement, this allowed me to refine my research and enhance the quality of this work.

I am also indebted to my teachers for their contribution to my academic development. Their lectures, discussions, and mentorship were important to my knowledge and understanding of sensors, signal processing, robotics and neural networks.

“If you find that you’re spending almost all your time on theory,
start turning some attention to practical things;
it will improve your theories.
If you find that you’re spending almost all your time on practice,
start turning some attention to theoretical things;
it will improve your practice.”
(Donald Knuth)

Abstract

Intelligent Transportation Systems (ITS) face significant challenges in data acquisition due to the high costs of infrastructure deployment and maintenance. This research explores the potential of autonomous vehicles (AVs) as mobile sensor platforms to create a virtual infrastructure that supplements traditional ITS data collection methods. Through a series of simulation experiments using the CARLA and Simu5G frameworks, this work evaluates the feasibility and impact of collecting and sharing sensor data from autonomous vehicles. Initial experiments demonstrate that adding a dedicated data collection module to vehicles results in minimal system overhead—only 1.1% increased memory usage with direct sensor collection and 2.6% with application performance management tools on reference hardware. Building on these findings, a simulation framework is developed to enable realistic modeling of sensor data collection, transmission over 5G networks, and cooperative perception among vehicles. Network simulations reveal that the bandwidth required for basic V2X communications (61.68 KB/s per vehicle, with low resolution sensor settings) consumes only 0.049% of a 5G antenna's capacity, suggesting that a single antenna could theoretically support over 2000 vehicles simultaneously. The research extends to security considerations by developing simulations of spoofing attacks in three critical V2X scenarios, demonstrating how malicious actors can compromise vehicle safety through false data injection, and some countermeasures that can be done to prevent these kind of attacks.

Keywords: Autonomous vehicles. Traffic simulation. 5G Radio Access Network. Intelligent Transportation Systems.

List of Figures

List of Tables

Contents

1 Introduction

The recent advancements in autonomous vehicles and Intelligent Transportation System technologies offer a great potential to improve the landscape of logistics and transportation, resulting in a safer and more efficient traffic environment. This change is propelled by a convergence of technological innovations and can generate great progress in the field.

ITS have significantly improved transportation quality by using applications capable of monitoring, managing, and improving the transportation system. However, the large number of devices required to provide data to ITS applications has become a challenge in recent years, particularly the high installation and maintenance costs made broad deployment impracticable. Despite several advances in smart city research and the internet of things, research on ITS is still in the early stages. For example, data integration and reuse from different sources are not yet supported by the underlying infrastructure that provides data to ITS applications.

Assuming a future where a share of vehicles are equipped with advanced sensors, computing systems, and artificial intelligence algorithms, these vehicles carry a significant number of sensors and have the potential to create a shared data infrastructure that can help improve traffic monitoring, performance, and safety.

Given this scenario, the following research questions will be evaluated: Is it possible to collect the Connected Vehicles data with minimal impact on the vehicle's embedded system? What is the feasibility to emulate a generic "plug and play" module that would work in any vehicle of this type? At scale, is it possible to use this data for improving vehicle sensing capabilities and traffic monitoring ?

In this sense, in order to experiment with data collection and AV shared perception techniques, this work contributes with: a proof of concept simulation environment for shared perception based on V2X Technologies, and a set of experiments using the developed POC for AV data collection and exchange.

1.1 Motivation

In modern road traffic systems, inefficiencies often cause congestion and traffic jams, particularly during peak periods. The concentration of vehicles in specific locations and times intensify these problems, resulting in longer travel times and frustration for commuters. Additionally, delayed detection of local problems, such as broken traffic lights, vehicle collisions, or pavement damage, further exacerbates traffic disruptions. These incidents can escalate, causing significant inconvenience and safety hazards for road users if not addressed

promptly.

On the other hand, autonomous vehicles offer a unique opportunity to utilize the vast amounts of data generated during their operation. Although much of this data is usually discarded after immediate use by the Simultaneous Localization and Mapping (SLAM) algorithm and control modules, it has substantial potential for reuse.

One possible approach is to analyze traffic behavior in real-time, which can provide valuable insights into traffic patterns, congestion dynamics, and driver behaviors. By utilizing this data, transportation authorities can improve route planning algorithms, resulting in more efficient traffic management and smoother navigation for commuters.

Furthermore, the wealth of data generated by autonomous vehicles presents a valuable resource for training purposes. This data can be used to create large datasets for machine learning algorithms, which can aid in the development and validation of advanced autonomous driving systems.

Finally, the availability of open data from autonomous vehicles has big potential for urban reuse. This data can provide insights into urban mobility patterns, infrastructure usage, and environmental impacts. City planners, researchers, and developers can benefit from this data.

1.2 Scientific contribution

In addition to this work, the author has contributed to the fields of computer science and intelligent transportation systems, resulting in the following publications:

1. L. Weigang, L. Martins, N. Ferreira, C. Miranda, L. Althoff, W. Pessoa, M. Farias, R. Jacobi, M. Rincon, "Heuristic Once Learning for Image & Text Duality Information Processing," 2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles, Haikou, China, 2022, pp. 1353-1359, doi: 10.1109/ SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse 56740.2022.00195,

2. C. Miranda, A. Silva, J. da Costa G. A. Santos, E. P. de Freitas, A. Vinel, "A virtual infrastructure model based on data reuse to support intelligent transportation system applications" in IEEE Access, vol. 13, pp. 40607-40620, 2025, doi: 10.1109/ACCESS.2025.3547160.

3. D. Alves da Silva, A. Silva, D. Limas, J. da Costa, L de Melo, C. Miranda, G. Santos, A. Vinel, P. Mendes, S. Verhoeven, J. Voigt-Antons, E. de Freitas, "Spoofer detection framework for V2X systems via tensor-based DoA estimation and Yolo-based object detection", forthcoming in IEEE Access.

1.3 Outline

The rest of this work is structured in the following way: Chapter 2 presents the Theoretical Background for the experiments technology stack, Chapter 3 presents the Literature Review and related works. Chapter 4 will discuss about the proper research questions and the research methodology, presenting the proposed techniques and performance metrics to evaluate experiment behavior. Chapter 5 shows the results and impacts on the network behavior based on each selected architecture. Chapter 6 concludes this work by summarizing the research questions responses and proposing future works.

2 Theoretical Background

This chapter presents the theoretical background for the technologies and ideas discussed in this work. It explores two distinct but connected areas. Firstly, it examines ITS applications and the challenges faced in the field. Secondly, it provides an overview of some sensors, and their role in autonomous vehicles technologies. In addition, the chapter contextualizes the areas of sensor fusion in relation to traffic monitoring and cooperative perception. Finally, the simulation environment used for the research experiments is presented.

2.1 Intelligent Transportation Systems

Intelligent Transportation Systems (ITS) are a collection of applications that utilize information and communication technologies to optimize various aspects of transportation and traffic management (CHEN; CHIANG; YANG, 2022). The goal is to promote safer, more efficient, and coordinated usage of transport networks, with a primary focus on road transport.

This includes infrastructure, vehicles, and users, as well as traffic and mobility management systems. These systems include a wide range of technologies, from basic navigation aids and traffic signal controls to advanced applications that integrate real-time data from various sources, such as weather updates, parking guidance systems, and predictive modeling tools.

Emerging technologies, such as smart cities and the Internet of Things, have driven the emergence of Intelligent Transportation Systems (QURESHI; ABDULLAH, 2013). Large cities are investing in the necessary data infrastructure to support this new paradigm. However, the placement of embedded devices in the infrastructure presents challenges in terms of deployment and maintenance costs (GUERRERO-IBÁÑEZ; ZEADALLY; CONTRERAS-CASTILLO, 2018).

ITS applications have used various data sources, such as smart cards, Radio Frequency Identification (RFID) tags, sensors, video cameras, Global Positioning Systems (GPS), social media and high-definition (HD) maps (BHATIA et al., 2022). In this context, Autonomous Vehicles (AVs) are seen as an attractive technology in which data generated from sensors can be reused for the input of sophisticated algorithms that feed ITS applications.

Figure 1 shows an example of an application scenario. A busy street (1) contains several sensors on traffic lights, RSUs and other systems integrated to the road infrastructure. A less busy street can be observed in 2. On this street, it is possible to take advantage of the vehicles sensors to complement existing information that existing infrastructure sensors

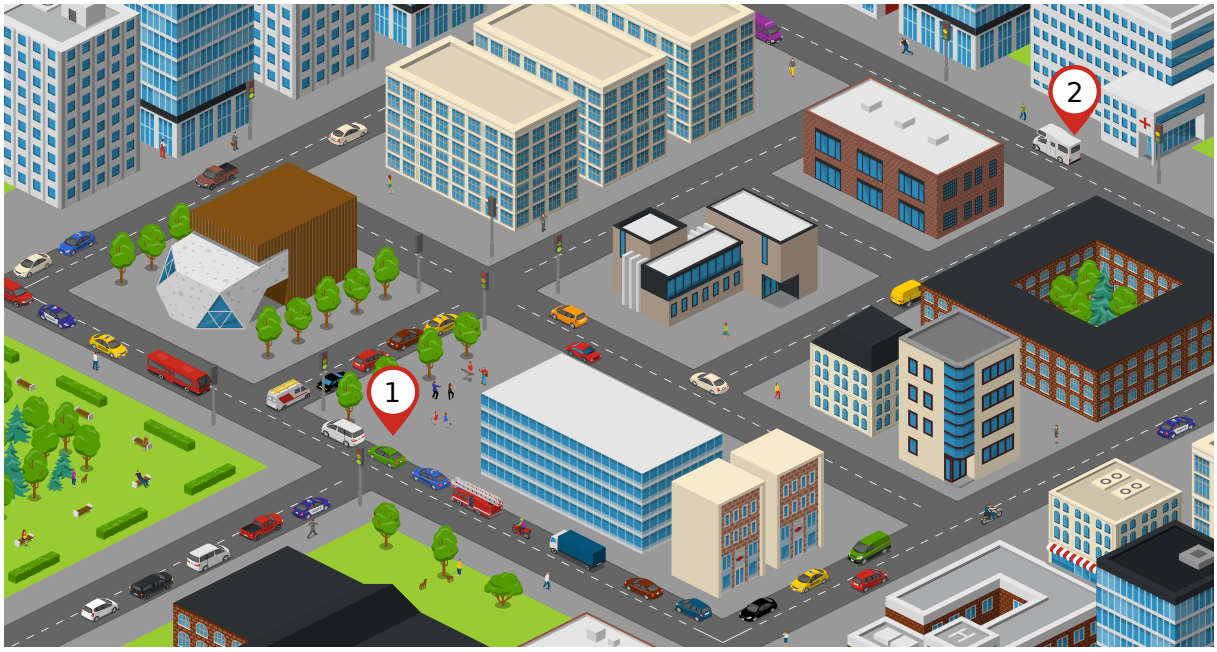

Figure 1 – 1 represents a busy street with traffic lights and cameras, 2 is a less busy street that contains no road sensing infrastructure.

can't provide. Also, with shared data on the other roads, a driver or an autonomous vehicle can make smart detour decisions to avoid traffic jams.

With a data service that could provide real time data of only position and speed, ITS technology can offer, among other things: traffic monitoring and forecasting even in locations where there are no stationary traffic sensing units.

2.2 Autonomous vehicles and sensors

According to the Society of Automotive Engineers (SAE International) J3016 standard, there are six levels of driving automation, from "no automation" to "full automation" and they are summarized in table 1.

Level	Name	Steering and Acceleration/Deceleration	Monitoring of Environment	Dynamic Driving Task
0	No Automation	Human driver	Human driver	Human driver
1	Driver Assistance	Human driver and System	Human driver	human driver
2	Partial Automation	System	Human driver	Human driver
3	Conditional Automation	System	System	Human driver
4	High Automation	System	System	System
5	Full Automation	System	System	System

Table 1 – J3016 levels of driving automation

The more automated the system in table 1, the more reliant it becomes to sensor data. State-of-the-art AVs utilize a wide range of on-board sensors. High sensor redundancy

is required for robustness and reliability in most of the vehicle tasks (YURTSEVER et al., 2020). Electric vehicles have and increasing number of sensors, as reference data points, the BYD Atto 3 has 5 cameras and the Tesla Model 3 has 9.

Sensor data provides critical information about the environment, the driving area, and potential obstacles in the vehicle path. The data collected by these sensors forms the basis for the autonomous vehicle's decision-making and control systems (LI, J. et al., 2023b).

The different modalities of sensors address different aspects of environmental perception. For example, cameras are adept at sensing color without emitting signals for measurement, making them valuable for tasks such as traffic light detection. However, they are sensitive to changes in illumination and have limitations in obtaining depth information.

Lidar and radar sensors, on the other hand, excel at measuring depth and distance to objects, providing critical 3D information that complements the data captured by cameras. Ultrasonic sensors, with their ability to detect obstacles and measure distance using sound waves, further enhance the sensor suite of autonomous vehicles.

2.3 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a set of methods and algorithms that find vehicle location in a space reference frame while simultaneously mapping the given surrounding space (ZHENG et al., 2023). This is critical to navigate and operate in unknown or dynamic environments, specially if existing maps or external localization systems are unavailable or connection is unstable.

SLAM combines sensor data, such as visual, inertial, and depth information. With this data it is possible to estimate the vehicle's pose and location and construct an updated map of the environment in real time. Using the self-generated map and continuous localization estimates, robotic systems can plan and execute trajectories, avoid obstacles, and interact with their environment in a robust and autonomous manner.

SLAM is critical for cooperative perception, as any shared perception message will contain location information of the vehicle and the perceived surrounding. If the vehicle location is imprecise, this will impact the localization of any detected objects.

Considering that Localization is the task to position the car relative to a reference frame in the environment, by having this shared reference frame, its possible to combine data from several different vehicles and apply that to a shared localization and mapping application.

When looking to State of the art SLAM techniques from the sensor point of view, they can be split in 3 types: visual SLAM, lidar SLAM, and multi-sensor SLAM. A recent SOTA comparison in (GARIGIPATI; STROKINA; GHABCHELOO, 2022) found that overall,

Lidar-based methods generally outperformed visual SLAM in outdoor settings and with dynamic objects, benefiting from their broader field of view.

2.4 Sensors

The main types of sensor devices used for perception in the context of autonomous vehicles are described in the following subsections.

2.4.1 Radar

Radar technology, short for Radio Detection and Ranging, is a detection system that uses radio waves to determine the range, angle, or velocity of objects. It was initially developed for military purposes prior to World War II. Radar systems can detect ships, missiles, vehicles, aircraft, and other objects by sending out pulses of high-frequency electromagnetic waves and observing the echoes that are returned. One of the advantages of this sensor is that velocity and position can be measured simultaneously based on the Doppler Effect.

In present days radar technology found a big number of applications, relevant examples for this work are: air traffic control, weather forecasting, traffic monitoring and autonomous vehicles. When used on autonomous vehicles, radar provides real-time data for obstacle detection, speed measurement, and collision avoidance. It can operate in various light conditions and can complement other sensors such as LiDAR and cameras to enhance vehicle perception systems.

Current standard frequency for Autonomous Vehicles by the European Telecommunications Standards Institute is between 77 and 81 GHz (ETSI TR 103 593 and (RAMASUBRAMANIAN; RAMAIAH, 2018)). These frequencies are fast enough to provide the reaction time required by autonomous vehicles and are accurate up to a distance of 250 meters.

Notable radar limitations include susceptibility to jamming and electromagnetic interference, multiple Reflections and Clutter. Clutter refers to unwanted echoes or signals detected by radar systems from objects that are not the target of interest. These may include reflections from the ground, buildings, trees, and other static objects, as well as atmospheric phenomena such as rain and snow.

For comprehensive awareness, autonomous vehicles typically have six to ten radar sensors placed around them (with an exception in Tesla vehicles). In 2022, Waymo's self-driving taxi had six radar sensors, while Mercedes-Benz used eight radar sensors in their test vehicles.

2.4.2 LiDAR

LiDAR is an acronym for Light Detection and Ranging, it is a remote sensing method that uses light in the form of a pulsed laser to measure the distance between objects. The system determines the distance to an object by emitting light pulses and measuring the time it takes for the reflected pulses to return.

This technology was initially developed in the 1960s, shortly after the invention of the laser. It was first used in military research (for satellite tracking) and meteorological tasks, like measuring clouds and pollution.

LiDAR has been applied across diverse fields over the years, besides autonomous vehicles, it has a big number of applications. Relevant recent usages are astronomy (Mars topology analysis and Navigation Doppler Lidar), Archaeology (Upano Valley sites) and Military.

The current state of technology employed in autonomous vehicles allows for a range of up to 400 meters. The sensor outputs the position and intensity of light reflected at a given point, allowing for the measurement of both distance and reflectance values. This information can be used to create a 3D point cloud of the vehicle's surroundings.

LiDAR is a significant component alongside cameras and radars and is favored for its precision in range measurement, which is crucial for detecting other vehicles, pedestrians, and various entities to ensure more safe traffic (LI, Y.; IBANEZ-GUZMAN, 2020).

In the literature, the biggest limitations of this kind of sensor are high costs, big volume of generated data management and bad performance with adverse surface types or under bad weather condition (KIM; PARK; KIM, 2023).

2.4.3 Cameras

Digital camera technology is designed to capture optical images using photosensitive sensors to convert light into electronic signals that form digital images. Cameras operate by focusing light through a lens onto a sensor array that records the light intensity and color of the 3D space into a projected 2D image.

An image sensor array typically consists of a CCD (Charge-Coupled Device) or CMOS (Complementary Metal-Oxide-Semiconductor) image sensor. These two types of image sensors are widely used in camera technology. CCD sensors are preferred for applications where image clarity and detail are the priority, such as in astrophotography and high-end digital photography, due to their high-quality images and better light sensitivity.

On the other hand, CMOS sensors have a construction that enables independent reading of each pixel. This feature contributes to faster processing speeds and lower power consumption compared to CCDs. Therefore, CMOS sensors are particularly attractive for

use in devices where battery life and rapid image capture are critical, such as mobile phones and some consumer-grade cameras.

The history of camera technology dates back to the early 1600s with the formal invention of the camera obscura (earlier experiments of the actual technology usage dates back to as far as 500 AD). Since then, significant progress has been made, particularly in digital imaging during the latter part of the 20th century. Initially used for artistic and documentary purposes, cameras quickly found applications in various fields, including science, security, entertainment, and more recently, in autonomous vehicle systems.

Compared to other sensors, cameras have very distinct features. They work for long and short range detection and have large field of vision and angular resolution. The cost can be high, in case advanced camera sensors with high resolution are being used, but can also be very low if simpler settings are used, especially when compared with LiDAR sensors.

Despite being critical components, cameras come with several limitations that can impact their performance and reliability, a few examples are: sensitivity to environmental conditions, like fog, heavy rain and night; limitations in perceiving depth; difficulties with direct sunlight and high lighting variations.

In autonomous vehicles, digital images provide visual information about the vehicle surroundings that other sensors cannot capture, like information from traffic lights or traffic signs. In recent years Tesla moved to a new program called Tesla Vision, in which it replaced radar and ultrasonic sensors by camera sensing, believing that the future in AV sensing is in image[1].

As reference, we observe that new electric vehicles come equipped with an increasing number of cameras, the BYD Atto 3 has 5 cameras and the Tesla Model Y currently has 9[2].

2.4.4 GNSS

GNSS (Global Navigation Satellite System), commonly referred to as GPS (Global Positioning System), is a satellite-based navigation system that provides geolocation information to a receiver anywhere on the planet. GPS refers to the North American Global Positioning System, while GNSS refers to the International Multi-Constellation Satellite System. This means that it has access to other satellite systems, such as GLONASS, Baidu, Galileo, and others, in addition to GPS.

Initially developed by the United States Department of Defense in the 1970s, the technology was intended for military navigation. Since then, it has expanded vastly in its applications across various sectors.

1 https://www.tesla.com/support/transitioning-tesla-vision
2 https://www.tesla.com/ownersmanual/modely/en_us/GUID-682FF4A7-D083-4C95-925A-5EE3752F4865.html

The sensor system operates by using a network of satellites that transmit radio signals to receivers on the Earth's surface. These signals provide data on the satellites' location and the exact time the signals were transmitted. The GNSS receiver calculates its own position by measuring the time delay between receiving the signals from at least four satellites. This time delay helps determine the distance from each satellite. By determining the distance to multiple satellites, the receiver can triangulate its position in three dimensions.

Recent advancements have improved the accuracy of GNSS technology in civilian use to within a few meters. In autonomous vehicles, GNSS provides critical positioning data that supports other sensors, such as cameras and LiDAR, to create a comprehensive understanding of the vehicle's environment.

GNSS systems face challenges such as signal disruption or blockage in closed or dense environments, as well as vulnerability to signal interference and spoofing. While GNSS provides global coverage and has broad applications, these challenges can limit its reliability and performance in critical navigation and timing tasks.

2.4.5 IMU

Inertial Measurement Unit (IMU) sensing consists of a combinations of accelerometers, gyroscopes, and magnetometers that provide data used to track an object's acceleration, rotational motion, and orientation. IMU technology is currently used in a wide range of applications, including navigation systems, consumer electronics, robotics, and also in the field of autonomous vehicles.

IMUs have an important role in autonomous vehicles by providing essential data for dead reckoning and improving the vehicle's ability to determine its position and orientation in situations where GNSS may be unreliable or unavailable, such as tunnels or urban canyons.

The sensor nature enables integration with other systems, such as GNSS, LiDAR, and radar, forming a comprehensive sensory network that significantly reduces the system's vulnerability to the failure of any single sensor.

However, this kind of sensor also have its limitations. Notable challenge are temperature sensitivity and drift errors in accelerometer and gyroscope readings, which can accumulate over time and lead to increasing inaccuracies if not regularly corrected or calibrated.

Not all autonomous vehicles use IMUs. For example, Waymo does not employ this sensor on their autonomous vehicles. Instead, they use a combination of LiDAR, radar, and cameras.

2.5 Cooperative perception

In an abstract view, cooperative perception can be seen as the collaborative exchange and integration of data among system components to enhance their overall situational awareness beyond individual capabilities.

The concept has derived from research in vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. The shared perception approach allows robots and autonomous vehicles to collectively perceive and interpret environment data, significantly increasing the accuracy and reliability of the information derived when compared to isolated systems which rely solely on onboard sensors (YU et al., 2022).

Although cooperative perception technology has promising benefits, it also faces significant limitations. High bandwidth and latency requirements are necessary due to the real-time sharing of data among multiple entities. Furthermore, challenges such as data privacy, security, and integration of heterogeneous systems are ongoing fields of research. The effectiveness of the technology is limited in areas with poor network coverage or compromised network infrastructure due to its dependence on consistent connectivity.

The current SOTA methods in cooperative perception technology seek to address these limitations by optimizing data transmission mechanisms(THANDAVARAYAN et al., 2023) and integrating machine learning algorithms to enhance data processing efficiency and accuracy (XU; XIANG, H., et al., 2022).

Intelligent Transportation Systems (ITS) can integrate these advancements in cooperative perception technology to create more efficient, safer, and more responsive transportation networks. Through ITS, vehicles can communicate with each other and with traffic management systems, resulting in improved traffic flow, reduced congestion, and enhanced safety.

The use of cooperative perception in ITS is relevant because it enables a shared situational awareness. This awareness allows proactive traffic management and incident response strategies not only on the personal level for the vehicle but for the road infrastructure as a whole.

2.6 Sensor fusion for ITS applications

In the same sense as cooperative perception, but at a lower level, sensor fusion is a method where data from various sources is combined to enhance individual measurements.

Current state of the art research on multi-sensor fusion systems for Autonomous Vehicles have a prevalence of three kinds of combinations in the literature: camera-LiDAR(CL), camera-radar(CR) and camera-LiDAR-radar(CLR).

Data fusion approaches can be categorized based on the abstraction level at which sensor data is integrated. According to (YEONG et al., 2021), when thinking about multi sensor data fusion, there are three primary data fusion classifications used in modern sensor systems: High-Level Fusion(HLF), Mid-Level Fusion(MLF), and Low-Level Fusion(LLF).

LLF is the most granular approach, also known as data level fusion, it consists of combining raw data from all sensors directly. This method is ideal for applications requiring high precision and extensive data detail, as it preserves all of the original sensor information. While it increases the potential accuracy and robustness of sensor interpretations, it also significantly increases the complexity of data processing and requires sophisticated calibration and alignment of sensors. It also generates a large volume of data that can be difficult to manage and process effectively.

MLF, also referred to as feature-level fusion, integrates data by extracting features from individual sensor data before combining them. This method balances between integrating raw data and processed results, providing a moderate level of detail and processing complexity. It allows for better context retention than HLF by using raw data's features rather than completely processed data. However, MLF requires large training sets for effective feature selection and can be challenging to calibrate accurately.

HLF works by first processing data from individual sensors, each of which independently performs detection or tracking algorithms. These results are then combined into a consolidated object definition. The main advantage of this approach is its simplicity, which requires less computation (since it is offloaded to the sensor systems) and less detailed understanding of the underlying sensor technologies. However, HLF can result in the loss of potentially valuable information, as only the processed results from each sensor are merged, discarding any raw data that may contain useful, subtle details.

The nature of sensor fusion presents complex challenges. With the large volume of data from diverse sensors, synchronization becomes imperative to ensure reasonable performance. Given that different sensors emit data at irregular intervals, they are subject to processing delays and transmission latency. If these factors are not properly addressed, they could result in miscommunication or error.

ITS represent a significant area where sensor fusion technology is extensively applied. Sensor fusion for ITS can combine multiple data sources like cameras, radars, and GNSS, to create a more comprehensive understanding of traffic scenarios.

A relevant example is in the context of ITS is Google Maps and Waze, which use data from a large number of users, along with input from mobile phone sensors (which are not necessarily the state of the art in sensing capabilities) and historical traffic data, to predict traffic levels and suggest optimal routes to users. These platforms are simple examples of ITS in action, where sensor fusion can facilitate real-time traffic monitoring and management.

2.7 Simulation environment

Considering the recent developments in autonomous vehicle technology, researchers need realistic simulations for training and evaluation of AI models and automotive operating systems. The security of V2X systems is also a growing concern, specially in the face of cyber attacks. For these reasons, simulation environments have and important role in analyzing the behavior of communication in V2X systems, and enabling research on topics such as jamming (DA SILVA et al., 2023) and spoofing (MÜLLER et al., n.d.).

Based on this premises, the CARLA simulator and Simu5G were selected for research experimentation. The CARLA simulator is used for physics and environment simulation and the Simu5G is used for network simulations. Both frameworks are open source and extensible, but none was designed with built-in features to integrate with one another. In this sense, a "simulation communication framework" was developed for this work.

Recent research (CISLAGHI et al., 2023) have successfully integrated CARLA and Simu5G by using Message Queues and a controller in each of the services. A similar approach was used and will be further explored in the chapters ahead.

2.7.1 CARLA Simulator

The CARLA simulator, standing for Car Learning to Act (MALIK; KHAN; EL-SAYED, 2022), is a platform designed for autonomous driving research and development. It can create controlled and customizable environments for testing autonomous vehicle technologies. CARLA also supports a wide range of scenarios, including weather variations and different traffic conditions. Figure 2 shows an example of the rendered simulation.

Figure 2 – A screenshot of a simulation in CARLA. Carla is based on the Unreal Engine 4

The core motivation for using CARLA is the requirement for risk-free, reproducible, and flexible testing platforms which traditional physical testing methods cannot provide. Its

ability to integrate with major machine learning frameworks (as well as ROS robot operating system) and substantial community involvement[3] makes it a reasonable research tool.

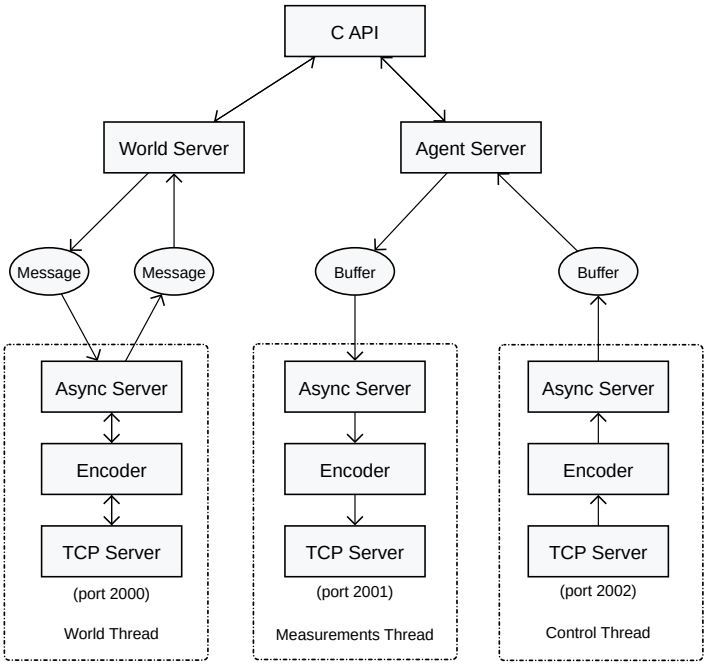

Figure 3 – CARLA architecture, from official documentation https://carla.readthedocs.io/en/stable/carla_server .

Figure 3 depicts the architecture of the CARLA simulator. It is primarily based on a server-client model, with the simulation running as the server and the external control managed by separate client instances. This server can be set up to accept multiple client connections, enabling collaborative testing environments.

The server can be accessed via TCP and has 3 interfaces, the world port, a control port, and the measurements port, each has a server behind which provide a different service.

The "World" server is responsible for controlling the overall simulation environment, while the Agent server has two threads: one dedicated to handling real-time metrics streaming(measurement port) and the other for interactive controls between clients and simulation(control port). This architecture ensures asynchronicity and control even with a considerable number of clients interacting with the server at the same time.

In CARLA, Actors are elements that perform actions within the simulations. These vary from vehicles to pedestrians, each created, managed, and removed through CARLA's simulated world object, that is accessible within its client API. The essence of the actors lies in their detailed blueprints, which are templates defining attributes like vehicle color,

[3] https://github.com/carla-simulator/carla

lidar channels, or pedestrian speed. These blueprints are crucial for adding new actors to the simulation as they specify configuration preconditions and spawn behaviors for the Actors.

Finally, at the CARLA environment, sensors serve as the primary medium for data collection and environmental interaction for autonomous systems undergoing testing. A diverse array of sensors is supported, including standard cameras for RGB, depth, and semantic segmentation, GNSS, IMU Lidar and radar. As the sensors simulate real-world device functionality, they facilitate comprehensive testing in a controlled and realistic setting.

2.7.2 OMNET++ and Simu5G

OMNeT++ is an open-source simulation framework designed for network and distributed system simulations. It provides an infrastructure based on components that can be hierarchically nested, allowing researchers to construct the topology of their simulations in a dynamic manner.

The framework supports the implementation of both physical and application-layer network protocols through the use of various modeling techniques, including finite state machines and message passing for communication between components. It contains a graphical interface base on the Eclipse IDE [4] that can assist in the design, execution, deployment, and analysis of simulation results. Figure 4 shows an examples of a running experiment at the graphical interface. Network topology is presented at the center of the interface and the network profile is presented below. Network events are displayed sequentially and can be inspected for details.

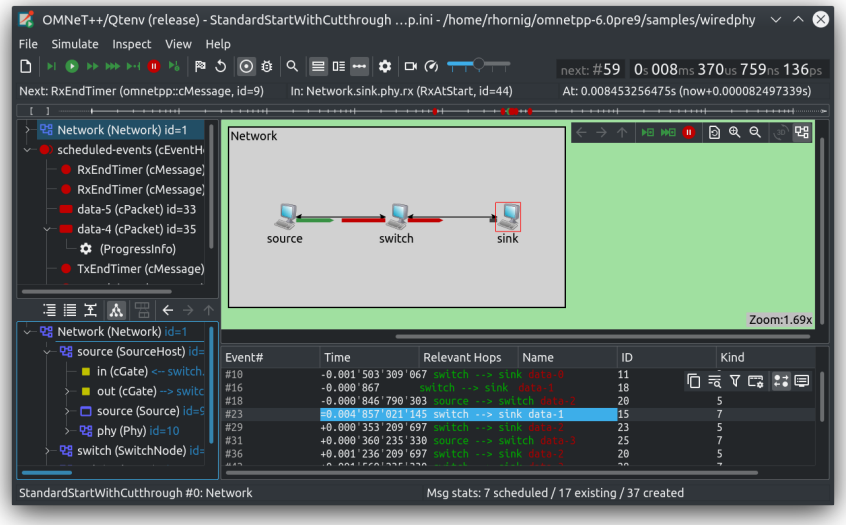

Figure 4 – A screenshot of the OMNET++ graphical interface from release 6. https://omnetpp.org/software/2020/11/03/omnet-6-pre9-released.html

[4] https://eclipseide.org/

Simu5G, on the other hand, is a library that extends the capabilities of the OMNeT++ simulator to model end-to-end behavior of 5G networks (NARDINI et al., 2020), including low latency, and high data rate communications. The library incorporates features such as dual connectivity, service-based architecture, and a wide variety of deployment scenarios from dense urban to rural settings, all within the scenario-based modeling provided by the OMNeT++ framework.

It specifically simulates the data plane of the 5G RAN (Release 16) and the core network. The library allows for simulation in both Frequency Division Duplexing (FDD) and Time Division Duplexing (TDD) modes across heterogeneous gNBs, ranging from macro to pico cell sizes, and it includes features such as handover support and inter-cell interference coordination via the X2 interface.

Dual connectivity with LTE (eNB) and 5G NR (gNB) base stations is also supported, alongside 3GPP-compliant protocol layers. Realistic, customizable channel models are used for the physical layer, and the system supports resource scheduling in both uplink and downlink directions. Simu5G also supports numerous mobility models for user equipments, including those designed for vehicular mobility contexts.

3 Literature Review

This chapter presents a summary of current state of the art research and works related to our proposed contributions with a focus on ITS and Cooperative Perception research.

3.1 Intelligent Transportation Systems

Within ITS applications, two big areas of research have have come to prominence. First, the current and future travel time prediction studies are helpful for route planning and traffic management. Second, there is a big number of studies on the construction of intelligent infrastructures, including smart cities, roadside devices, and IoT to improve communication performance and security. Other notable research is generally orthogonal to these two branches is also included to provide an overview of the field.

For travel forecasting (CHEN; CHIANG; YANG, 2022) and (BHATIA et al., 2022), evaluate AI algorithms and mathematical progressions to predict travel times. (HAYDARI; YILMAZ, 2020) reviews extensive literature on deep learning technologies for travel time prediction. For most of these works, real-time data is crucial for an accurate answer. In line with this, (LIU et al., 2018) and (WANG, Y. et al., 2021) propose models or approaches for data collection ranging from installing devices in vehicles to using mobile sources for data generation and training of AI algorithms.

An extensive review composed of 586 articles is done in (KAFFASH; NGUYEN; ZHU, J., 2021). The article discusses big data algorithms and their applications in ITS, where the increasing amount of data requires advanced, data-driven approaches. The application of big data algorithms in ITS includes areas such as traffic flow prediction, travel time and route planning, and improving vehicle and road safety. Authors identify gaps in the field and show that the most used algorithms for ITS are based on deep learning methods.

In contrast to the described above, the second research branch proposes architectures for developing an intelligent infrastructure. The research's primary focus is installing embedded devices on the roads. In (ZHU, L. et al., 2018), a cheap embedded device to popularize ITS applications is proposed. Other works, such as (WANG, Y. et al., 2021) and (GUILLEN-PEREZ; CANO, 2021; GARG; CHLI; VOGIATZIS, 2018; LIANG et al., 2019), propose using data from existing sources, such as the use of IoT systems, images from security cameras, and others.

To further explore what can be achieved by a combination of modern technologies to improve ITS, (DAI et al., 2019) investigates the applications of AI, edge computing, and caching technologies to improve the efficiency of ITS systems. RSUs are used to offload the

computing closer to the network edge. The AI decision-making process provides a more efficient resource allocation and optimizes the network routing plan. With the inclusion of edge offloading and caching, latency and communication time are drastically reduced.

In (TANWAR et al., 2019) authors experiment with a non-orthogonal Multiple Access network to meet current industry and standards and government regulation. Authors model the end-to-end latency under the designed architecture and analyze the reliability concerning the interruption probability of the network. Statistical analysis is used to model different latency components including transmission, processing, and propagation latency. Results show that its possible, with a combination of wifi and 5g, to keep latency under 5ms.

A survey on the latest advancements in Blockchain for IoV is done in (MOLLAH et al., 2021). Authors evaluate the integration of of these technologies to address ITS communication challenges. The potential benefits of this combination are enhanced data security, increased reliability, and system transparency. While recognizing the promise of the convergence of blockchain and IoV in enriching the intelligence and security of transportation systems, it also highlights the existence of several practical adoption challenges that need to be overcome. These include issues related to scalability, energy consumption, privacy concerns, and lack of regulation.

In (BAO; WANG, Q.; JIANG, 2021) authors define digital twin in transportation and its possible applications. Similarities and differences between traffic simulation and digital twin are discussed. The article explores modeling vehicle driving behavior and environment simulation with the Digital twin. A three layers architecture is proposed, including data access layer, calculation and simulation layer, management and application layer.

Looking at the new 5G communication standard, (GOHAR; NENCIONI, 2021) discusses the role it has in enhancing smart city functions, with a particular focus on ITS. It explains how 5G technologies are critical to realizing the concept of smart cities by providing high-speed connectivity, low latency, and the ability to handle a large number of connections. Particularly for ITS, the article discusses how 5G will enhance the interconnectivity of different transportation modes, improve traffic management, and increase overall transportation efficiency.

To further improve AV and ITS capabilities, (ELGHAZALY et al., 2023) explores the use of High Definition maps. A review of the state of the art of HD maps uses in the various functions of autonomous driving systems is presented. Authors discuss the AV components relying on Map Data, like SLAM, scene understanding, motion prediction and planning modules and how HD maps can greatly improve AV capabilities. Furthermore, HD maps and GNSS data can be fused, offering robust and precise localization services to AVs.

Considering the above articles, an important highlight is that at this moment there is no research on building virtual infrastructures that feed ITS and traffic management systems

with AV data in real-time.

3.2 Cooperative Perception

In (CUI et al., 2022) the authors summarize multi-sensor fusion methods, communication technology, and shared perception strategies. The impact of communication cost and robustness of vehicle positioning errors is analyzed. The authors argue that to achieve autonomy levels above 3, it is necessary to leverage advanced sensing technology, edge computing, communication, and other technologies to build a cooperative perception system. This work also reinforces the idea that the future lies in V2X applications, supported by 5G communication technology.

Another recent review took place in (XIANG, C. et al., 2023), the work summarizes the applications of multi-sensor fusion classification strategies in cooperative perception. It proposes a multi-sensor fusion taxonomy for autonomous driving perception and classifies fusion strategies into symmetric and asymmetric fusion with seven subcategories.

(XU; XIANG, H., et al., 2022) examines the potential of V2X communication to enhance the perception performance of autonomous vehicles. The authors propose a novel vision Transformer architecture, called V2X-ViT, which combines data from multiple vehicles' Lidar sensors and achieves state-of-the-art performance results compared to similar techniques that employ intermediate fusion. The article considers both an ideal communication channel and a noisy setting, where pose error and time delay are both considered. No network-specific experiments were developed.

In their work, the authors of (LI, J. et al., 2023a) address the issue of intermediate fusion in order to account for the possibility of Lossy Communication channels on V2V communications. The LC proposal is designed to accommodate real-world scenarios in which there are Doppler shifts introduced by fast-moving vehicles, interference generated by other communication networks, and dynamic topologies caused by routing failures as well as various weather conditions.

An additional work that attempts to address lossy communication scenarios is presented in (REN et al., 2024). Instead of implementing intermediate fusion, the authors develop a prediction model to extract multi-scale spatial-temporal features based on V2X communication conditions, thereby capturing the most significant information for the prediction of the missing information.

In (THANDAVARAYAN et al., 2023), authors propose baseline mobility-based generation rules for cooperative perception messages with mechanisms to control the redundancy of the information and reduce channel overhead. The proposed techniques improve perception, reduce channel load, and enhance scalability for all cooperative perception communications.

To Address offline datasets for AI model training, (XU; XIA, et al., 2023) creates a dataset specifically for V2V cooperative perception research. Three cooperative perception tasks are introduced with benchmarks for model evaluation. According to the authors, biggest challenges in cooperative detection include GPS error, asynchronicity, and bandwidth limitation.

In (YIN et al., 2023) a multimodal transformer model is introduced for cooperative perception. This new model employs lidar and camera fusion from different vehicles to achieve SOTA cooperative perception performance. The transformer integrates point-based and voxel-based features into a single 3D representation.

To address the some inherent challenges in cooperative perception such as the loss of semantic information, perception errors, (SONG et al., 2024) proposes improving vehicle pose calibration. An object association approach named context-based matching is used, which calibrates multi agents pose to improve shared perception precision. Object association precision is achieved with decimeter-level relative pose calibration accuracy.

To further facilitate the development of research in the field, (CARLETTI et al., 2024) presents a new co simulation tool that allows physical and network simulations (integrates CARLA, OpenCDA, and ns-3). The framework facilitates analysis of vehicular networks under various V2X technologies and enables evaluation of AIML-enabled autonomous driving applications leveraging realistic sensor data.

In the topic of safety (NOH et al., 2024) proposes a method for expanding the field of view for autonomous vehicles by using edge infrastructure sensors (e.g. RSU systems). The "Infrastructure cooperative autonomous driving" system demonstrated an improvement in safety speed of 17% and a reduction in collisions in simulated scenarios.

Considering the large volume of sensor data generated recently by both autonomous vehicles and edge devices, (HAKIM et al., 2024) proposes the use of a Spatial and Temporal Clustering algorithm which not only reduces the communication payload between vehicles by clustering perceived objects across AVs but also optimizes the use of communication resources. The work presents significant enhancement in the efficiency of vehicle-to-vehicle communication networks, increasing information reception by 10% and reducing communication payload by approximately 41% compared to previous ETSI standards.

in (CISLAGHI et al., 2023), authors have successfully integrated CARLA and Simu5G to simulate vehicle teleoperation. Vehicle sensor data was sent to a remote controller that would transmit steering and acceleration commands to the vehicle.

Based on the above literature review, evidence suggests that ITS and Cooperative perception areas share a significant number of articles with parallel applications. For instance, in both fields there are articles that investigate the usage of algorithms and optimization of the communication channel for scalability purposes.

4 Methodology

In order to answer the research questions presented in the introduction, two sequential works are presented in this chapter. Firstly, the feasibility and impact of additional data collection from Autonomous vehicles is evaluated in a simulated environment. After the first experiments facilitate data collection, a traffic monitoring solution is experimented with.

After the the data collection experimentation, a research framework is developed based on related works to experiment with shared perception in V2V and V2X scenarios.

4.1 AV data collection

In this section we evaluate the impacts of a data acquisition model on AV concerning local processing, memory and storage.

The experiment was built on top of the CARLA simulator. CARLA contains a client-server architecture where the server runs the simulation computation and rendering. The client is responsible for information collection and interaction in the simulation world through "Actors", as previously presented in subsection 2.7.1. In this setup, two computers were used, one for the simulation server and one for the client.

To establish mechanisms for persistent and pervasive data collection for the presented environment, two scenarios were considered: (i) collect data from sensor actors, which are managed by the simulation server, and offer callback utility functions directly to the CARLA client, and (ii) collect data directly from CARLA's underlying data layer, using application performance management (APM) tools.

Both computers shared the same local network. Communication between client and server was established through CARLA's Client library, leveraging its TCP socket setup across default ports: 2000, 2001, and 2002. This setup enables asynchronous encoding of messages using Protocol Buffers (Protobuf), to enhance efficiency in data transmission and networking operations.

The source code is available on GitHub[7]. The Carla Client operates on a personal laptop with the following specifications: Ubuntu 22.04 Operational System, 6.5.0-26-generic kernel, Intel i7-13650HX with 14 cores and 2.6 GHz base frequency, 32Gb RAM DDR4, 1 TB M2 SSD storage. The vehicle agents used in the simulation are available at the CARLA challenge[8] and also possess github repositories.

[7] https://github.com/av-data-research-group/carla-data-collection
[8] https://leaderboard.carla.org/leaderboard/

In (ETSI, 2019), 3GPP published a standard for a cooperative perception Service (CPS) and cooperative perception Message (CPM), the CPM describes all kinds of sensor data and confidence levels to be shared between vehicles. Message size will scale up with the amount of sensors data attached and fusion or processing level of the present data.

To simplify the collection process and measurements in the present experiments, saved data only includes sensor output. CPM standard data format is not used as added metadata would not present a big influence in data volume. CPM will be considered for future work exploring communication experiments.

The document also cites a Frequency and Content Management entity, which determines the optimal CPM generation period and number of perceived objects and regions to be included in the CPM. That has an important impact in respect to the channel usage. Since in this stage of experimentation we are using raw sensor data and no perceived objects, we arbitrarily defined a period of 3 seconds of frequency for sensor data collection.

In order to precisely monitor Memory and CPU consumption, the python client process was monitored by its current PID. At the tests done with the APM server software, its process was also monitored and memory and CPU usage were summed. The Ubuntu System Monitor and a combination of the following shell commands were used: `top -p PID`, `cat /proc/PID/statm`, `ps faux | grep PID`, `pmap -x PID` and `docker stats`. CPU usage will be expressed in percentage relative to the system CPU.

Figure 5 – A camera capture of the simulation. This camera was attached to the top of the car, and placed facing "front" relative to it.

Figure 5 shows an example of a collected image from a vehicle camera in the simulation. Camera pose didn't reproduce its positioning in autonomous vehicles in the real world, as that is not relevant for measuring the system impact. An arbitrary positioning was used to help simulation monitoring, for example, to check if the vehicle was working as expected, didn't collide, among other problems that could happen during the experiment.

Table 2 presents memory and CPU consumption data for a single vehicle with 3

sensors that was created and integrated into the existing traffic manager of the simulation environment. In this simulated environment, the Elastic APM python agent creates several objects for monitoring and introspection, the memory spent is more than double the memory collected directly from the sensors.

Usage	Memory	CPU	Memory difference
without any data collection	29.0 MB	0.44%	-
with client data collection	38.9 MB	0.55%	9.9 MB
with profiling data collection	105.2 MB	0.67%	76.2 MB

Table 2 – Total memory and CPU comparison between collection methods. The simulation server is not accounted for.

Increase in CPU usage for the profiling solution was smaller than anticipated, our main hypothesis for this is that we are only monitoring a small number of objects (3 sensors) and the load is more memory and I/O based (for logging reasons). Total storage used after 1 hour of navigation was 441 MB, accounting for the images of the 2 cameras and 1 Lidar sensor.

New electric vehicles come equipped with an increasing number of cameras, the BYD Atto 3 has 5 cameras and the Tesla Model 3 currently has 9. To further evaluate the system load in respect to these kind of sensors only, we evaluated CPU and RAM usage against an increasing number of cameras. The cameras used a default setup of 960x480 resolution and 120 degrees field of view. Results are presented in Table 3.

# of cameras	Memory	CPU
1	34.1 MB	0.57%
2	38.4 MB	0.82%
3	41.5 MB	1.04%
4	44.3 MB	1.19%
5	46.4 MB	1.42%
6	48.0 MB	1.58%
7	49.4 MB	1.76%
8	51.2 MB	1.95%
9	52.7 MB	2.13%

Table 3 – Total memory and CPU usage for an increasing number of cameras, both increase linearly with the number of cameras.

To experiment with SOTA self driving models, Table 4 presents impact on memory and CPU from collecting data for a Transfuser (CHITTA et al., 2022) agent, the winner of one of the CARLA Challenges event. Sensor setup was reproduced from the authors github repository[9] containing 3 cameras (with higher resolution than previous baseline experiment), one lidar sensor, an IMU, a GNSS and a speedometer. A total of 7 sensors. As reference, for the Transfuser model training, each camera has, in addition to RGB, depth maps and semantic segmentation data.

[9] https://github.com/autonomousvision/transfuser/

Usage	Memory	CPU	Memory difference
without extra collection	29.4 MB	0.4%	-
with sensor data collection	45.8 MB	0.91%	16.4 MB
with profiling data collection	109.2 MB	1.09%	79.8 MB

Table 4 – Total memory and CPU comparison between data collection methods for the Transfuser agent. Total sensor count in the CARLA simulator: 7. Simulation server is not accounted for in used memory.

The same baseline model was used for navigation (not the Transfuser model), so the usage without the data collection in practice is the same. We can observe that the memory spent in the sensor data collection method increased around 20%, this can be attributed to the growth of sensors on the vehicle. On the other hand, the memory from the APM data collection tools kept being much more significant, even though it didn't scale with the number of sensors like it did in the first experiment. For this experiment, the total storage used after 1 hour of data collection was 3.1 GB, This big increase in used memory is attributed to the increase in amount of sensors, number of cameras and camera resolution. Lidar setup didn't change between experiments.

To better understand the impact this system would have on a vehicle in the real world, we considered the Delphi Audi zFAS, which is a central driver assistance controller released in 2018, for Audi's A8 model. The controller owns a 4 GB SDRAM, DDR3L-1866, so with the transfuser setup (7 sensors) that translates to 1.1% of total memory usage with the direct sensor collection, and 2.6% with the APM data collection, a neglectable impact on a 6 year old controller. Additionally, the camera test still shows a feasible expenditure with 9 cameras: 52.7MB or 1.32% of the total controller memory.

Autonomous vehicles have security and decision modules that need to access data with minimum latency. For this reason, CPU and memory assets must be rationalized as much as possible, the presented sensor data collection mechanism is able to meet that necessity.

In a nutshell, the costlier performance of the APM solution was expected due to the software evaluation infrastructure built around it. In physical vehicles, on the other hand, the extra load will be less than that presented in this work, considering that instead of a python profiler, a profiler is used at the vehicle operational system level.

The increase in amount of storage needed for the extra module is relevant as we show that for the transfuser model, this would be around 3GB for 1 hour. Camera and lidar data are raw and represent the biggest part of this volume, with 2.8GB dedicated to the 960x480 resolution images from the 3 cameras.

4.2 A traffic monitoring experiment

Considering that starting code is available for data collection, an additional data collection experiment was designed to take advantage of it.

In order to evaluate a simple traffic monitoring application, three "monitor vehicles" were placed in the simulation. For each vehicle, GNSS and speedometer sensor data was collected in the same way it was done in the previous experiment.

At the CARLA simulation, a map called Town10HD_Opt was selected[1]. This map's road network consists of a grid layout, including numerous different junctions and traffic lights.

A light and a heavy traffic scenario were considered, one with only the three vehicles in the simulation, allowing for top speed in the selected map, and one with the monitor vehicles and another 100 cars, to examine the heavy traffic scenario.

Figure 6 – Traffic comparison, on the left the light traffic simulation, and on the right the heavy one. Points represent the vehicle location and the speed during collection step. Right color map is represented in meters per second.

Each vehicle built a log of the collected data, the log was monitored in real time and data was collected for 20 minutes. At the end of the simulation, map data was fused with the GNSS and speed data from the vehicles to generate the proposed traffic monitor.

Figure 6 depicts the experiment end result. Its possible to observe that even in the light traffic scenarios, some points of the road have a lower mean speed. This points are present in junctions or traffic lights.

Another thing that was expected was the low average speed even for the light traffic scenario (8.5 m/s or 30.6 km/h). This happened because the selected map represent a

[1] https://carla.readthedocs.io/en/latest/map_town10/

"downtown" location, consisting of several junctions and traffic lights, that would naturally lower the speed of passing vehicles.

As expected, in the heavy scenario situation, traffic light and junctions are the places with the biggest impact, its possible to observe a large traffic jam on the lower left corner of the map. Considering the simplicity of the generated code, it can be reused to evaluate light and heavy in all other CARLA maps.

If a minimal number of vehicles share this kind of data with a single, centralized server, its possible to have real time speed data with a reasonable error. The error rate increases with the decreasing of number of vehicles, as data gets "old" until another vehicle passes by the location to metrify speed again.

4.3 Data transmission simulation

In order to create the network experiments part of the cooperative perception simulation, we built a new application for communication simulation on top of previous experiments code. These new experiments do not use APM methods for data collection, and rely solely on CARLA's sensor data implementation.

The new addition takes advantage of the Carlanet project (CISLAGHI et al., 2023), in which Omnet and CARLA communication is implemented as a message queue with ZeroMQ.

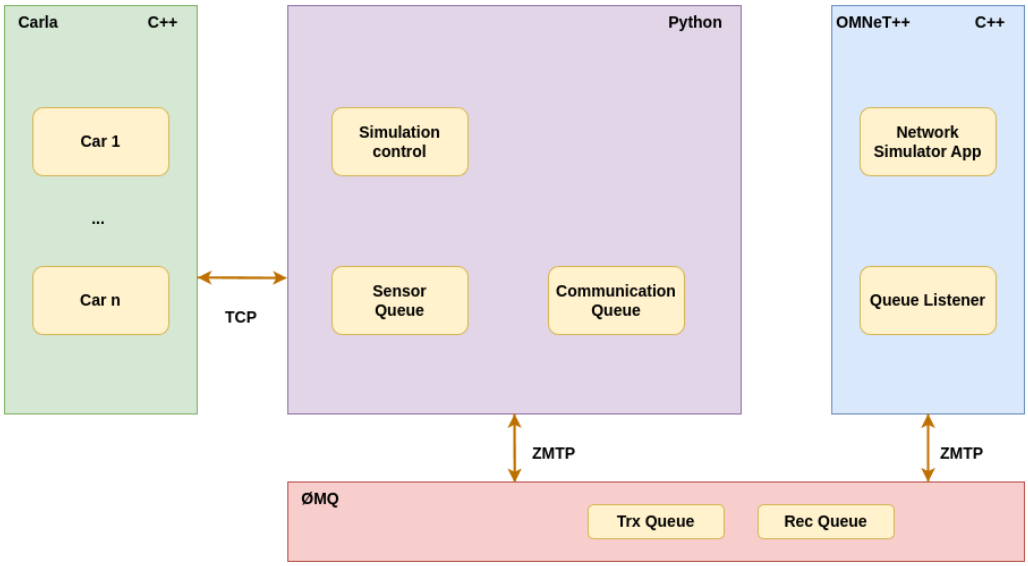

Figure 7 – Block diagram containing the simulation framework components. Carla and experiment communicate via TCP and Omnet++ communication happens through ZeroMQ.

As a significant amount of stack has been added to the experiment code, Figure 7 portraits the new overall setup. Communication with CARLA maintains the TCP standard

with ports 2000, 2001 and 2002. When sensor data is available for transmission, or when the vehicle controller is asked for, sensor data is sent to OMNET++ through a transmission queue for network simulation.

Default format for data transmission in the Carlanet setup is json messages, so when camera image data is transmitted, it has to be base64 encoded and then decoded as utf-8 to text, as bytes are not serializable in json format. An example of the json message is presented below in "Code" 4.1.

Code 4.1 – json message example

```
{
   "message_type":"GENERIC_RESPONSE",
   "simulation_status":0,
   "user_defined":{
      "msg_type":"VEHICLE_DATA",
      "camera_data":"gHx8/4F9ff+BfX3/gX19/.../gX19/",
      "actor_positions":[
          {
             "actor_id":"carlaNodeCar_1",
             "position":[
                325.1853942871094,
                1.9907456636428833,
                0.0019066429231315851
             ],
             "rotation":[
                -0.010655094869434834,
                0.06594514101743698,
                -0.0001220702906721 2716
             ],
             "velocity":[
                6.835569858551025,
                0.008157790638506413,
                8.010864007701457e-07
             ],
             "is_net_active":true
          }
      ]
   }
}
```

To keep both simulations synchronized, a "Simulation step" message is also exchanged between the simulations. At every simulation step, vehicle location and speed are sent to Omnet++. This is needed to correctly execute the 5G network simulations.

With the initial communication setup done, we reproduce the traffic monitoring experiment with location and speed data being transmitted from the vehicles to a 5G antenna placed in the center of CARLA‘s TOWN01, the smallest map available.

The antenna is connected to a Remote Service Agent that sits in the python part of

the code. When a new data message arrives, it performs network simulation (from car to antenna) at Omnet++ and forwards it to the experiment Receiving Queue for the Remote Agent to process. Once in the Receiving Queue, the remote service saves it to the local file system for processing by reusing code from the traffic monitoring experiment.

The following Sequence Diagram in Figure 8 represents how the experiments were performed. An initial configuration phase is done where all the simulators are started and configured. Omnet++ contains the trigger to start the entire experiment, after started, it triggers the experiment configuration both in itself and in carla, where the configured Actors are placed and vehicle control is configured.

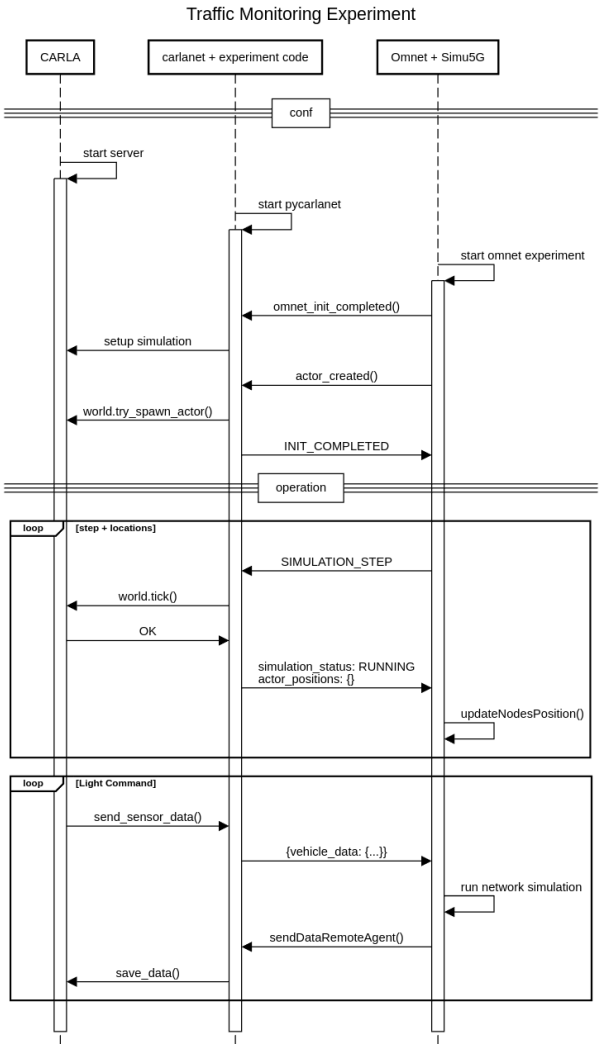

Figure 8 – Sequence diagram representing the data flow in the experiment application. After an initial configuration stage, 2 communication loops are executed: One for simulation synchronization and other for data sharing.

Two communication loops are then initiated, the "Simulation step" and the "Vehicle data" messages start being exchanged by using the ZeroMQ Service.

Traffic monitoring experiment is run for TOWN01, with a remote data service to represent an ITS traffic management system, again with 0 and 100 cars.

Results show the same standard as previous traffic monitoring experiment: average speed around traffic lights drop significantly when the map is full of cars.

Results can be seen at Figure 9. datapoints are sparser even though the same number of monitor cars were used, that is due to the fact that the collection average time for each location and speed was increased from 3 to 9 seconds, in adaptations from the initial code to the Carlanet standard.

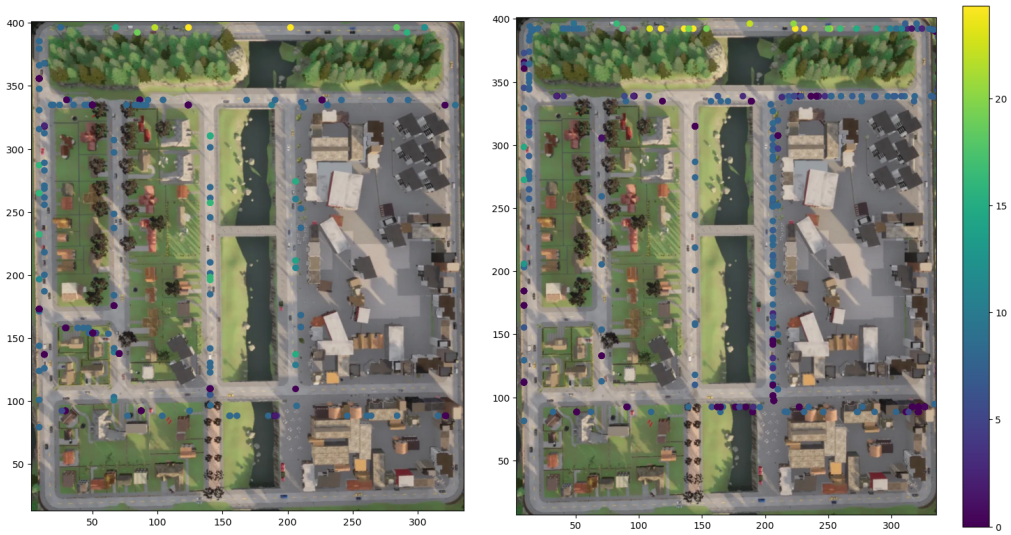

Figure 9 – Traffic comparison, on the left the light traffic simulation, and on the right the heavy one. Points represent the vehicle location and the speed during collection step. Right color map is represented in meters per second.

Average speed was 18 km/h at the "No traffic" simulation and 10.8 km/h at the "High traffic" one. A thing to note is that Town01 is actually bigger than Town10, so an addition of 100 cars didn't have the same impact as it had in the previous traffic monitoring simulation.

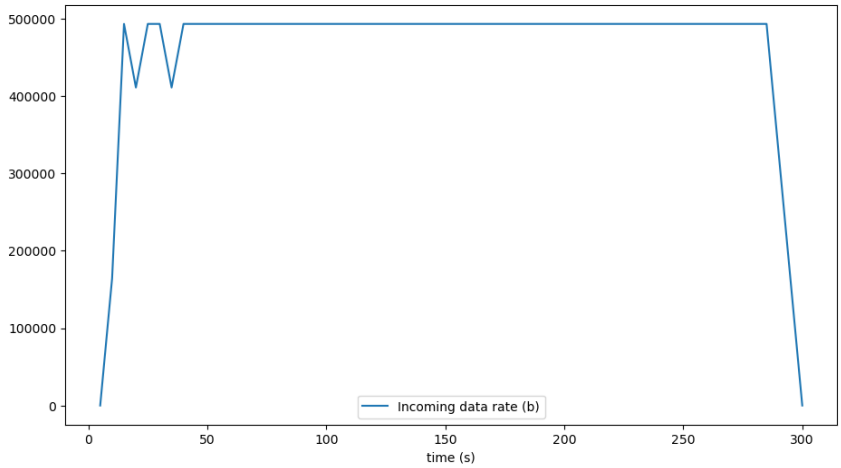

Figure 10 – Bit rate of a single vehicle during simulation time. Two messages are exchanged, Simulation step and traffic monitoring.

In an additional experiment made to measure channel usage, figure 10 shows the Bit

rate of the data transmissions from a single car for around 6 minutes of simulation: A total of 61,68 KB/s considering both Simulation step and traffic monitoring messages.

Our hypothesis for the first fall in the Bit rate after an established connection is occlusion or some sort of signal interference that was not purposely added to the simulation.

Considering that a 5G antenna capacity is 10 Gbps, both messages account for 0.049% the maximum channel usage. If a dedicated antenna is considered, a single one is capable of receiving messages from 2040 vehicles at the same time.

Current B5G/6G research estimate big improvements with bitrates getting up to 100 Gbps, further increasing the feasibility of a virtual data infrastructure for traffic monitoring purposes.

4.4 Cooperative Perception simulation framework for V2X Security experiments

Building upon our previous data collection and transmission experiments, we developed a integration framework between CARLA and Simu5G simulators to enable cooperative perception testing focused on V2X cybersecurity. This integration addresses the critical need for realistic simulation environments that can model both complex traffic scenarios and 5G network communications simultaneously, particularly when evaluating security aspects like jamming and spoofing attacks.

The integration architecture, called B5GCyberTestV2X_CARLANet, is the central connection point between CARLA and Simu5G. This framework enables vehicles within the CARLA environment to share sensor data through simulated 5G connections, creating a bootstrapped environment for cooperative perception algorithms under various security threat scenarios.

Within B5GCyberTestV2X_CARLANet, several key submodules work together to facilitate the operation and programming of use case scenarios. Figure 11 shows the system architecture. The "Data" submodule collects information from sensors incorporated into the CARLA simulator, sending this information simultaneously to the TX submodule and the "Fusion" submodule. Meanwhile, the "Control" submodule manages vehicle actuators within the CARLA environment, ensuring precise control over vehicle behavior based on cooperative perception inputs. Any algorithm can be used with the control submodule, including CARLA's auto pilot feature. This setup creates a closed feedback loop where vehicles can share sensor data, process information from other vehicles, and adjust their behaviors accordingly.

The communication between CARLA and B5GCyberTestV2X_CARLANet continues to utilize TCP with Protocol Buffers (protobuf) for efficient data serialization, operating across

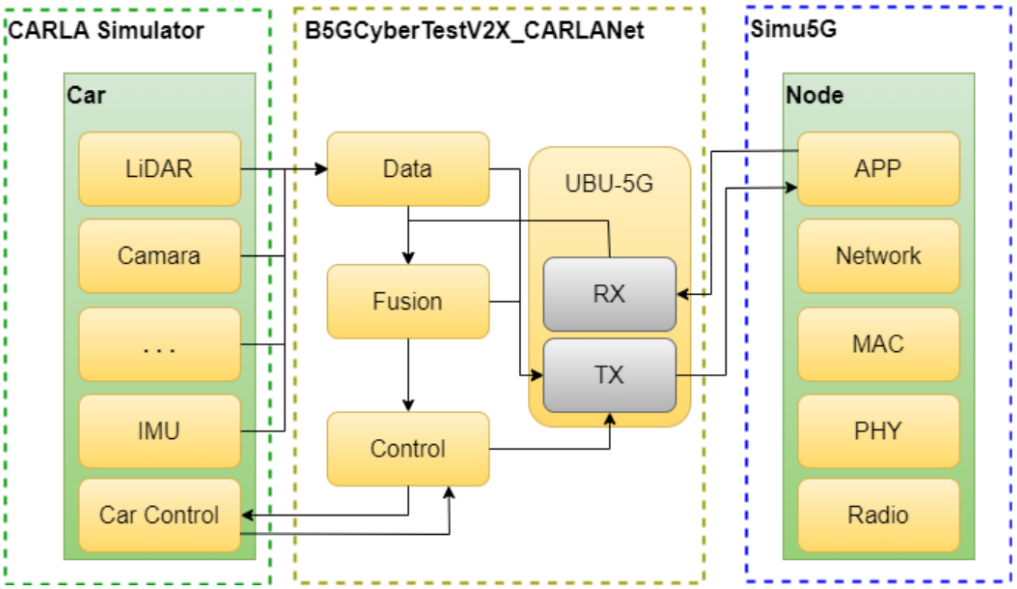

Figure 11 – Block diagram depicting the B5GCyberTestV2X_CARLANet architecture for integration between CARLA and Simu5G simulators. The diagram illustrates the data flow from vehicle sensors (LiDAR, Camera, IMU) through the central integration module to the Simu5G network simulation. Key components include the Data submodule for sensor acquisition, Fusion for data integration, Control for vehicle actuation, and the UBU-5G module containing RX/TX components that facilitate bidirectional communication with the Simu5G network stack.

ports 2000, 2001, and 2002. For the connection between B5GCyberTestV2X_CARLANet and Simu5G, we continue to use the ZeroMQ service with the ZMTP protocol, working on top of the Carlanet project.

Our simulation workflow begins with the B5GCyberTestV2X_CARLANet module initializing both simulators using configurations present in a single configuration file. During operation, an iterative cycle processes sensor data collection from CARLA vehicles, transmission of this data through the simulated 5G network, reception by other vehicles, fusion with local sensor data, and control actions based on the combined information.

For the cooperative perception tests, Figure 12 depicts a use case that was implemented involving two autonomous vehicles communicating via V2V connections. In this scenario, Car 1 collects sensor data and transmits it to Car 2 through the simulated 5G network. Car 2 then fuses this data with its own sensor readings and executes control algorithms based on this combined information.

For monitoring and logging, each component of the system features its own implementation managed by the experiment code. Additionally, we used CARLA's Recorder feature to document events during simulation, allowing for detailed post-analysis and reproducibility of results.

This integration framework represented a significant advancement in simulation for autonomous vehicle research in the cybersecurity domain. By combining traffic simula-

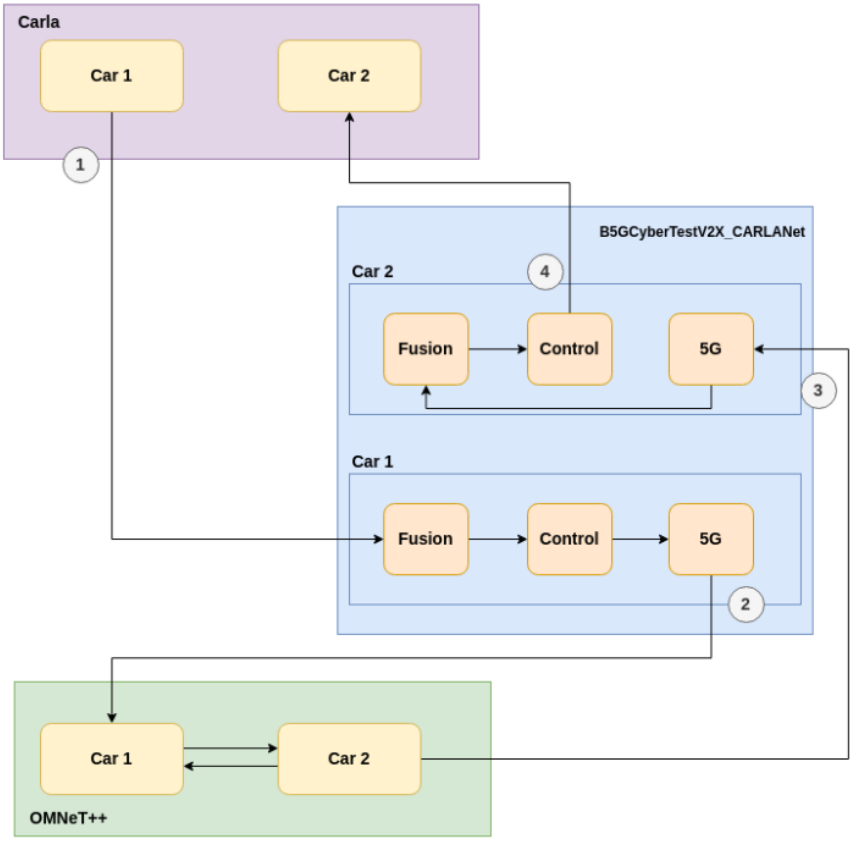

Figure 12 – Block diagram of a shared perception use case in the B5GCyberTestV2X framework. 1. Car 1 in CARLA sends sensor data to the B5GCyberTestV2X_CARLANet module; 2. data is transmitted to the OMNeT++ environment for network simulation; 3. Car 2 receives the processed data through its 5G module; 4. fusion and control components process the combined information to execute appropriate actions in the CARLA simulator.

tion with realistic 5G network modeling, the framework enables the testing of cooperative perception systems under various security scenarios.

4.5 A Simulation Dataset for Vehicular Cybersecurity experiments

Building on top of the developed data sharing framework, a series of standalone experiments were conducted to research vehicular cybersecurity, specifically focusing on Vehicle-to-Everything (V2X) communication security and spoofing attack detection.

This research addressed critical vulnerabilities in V2X systems where malicious actors can inject false information, disrupting traffic flow and compromising autonomous vehicle safety. The study combined hardware-level signal analysis (Direction of Arrival) with AI vision detection methods (YOLO V8) to develop a new method for detecting spoofing attacks and identifying the physical location of malicious senders.

Three autonomous driving scenarios were simulated using the CARLA simulator

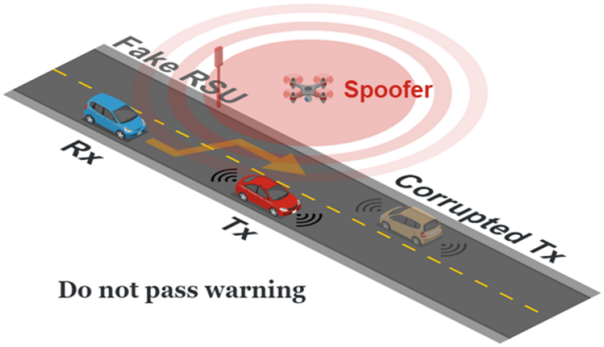

Figure 13 – Do Not Pass Warning illustration depicting a drone-based attack where a malicious actor deploys a counterfeit roadside unit (RSU) to inject corrupted data into the vehicle network. The compromised signals disrupt V2X communications between the transmitting and receiving vehicles, creating unsafe conditions during passing maneuvers.

to generate the datasets. The first scenario, depicted in Figure 13, "Do Not Pass Warning", represents a common highway situation where vehicles need to make overtaking decisions. This scenario involves communication between vehicles to ensure safe passing maneuvers, particularly critical when one vehicle needs to determine if there is sufficient time and space to overtake another vehicle safely.

The second scenario, "Vulnerable Road User Alerts at Blind Intersections", is illustrated in figure 14 addresses the challenge of detecting and responding to pedestrians at intersections with limited visibility. This scenario is particularly relevant it is a common occurrence in urban environments where buildings or other obstacles can obstruct a vehicle's direct line of sight.

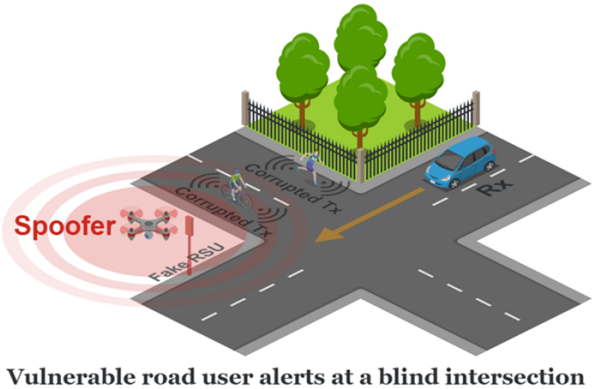

Figure 14 – Vulnerable Road User Alert system being compromised at a blind intersection. A drone-based attack using fake infrastructure signals disrupts critical safety communications between vehicles and pedestrians, creating heightened risk at an already visibility-limited crossing.

The third scenario is called "Left Turn Assist" (available in figure 15) focuses on the complex decision-making process required when vehicles execute left turns at intersections, requiring coordination and communication between multiple vehicles to prevent collisions.

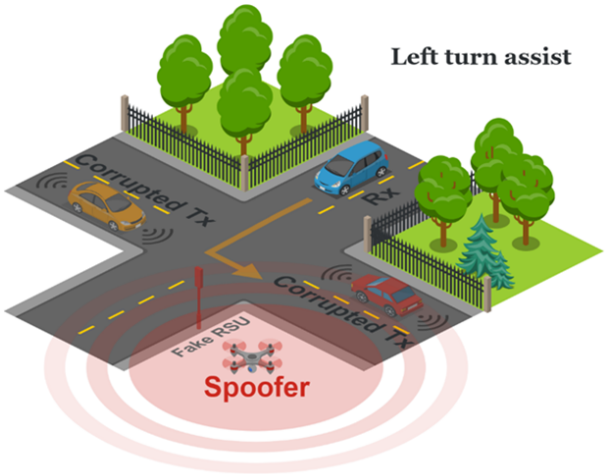

Figure 15 – Left Turn Assist system compromised by a drone attack that uses counterfeit infrastructure signals to interfere with vehicle communications at the intersection. The spoofed data affects vehicles' ability to safely coordinate left turns, creating potential collision risks.

For each of these scenarios, we collected two distinct subsets of data. The first subset captured normal operation data, where V2X communications functioned as intended, resulting in safe navigation and accident prevention. The second subset contained attack scenario data, where communications were compromised through spoofing attacks, leading to potential or actual accidents.

The dataset includes temporal sets of: vehicle position coordinates in three dimensions, vehicle orientation including pitch, yaw, and roll measurements, vehicle velocity vectors, camera captures from vehicle-mounted sensors, pedestrian position data where applicable, drone position and orientation data, and timestamp information for synchronization purposes.

In the Do Not Pass Warning scenario, data was collected from three vehicles and one drone. The normal operation subset demonstrated successful overtaking maneuvers where vehicles maintained safe distances based on accurate position and velocity information. The attack scenario subset revealed how spoofed location data led to unsafe passing decisions, resulting in collisions.

Figure 16 shows the Vulnerable Road User scenario. In this scenario, we gathered data from two vehicles, one pedestrian, and one drone (the cyclist was considered a vehicle to the simulation). Normal operation data showed the system successfully detecting and responding to pedestrians at blind intersections. The attack scenario data documented instances where compromised sensor information resulted in vehicles failing to detect or appropriately respond to pedestrian presence.

For the Left Turn Assist scenario, three vehicles and one drone were monitored. The normal operation subset captured successful left turn maneuvers enabled by proper

Figure 16 – Simulation scenario of Vulnerable Road user use case, the vehicle can be seen about to collide with the pedestrian. The Spoofer drone can be seen in the sky, at the upper left corner of the image.

V2V communication. The attack scenario subset illustrated how manipulated position and velocity data could lead to incorrect turn timing decisions and potential accidents. An example of attack situation can be seen in figure 17.

Figure 17 – A picture illustrating an attack in the Left Turn Assist scenario. The Tesla Cybertruck does a improperly timed left turn and ends up colliding with a police vehicle. The spoofer drone can be seen in the sky, close to the intersection.

The spatial and temporal data collected enabled accurate identification of malicious transmitters' physical locations through Direction of Arrival (DoA) estimation techniques. Furthermore, having both normal operations and attack scenarios helped training and validating our AI-based framework for attacker classification and countermeasure selection.

4.6 Generating data for vision model finetuning

To enable robust drone detection capabilities for identifying potential attackers, a specialized dataset was created for fine-tuning the YOLOv8 object detection model. The complete dataset comprises 4,824 annotated images focused on drone detection across various environments and conditions. This dataset was divided into a training set containing 3,877 images (80% of the total) and a validation set of 947 images (20% of the total).

A significant portion of the dataset was generated using the CARLA simulation environment. The data collection process involved placing stationary drones at strategic locations across different CARLA maps. Due to limitations in the simulation software's drone physics capabilities at the time of data collection, the drones remained in fixed positions throughout the collection process. Given the technical constraints of implementing new vehicle models in CARLA, a single drone model was utilized, the model and its dimensions inside the simulation are available in Figure 18, though its appearance was varied through random color alterations to introduce visual diversity.

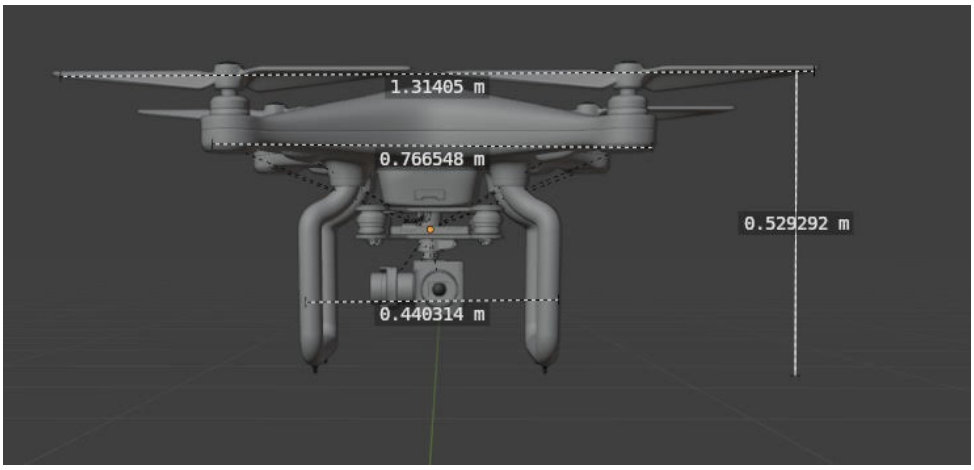

Figure 18 – A picture of the drone model inserted in the simulation

The image collection process in the simulation environment was conducted through automated vehicle routes with mounted cameras. These vehicles would navigate through the city environments, capturing images at approximately 0.3-second intervals. Each collection session lasted several minutes, during which simulation parameters were periodically randomized to create diverse scenarios. These parameters included weather conditions and time of day, ensuring the dataset captured a wide range of lighting conditions and atmospheric effects. One example can be seen in Figure 19.

To supplement the simulated data and enhance the model's generalization capabilities, synthesized images showing drones in various lighting conditions and backgrounds were also incorporated in the dataset, as well as augmented versions of base images.

The annotation process required careful attention to detail and consistency. Each image received manual bounding box annotations indicating drone locations within the frame.

Figure 19 – One of the raw images generated by the simulation for further annotation. A drone can be seen on the right with the sky as the background, while a second drone is a little harder to see is at the far left, with some buildings in the background.

A single-class labeling approach was maintained, focusing solely on drone detection, as this aligned with the specific need to identify potential attack sources. Annotation standards remained consistent across the entire dataset to ensure data quality.

The YOLOv8 model was trained with specific parameters optimized for this use case. The input format followed the NCHW standard (Batch, Channels, Height, Width) with image dimensions of 3×640×640 pixels. A batch size of 32 was used with the Adam optimizer and Cosine Annealing for learning rate scheduling. The initial learning rate was set to 0.005, gradually decreasing to 1.0e-05 throughout the training process. The model trained for 200 epochs to ensure proper convergence.

The performance metrics from the training process demonstrated strong detection capabilities. The model achieved a Mean Average Precision (mAP50) of approximately 0.9, indicating excellent accuracy in drone detection at the standard 50% Intersection over Union (IoU) threshold. The more stringent mAP50-95 metric reached approximately 0.6, showing robust performance across varying IoU thresholds. Both precision and recall metrics stabilized near 0.9, indicating balanced performance in terms of false positives and false negatives.

The fine-tuned model became an important component of the larger spoofer detection framework. The high-accuracy drone detection capabilities were essential for identifying potential attack sources in V2X environments, complementing the signal processing approaches with visual confirmation of threat vectors. The model's reliable performance across various conditions enhanced the framework's overall ability to detect and respond to spoofing attacks in most real-world scenarios (at low lighting the results were not as good).

4.7 Future Works: Security Challenges and Opportunities in V2X Cooperative Perception

While significant progress has been made in establishing the B5GCyberTestV2X framework, training detection models, and simulating attack scenarios, several promising research directions remain unexplored. Building on our current foundation, several advanced research areas can be considered.

In the area of attack detection and mitigation, more sophisticated anomaly detection algorithms could be developed to identify vehicles purposely sharing incorrect location and speed data. This might include implementing real-time trust scoring mechanisms for data sources based on consistency and behavioral patterns. Additionally, adaptive countermeasures could be designed to automatically respond to different types of V2X attacks beyond the drone-based spoofing scenarios already simulated.

For resilient cooperative perception, future work could investigate predictive algorithms that maintain accurate environmental awareness during communication disruptions or jamming attacks. A critical challenge to address would be developing conflict resolution strategies when vehicle sensors and cooperative perception data provide contradictory information, such as when an object is placed in front of a vehicle in map data but not detected by onboard sensors.

Simulation capabilities could be further expanded to better model real-world scenarios. Additional network simulation tools might be integrated to model more complex attack vectors. The implementation of moving drone models with realistic physics would enhance the fidelity of aerial attack simulations. Furthermore, scenarios could be developed for testing system resilience against coordinated multi-vector attacks involving multiple malicious actors, providing insights into system vulnerabilities.

Looking to real-world applications, hardware-in-the-loop testing using the simulation framework connected to actual vehicle systems could be designed. Methodologies for transferring security insights from simulation to real-world V2X systems would be essential for practical implementation.

By exploring these research directions, the security and reliability of cooperative perception systems in autonomous driving could be further improved, making them more resilient against sophisticated cyber attacks and ensuring their robust operation in diverse real-world scenarios.

5 Results and Discussion

On the data collection part of this research, we evaluate the impact of a data aquisition model on AVs regarding local processing, storage and RAM usage (these parameters can also be used to estimate energy consumption). The experiment used the CARLA simulator in a client-server architecture and applied real-time data collection directly from the simulation server, as well as using application performance management tools.

Resulting data show that an extra data collection module at the vehicles offer a negligible 1.1% increase in total memory usage with direct sensor collection and a 2.6% increase with application performance management (APM) data collection on the reference hardware.

The traffic monitoring experiment was designed to test the research question about the feasibility of traffic monitoring by the reuse of AV data for real-time traffic monitoring. Three "monitor vehicles" were placed in the simulation to collect GNSS and speedometer data. Two traffic conditions were simulated: light and heavy traffic. We show that it is feasible but we don't discuss network, identification and security concerns, important parameters in this topic, as that is reserved for future experiments.

The results obtained in the simulation by the proof-of-concept indicate that the proposed architecture could be applied in real world AVs; this form of data reuse can significantly improve ITS performance in its biggest challenges with minimal impact on current technology stack. Moreover, the reported experience with proof-of-concept allowed the identification of other promising research directions.

Our data transmission simulation expanded these concepts by integrating the CARLA simulator with communication network modeling through the B5GCyberTestV2X framework. This integration enabled realistic 5G network simulations for V2X communications, revealing that the bit rate required for basic cooperative perception (approximately 61.68 KB/s per vehicle) consumes only 0.049% of a 5G antenna's capacity. This finding suggests that a single antenna could theoretically handle data from over 2000 vehicles simultaneously, confirming the technical feasibility of large-scale V2X implementations.

The development of the cooperative perception simulation framework for V2X security experiments represents a significant advancement in our research. By enabling vehicles to share sensor data through simulated 5G connections, we created a environment for testing cooperative perception under various security scenarios. The framework's modular architecture with separate components for data collection, fusion, and vehicle control facilitated detailed analysis of how vehicles can effectively share and utilize perception data.

Our cyber security experiments demonstrated the vulnerability of V2X systems to spoofing attacks. Through three detailed attack scenarios (Do Not Pass Warning, Vulnerable Road User Alerts, and Left Turn Assist), we simulated how malicious actors could compromise vehicle safety by injecting false information. The dataset created from these simulations provides valuable ground truth for developing and validating security countermeasures.

The vision model fine-tuning component of our research addressed the critical need for robust drone detection as part of a broader attack mitigation strategy. By creating a specialized dataset of 4,824 annotated images and finetuning a YOLOv8 model, we achieved high detection accuracy (mAP50 of approximately 0.9), demonstrating that visual confirmation can complement signal-based approaches in identifying potential threats.

Right now the amount of sensor data that would be transmitted, reflecting what we measured in usage of storage, is considerable. To tackle this problem, one of the possible research directions is the local pre-processing of the data in the vehicles prior to transmission and analyzing the impacts in relation to computing, like explored in (THANDAVARAYAN et al., 2023). Another involves proposing to improve the reliability of B5G in the presence of intermittent connectivity that can degrade data accuracy.

The B5GCyberTestV2X framework provides a starting platform for future research in V2X security, enabling more sophisticated simulations and the development of advanced defensive strategies. Through continued refinement of these tools and methodologies, the security and reliability of cooperative perception systems in autonomous driving could be significantly enhanced, making them more resilient against sophisticated cyber attacks and ensuring their robust operation in diverse real-world scenarios.

References

BAO, L.; WANG, Q.; JIANG, Y. Review of digital twin for intelligent transportation system. In: IEEE. 2021 International Conference on Information Control, Electrical Engineering and Rail Transit (ICEERT). 2021. P. 309–315. Cit. on p. 30.

BHATIA, V.; JAGLAN, V.; KUMAWAT, S.; SIWACH, V.; SEHRAWAT, H. Intelligent Transportation System Applications: A Traffic Management Perspective. In: INTELLIGENT Sustainable Systems. Springer, 2022. P. 419–433. Cit. on pp. 16, 29.

CARLETTI, C. M. R.; CASETTI, C.; HÄRRI, J.; RISSO, F. ms-van3t-CARLA: an open-source co-simulation framework for cooperative perception evaluation. In: WONS 2024, 19th Wireless On-demand Network systems and Services Conference. 2024. Cit. on p. 32.

CHEN, M.-Y.; CHIANG, H.-S.; YANG, K.-J. Constructing Cooperative Intelligent Transport Systems for Travel Time Prediction With Deep Learning Approaches. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, 2022. Cit. on pp. 16, 29.

CHITTA, K.; PRAKASH, A.; JAEGER, B.; YU, Z.; RENZ, K.; GEIGER, A. **TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving**. 2022. arXiv: 2205.15997 [cs.CV]. Cit. on p. 35.

CISLAGHI, V.; QUADRI, C.; MANCUSO, V.; MARSAN, M. A. Simulation of tele-operated driving over 5g using carla and omnet++. In: IEEE. 2023 IEEE Vehicular Networking Conference (VNC). 2023. P. 81–88. Cit. on pp. 25, 32, 38.

CUI, G.; ZHANG, W.; XIAO, Y.; YAO, L.; FANG, Z. Cooperative Perception Technology of Autonomous Driving in the Internet of Vehicles Environment: A Review. **Sensors**, v. 22, n. 15, 2022. ISSN 1424-8220. DOI: 10.3390/s22155535. Available from: <https://www.mdpi.com/1424-8220/22/15/5535>. Cit. on p. 31.

DA SILVA, A. S.; DA COSTA, J. P. J.; SANTOS, G. A.; MIRI, Z.; FAUZI, M. I.; VINEL, A.; FREITAS, E. P. de; KASTELL, K. Radio Jamming in Vehicle-to-Everything Communication Systems: Threats and Countermeasures. In: IEEE. 2023 23rd International Conference on Transparent Optical Networks (ICTON). 2023. P. 1–4. Cit. on p. 25.

DAI, Y.; XU, D.; MAHARJAN, S.; QIAO, G.; ZHANG, Y. Artificial intelligence empowered edge computing and caching for internet of vehicles. **IEEE Wireless Communications**, IEEE, v. 26, n. 3, p. 12–18, 2019. Cit. on p. 29.

ELGHAZALY, G.; FRANK, R.; HARVEY, S.; SAFKO, S. High-definition maps: Comprehensive survey, challenges and future perspectives. **IEEE Open Journal of Intelligent Transportation Systems**, IEEE, 2023. Cit. on p. 30.

ETSI, I. Intelligent transport system (its); vehicular communications; basic set of applications; analysis of the collective-perception service (cps). **Draft TR 103 562 V0. 0.15**, 2019. Cit. on p. 34.

GARG, D.; CHLI, M.; VOGIATZIS, G. Deep reinforcement learning for autonomous traffic light control. In: IEEE. 2018 3rd ieee international conference on intelligent transportation engineering (icite). 2018. P. 214–218. Cit. on p. 29.

GARIGIPATI, B.; STROKINA, N.; GHABCHELOO, R. **Evaluation and comparison of eight popular Lidar and Visual SLAM algorithms**. Aug. 2022. DOI: 10.48550/arXiv.2208.02063. Cit. on p. 18.

GOHAR, A.; NENCIONI, G. The role of 5G technologies in a smart city: The case for intelligent transportation system. **Sustainability**, MDPI, v. 13, n. 9, p. 5188, 2021. Cit. on p. 30.

GUERRERO-IBÁÑEZ, J.; ZEADALLY, S.; CONTRERAS-CASTILLO, J. Sensor Technologies for Intelligent Transportation Systems. **Sensors**, v. 18, n. 4, 2018. ISSN 1424-8220. DOI: 10.3390/s18041212. Cit. on p. 16.

GUILLEN-PEREZ, A.; CANO, M.-D. Intelligent IoT systems for traffic management: A practical application. **IET Intelligent Transport Systems**, Wiley Online Library, v. 15, n. 2, p. 273–285, 2021. Cit. on p. 29.

HAKIM, B.; ELBERY, A. A.; HEFEIDA, M.; ALASMARY, W. S.; ALMOTAIRI, K. H.; NOURELDIN, A. STC: Spatial and Temporal Clustering for Cooperative Perception System. **IEEE Transactions on Vehicular Technology**, p. 1–11, 2024. DOI: 10.1109/TVT.2024.3376544. Cit. on p. 32.

HAYDARI, A.; YILMAZ, Y. Deep reinforcement learning for intelligent transportation systems: A survey. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 23, n. 1, p. 11–32, 2020. Cit. on p. 29.

KAFFASH, S.; NGUYEN, A. T.; ZHU, J. Big data algorithms and applications in intelligent transportation system: A review and bibliometric analysis. **International journal of production economics**, Elsevier, v. 231, p. 107868, 2021. Cit. on p. 29.

KIM, J.; PARK, B.-j.; KIM, J. Empirical Analysis of Autonomous Vehicle's LiDAR Detection Performance Degradation for Actual Road Driving in Rain and Fog. **Sensors**, v. 23, n. 6, 2023. ISSN 1424-8220. DOI: 10.3390/s23062972. Available from: <https://www.mdpi.com/1424-8220/23/6/2972>. Cit. on p. 20.

LI, J.; XU, R.; LIU, X.; MA, J.; CHI, Z.; MA, J.; YU, H. Learning for Vehicle-to-Vehicle Cooperative Perception Under Lossy Communication. **IEEE Transactions on Intelligent Vehicles**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, n. 4, p. 2650–2660, Apr. 2023a. ISSN 2379-8858. DOI: 10.1109/tiv.2023.3260040. Available from: <http://dx.doi.org/10.1109/TIV.2023.3260040>. Cit. on p. 31.

LI, J.; XU, R.; LIU, X.; MA, J.; CHI, Z.; MA, J.; YU, H. Learning for vehicle-to-vehicle cooperative perception under lossy communication. **IEEE Transactions on Intelligent Vehicles**, IEEE, 2023b. Cit. on p. 18.

LI, Y.; IBANEZ-GUZMAN, J. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. **IEEE Signal Processing Magazine**, IEEE, v. 37, n. 4, p. 50–61, 2020. Cit. on p. 20.

LIANG, X.; DU, X.; WANG, G.; HAN, Z. A deep reinforcement learning network for traffic light cycle control. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 2, p. 1243–1253, 2019. Cit. on p. 29.

LIU, Y.; LE CLAIR, A.; DOUDE, M.; BURCH, V. R. F. Development of a data acquisition system for autonomous vehicle systems. **International Journal of Vehicle Structures & Systems**, MechAero Foundation for Technical Research & Education Excellence, v. 10, n. 4, 2018. Cit. on p. 29.

MALIK, S.; KHAN, M. A.; EL-SAYED, H. CARLA: Car Learning to Act — An Inside Out. **Procedia Computer Science**, v. 198, p. 742–749, 2022. 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare. ISSN 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.12.316. Available from: <https://www.sciencedirect.com/science/article/pii/S1877050921025552>. Cit. on p. 25.

MOLLAH, M. B.; ZHAO, J.; NIYATO, D.; GUAN, Y. L.; YUEN, C.; SUN, S.; LAM, K.-Y.; KOH, L. H. Blockchain for the Internet of Vehicles Towards Intelligent Transportation Systems: A Survey. **IEEE Internet of Things Journal**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, n. 6, p. 4157–4185, Mar. 2021. ISSN 2372-2541. DOI: 10.1109/jiot.2020.3028368. Available from: <http://dx.doi.org/10.1109/JIOT.2020.3028368>. Cit. on p. 30.

MÜLLER, C.; COSTA, J. P. da; SANTOS, G. A. dos; MUNOZ, Y.; NOZARIAN, F.; VOZNIAK, I.; SILVA, A. S. da; HESSLER, A. Technical Report on Driving Use Cases for Highly Automated Driving and Key Performance Indicators. Cit. on p. 25.

NARDINI, G.; SABELLA, D.; STEA, G.; THAKKAR, P.; VIRDIS, A. Simu5G–An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks. **IEEE Access**, v. 8, p. 181176–181191, 2020. DOI: 10.1109/ACCESS.2020.3028550. Cit. on p. 28.

NOH, J.; JO, Y.; KIM, J.; MIN, K. Enhancing Transportation Safety with Infrastructure Cooperative Autonomous Driving System. **International Journal of Automotive Technology**, v. 25, Feb. 2024. DOI: 10.1007/s12239-024-00011-z. Cit. on p. 32.

QURESHI, K. N.; ABDULLAH, A. H. A survey on intelligent transportation systems. **Middle-East Journal of Scientific Research**, v. 15, n. 5, p. 629–642, 2013. Cit. on p. 16.

RAMASUBRAMANIAN, K.; RAMAIAH, K. Moving from legacy 24 GHz to state-of-the-art 77-GHz radar. **ATZelektronik worldwide**, Springer, v. 13, n. 3, p. 46–49, 2018. Cit. on p. 19.

REN, S.; LEI, Z.; WANG, Z.; DIANATI, M.; WANG, Y.; CHEN, S.; ZHANG, W. **Interruption-Aware Cooperative Perception for V2X Communication-Aided Autonomous Driving**. 2024. arXiv: 2304.11821 [cs.RO]. Cit. on p. 31.

SONG, Z.; XIE, T.; ZHANG, H.; LIU, J.; WEN, F.; LI, J. **A Spatial Calibration Method for Robust Cooperative Perception**. 2024. arXiv: 2304.12033 [cs.RO]. Cit. on p. 32.

TANWAR, S.; TYAGI, S.; BUDHIRAJA, I.; KUMAR, N. Tactile internet for autonomous vehicles: Latency and reliability analysis. **IEEE Wireless Communications**, IEEE, v. 26, n. 4, p. 66–72, 2019. Cit. on p. 30.

THANDAVARAYAN, G.; SEPULCRE, M.; GOZALVEZ, J.; COLL-PERALES, B. Scalable cooperative perception for connected and automated driving. **Journal of Network and Computer Applications**, v. 216, p. 103655, 2023. ISSN 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2023.103655. Available from: <https://www.sciencedirect.com/science/article/pii/S1084804523000747>. Cit. on pp. 23, 31, 52.

WANG, Y.; SU, Z.; XU, Q.; FANG, D. Trusted and Collaborative Data Sharing with Quality Awareness in Autonomous Driving. In: IEEE. ICC 2021-IEEE International Conference on Communications. 2021. P. 1–6. Cit. on p. 29.

XIANG, C.; FENG, C.; XIE, X.; SHI, B.; LU, H.; LV, Y.; YANG, M.; NIU, Z. Multi-Sensor Fusion and Cooperative Perception for Autonomous Driving: A Review. **IEEE Intelligent Transportation Systems Magazine**, v. 15, n. 5, p. 36–58, 2023. DOI: 10.1109/MITS.2023.3283864. Cit. on p. 31.

XU, R.; XIA, X.; LI, J.; LI, H.; ZHANG, S.; TU, Z.; MENG, Z.; XIANG, H.; DONG, X.; SONG, R.; YU, H.; ZHOU, B.; MA, J. **V2V4Real: A Real-world Large-scale Dataset for Vehicle-to-Vehicle Cooperative Perception**. 2023. arXiv: 2303.07601 [cs.CV]. Cit. on p. 32.

XU, R.; XIANG, H.; TU, Z.; XIA, X.; YANG, M.-H.; MA, J. **V2X-ViT: Vehicle-to-Everything Cooperative Perception with Vision Transformer**. 2022. arXiv: 2203.10638 [cs.CV]. Cit. on pp. 23, 31.

YEONG, D. J.; VELASCO-HERNANDEZ, G.; BARRY, J.; WALSH, J. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. **Sensors**, v. 21, n. 6, 2021. ISSN 1424-8220. DOI: 10.3390/s21062140. Available from: <https://www.mdpi.com/1424-8220/21/6/2140>. Cit. on p. 24.

YIN, H.; TIAN, D.; LIN, C.; DUAN, X.; ZHOU, J.; ZHAO, D.; CAO, D. V2VFormer++: Multi-Modal Vehicle-to-Vehicle Cooperative Perception via Global-Local Transformer. **IEEE Transactions on Intelligent Transportation Systems**, PP, p. 1–14, Jan. 2023. DOI: 10.1109/TITS.2023.3314919. Cit. on p. 32.

YU, G.; LI, H.; WANG, Y.; CHEN, P.; ZHOU, B. A review on cooperative perception and control supported infrastructure-vehicle system. **Green Energy and Intelligent Transportation**, v. 1, n. 3, p. 100023, 2022. ISSN 2773-1537. DOI: https://doi.org/10.1016/j.geits.2022.100023. Available from: <https://www.sciencedirect.com/science/article/pii/S2773153722000238>. Cit. on p. 23.

YURTSEVER, E.; LAMBERT, J.; CARBALLO, A.; TAKEDA, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. **IEEE Access**, v. 8, p. 58443–58469, 2020. DOI: 10.1109/ACCESS.2020.2983149. Cit. on p. 18.

ZHENG, S.; WANG, J.; RIZOS, C.; DING, W.; EL-MOWAFY, A. Simultaneous Localization and Mapping (SLAM) for Autonomous Driving: Concept and Analysis. **Remote Sensing**, v. 15, n. 4, 2023. Cited by: 15; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3390/rs15041156. Available from: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85149210687&doi=10.3390%2frs15041156&partnerID=40&md5=39b658695fb8e931ef6a46abefb5f5db>. Cit. on p. 18.

ZHU, L.; YU, F. R.; WANG, Y.; NING, B.; TANG, T. Big data analytics in intelligent transportation systems: A survey. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 20, n. 1, p. 383–398, 2018. Cit. on p. 29.

Appendix

APPENDIX A – Research Source Code

All the source code for the present research was developed with the intent of reuse to enable easier code setups and scaffolding for quickly iterable research scenarios. As an example, sensors abstractions were created where a single function call would add different sensor sets at any vehicle or pedestrians at the simulation scenarios. Another example is the network simulations, where it is now possible to add a sensor to a vehicle, already knowing that data on that sensor will be transmitted, with a small number of configuration parameters.

The source code for the different research experiments is available in several github repositories (https://github.com/av-data-research-group/carla-data-collection for the first experiments, https://github.com/carlanet/carlanetpp for the network simulation framework and https://github.com/aassilva/CarlaNetpp for the network and attack experiments) and can be divided as following: 1. Code for data collection without network simulation; 2. Data collection with network simulation; 3. B5GCyberTestV2X code with interfaces for implementing vehicle control and data sharing funcionalities; 4. Spoofing attack scenarios experiment code; 5. Drone scenario generation.

Below we transcribe some files in an attempt to share the most meaningful part of the source code. The first file is a simple demonstration of the data collection process with the CARLA simulator.

Code A.1 – Basic starting code for a simple data collection example, using a single vehicle mounted with 2 cameras and a lidar sensor

```
1
2  import carla
3  import random
4  import time
5
6
7  ### Client
8
9  client = carla.Client('localhost', 2000)
10
11 client.set_timeout(10.0)
12 world = client.get_world()
13
14 # world.__dir__()
15 for map_name in client.get_available_maps():
16     print(map_name)
```

```
17 world.get_map().name
18
19 # Load map
20 world = client.load_world("Town10HD_Opt")
21
22 # Get blueprint library and filter only cars
23 vehicle_blueprints =
      world.get_blueprint_library().filter('*vehicle*')
24 for idx, blueprint in enumerate(vehicle_blueprints):
25     print(idx, blueprint.id)
26     if idx == 10:
27         break
28
29 ### Add vehicle to world
30
31 # Get maps spawn points
32 spawn_points = world.get_map().get_spawn_points()
33
34 # Tesla Cybertruck
35 vehicle = world.try_spawn_actor(list(vehicle_blueprints)[26],
      random.choice(spawn_points))
36 print(vehicle)
37
38 vehicle_transform = vehicle.get_transform()
39 vehicle_transform.location.z += 2.0
40 world.get_spectator().set_transform(vehicle_transform)
41
42 ### Add sensors to car
43
44 # time in seconds to collect data
45 SENSOR_TICK = 3
46
47 def get_cam_blueprint(world):
48     cam_bp =
          world.get_blueprint_library().find('sensor.camera.rgb')
49     cam_bp.set_attribute("image_size_x",str(400))
50     cam_bp.set_attribute("image_size_y",str(300))
51     cam_bp.set_attribute("fov",str(100))
52     cam_bp.set_attribute("sensor_tick",str(SENSOR_TICK))
53     return cam_bp
54
55
56 def get_lidar_blueprint(world):
57     lidar_bp =
          world.get_blueprint_library().find('sensor.lidar.ray_cast')
58     lidar_bp.set_attribute('sensor_tick', str(SENSOR_TICK))
59     lidar_bp.set_attribute('channels', '64')
60     lidar_bp.set_attribute('points_per_second', '1120000')
61     lidar_bp.set_attribute('upper_fov', '30')
62     lidar_bp.set_attribute('range', '100')
63     lidar_bp.set_attribute('rotation_frequency', '100')
64     return lidar_bp
```

```
65
66 camera_1_init_trans = carla.Transform(carla.Location(z=2.3))
67 camera_2_init_trans = carla.Transform(carla.Location(z=2.3),
       carla.Rotation(yaw=180))
68 lidar_init_trans = carla.Transform(carla.Location(z=3.0))
69
70 # creates camera and attaches it to the vehicle
71 camera_1_bp = get_cam_blueprint(world)
72 camera_2_bp = get_cam_blueprint(world)
73 camera_1 = world.spawn_actor(camera_1_bp, camera_1_init_trans,
       attach_to=vehicle)
74 camera_2 = world.spawn_actor(camera_2_bp, camera_2_init_trans,
       attach_to=vehicle)
75 camera_1.listen(lambda image:
       image.save_to_disk(f"sensors/camera_1/...{image.frame}.png"))
76 camera_2.listen(lambda image:
       image.save_to_disk(f"sensors/camera_2/...{image.frame}.png"))
77
78 # creates lidar and attaches it to the vehicle
79 lidar_bp = get_lidar_blueprint(world)
80 lidar = world.spawn_actor(lidar_bp, lidar_init_trans,
       attach_to=vehicle)
81 lidar.listen(lambda data:
       data.save_to_disk(f'sensors/lidar/...{data.frame}.ply'))
82
83 # Adds vehicle to Traffic manager
84 vehicle.set_autopilot(True)
85
86 # Faz o spectator seguir o carro
87 spectator = world.get_spectator()
88 while True:
89     transform = camera_1.get_transform()
90     transform.location.z += 2.0
91     spectator.set_transform(transform)
92     time.sleep(0.004)
93
94 # Remove o carro da simulação
95 vehicle.destroy()
```

The experiment code establishes a connection to the CARLA server running locally, allowing control of to the simulation. After connecting, it loads the high-definition urban environment "Town10HD_Opt" and spawns a Tesla Cybertruck at a randomly selected spawn point. The script then configures and attaches multiple sensors to the vehicle: two RGB cameras positioned to capture front and rear views, and a high definition LiDAR sensor mounted on the roof. Each sensor is calibrated with specific parameters - the cameras are set with a 100 degree field of view and resolution of 400x300 pixels, while the LiDAR is configured with 64 channels, capturing 1.12 million points per second with a range of 100 meters. Please note that CPU and memory usage do scale with image and LIDAR definition.

The script implements a data collection pipeline where sensor outputs are automatically saved to disk at regular 3-second intervals. Camera images are stored as PNG files, while Lidar point clouds are saved as PLY files, organizing them by sensor type and frame number. This approach allows the creation of datasets capturing the vehicle's perspective as it navigates through the environment. The vehicle's movement is controlled by CARLA's built-in Traffic Manager through the autopilot function, ensuring realistic traffic behavior while data is being collected.

A spectator camera is programmed to follow the vehicle throughout the simulation, to provide the research team with a third-person view of the data collection process.

The following code represents a piece of the network data transmission capabilites.

Code A.2 – Simple data collection example, now using network simulation capabilities.

```
import random
import carla
import datetime
from carla import libcarla, ActorBlueprint
import traceback

from pycarlanet import CarlanetActor
from pycarlanet import CarlanetManager
from pycarlanet import CarlanetEventListener, SimulatorStatus

import math
import os
import pandas as pd

AUTO_PILOT = True
NUM_VEHICLES = 0

class B5GCyberTestV2X(CarlanetEventListener):
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.carlanet_manager = CarlanetManager(5555, self,
            log_messages=True)

        self.client = self.sim_world = self.carla_map = None
        self.carlanet_actors = dict()
        self._car = None
        self.remote_agent = RemoteAgent()

    def start_simulation(self):
        self.carlanet_manager.start_simulation()

    def omnet_init_completed(self, run_id, carla_configuration,
        user_defined) -> (SimulatorStatus, World):
        random.seed(carla_configuration['seed'])

```

```
35          print(f'OMNeT world is completed with the id {run_id}')
36          world = user_defined['carla_world']  # Retrieve from
               user_defined
37
38          client: libcarla.Client = carla.Client(self.host,
               self.port)
39          client.set_timeout(15)
40          sim_world = client.load_world("Town01")
41          #sim_world = client.load_world("Town10HD_Opt")
42
43          settings = sim_world.get_settings()
44
45          settings.synchronous_mode = False
46          settings.fixed_delta_seconds = None
47          settings.no_rendering_mode = False
48          settings.fixed_delta_seconds =
               carla_configuration['carla_timestep']
49
50          #sim_world.set_weather(carla.WeatherParameters.ClearNight)
51
52          sim_world.apply_settings(settings)
53          sim_world.tick()
54
55          traffic_manager = client.get_trafficmanager()
56          traffic_manager.set_synchronous_mode(False)
57          traffic_manager.set_random_device_seed(
58              carla_configuration['seed']
59          )
60          sim_world.tick()
61
62          client.reload_world(False)  # Reload map keeping the world
               settings
63
64          sim_world.tick()
65          self.client, self.sim_world = client, sim_world
66          self.carla_map = self.sim_world.get_map()
67
68          return SimulatorStatus.RUNNING, self.sim_world
69
70      def actor_created(self, actor_id: str, actor_type: str,
           actor_config) -> CarlanetActor:
71          if actor_type == 'car':  # and actor_id == 'car_1':
72              blueprint: ActorBlueprint = random.choice(
73                  self.sim_world.get_blueprint_library().filter(
74                      "vehicle.tesla.model3"
75                  ))
76
77              spawn_points =
                   self.sim_world.get_map().get_spawn_points()
78              # Attach sensors
79              spawn_point = random.choice(spawn_points)
80              print(blueprint)
```

```
81              response = self.client.apply_batch_sync(
82                  [carla.command.SpawnActor(blueprint, spawn_point)]
83              )[0]
84              carla_actor: carla.Vehicle =
                    self.sim_world.get_actor(response.actor_id)
85
86              carla_actor.set_simulate_physics(True)
87              if AUTO_PILOT:
88                  carla_actor.set_autopilot(True)
89
90              carlanet_actor = CarlanetActor(carla_actor, True)
91              self.carlanet_actors[actor_id] = carlanet_actor
92              self._car = carlanet_actor
93
94              self.vehicle = carla_actor
95
96              self.sim_world.tick()
97
98              #camera_sensor = TeleCarlaCameraSensor(2.2)
99              #camera_sensor.attach_to_actor(self.sim_world,
                    carla_actor)
100             self.actor_id = actor_id
101
102             # send spectator to camera position
103             spectator = self.sim_world.get_spectator()
104             transform = self.vehicle.get_transform()
105             spectator.set_transform(transform)
106
107             if NUM_VEHICLES > 0:
108                 vehicle_blueprints =
                        self.sim_world.get_blueprint_library().filter(
109                         '*vehicle*'
110                     )
111                     for _ in range(NUM_VEHICLES):
112                         npc_vehicle = self.sim_world.try_spawn_actor(
113                             list(vehicle_blueprints)[38],
                                    random.choice(spawn_points)
114                         )
115                         if npc_vehicle is None:
116                             print("vehicle is none")
117                         else:
118                             npc_vehicle.set_autopilot(True)
119
120             return carlanet_actor
121         else:
122             raise RuntimeError(f'I don\'t know this type
                    {actor_type}')
123
124
125     def carla_init_completed(self):
126         super().carla_init_completed()
127
```

```
128     def before_world_tick(self, timestamp) -> None:
129         super().before_world_tick(timestamp)
130 
131     def carla_simulation_step(self, timestamp) -> SimulatorStatus:
132         self.sim_world.tick()
133         # Do all the things, save actors data
134         if timestamp > 100:  # ts_limit
135             return SimulatorStatus.FINISHED_OK
136         else:
137             return SimulatorStatus.RUNNING
138 
139     # get light control enum from int value
140     @staticmethod
141     def _str_to_light_control_enum(light_value):
142         if light_value == '0':
143             return carla.VehicleLightState.NONE
144         elif light_value == '1':
145             return carla.VehicleLightState.Position
146         elif light_value == '2':
147             return carla.VehicleLightState.Brake
148         else:
149             return carla.VehicleLightState.All
150 
151     # get int value from light control enum
152     @staticmethod
153     def _light_control_enum_to_str(light_enum):
154         if light_enum == carla.VehicleLightState.NONE:
155             return '0'
156         elif light_enum == carla.VehicleLightState.Position:
157             return '1'
158         elif light_enum == carla.VehicleLightState.Brake:
159             return '2'
160         else:
161             return '3'
162 
163 
164 
165 
166     def standard_message(self, timestamp, user_defined_message) ->
        (SimulatorStatus, dict):
167         # Handle the action of the actors in the world
            (apply_commands, calc_instruction)
168         # es: apply_commands with id command_12 to actor with id
            active_actor_14
169         if user_defined_message['msg_type'] == 'LIGHT_COMMAND':
170 
171             #convert enum value to enum
172             next_light_state = self._str_to_light_control_enum(
173                 user_defined_message['light_next_state']
174             )
175             self._car.set_light_state(next_light_state)
176 
```

```
177             msg_to_send = {'msg_type': 'LIGHT_UPDATE',
178                            'light_curr_state':
                                  self._light_control_enum_to_str(
179                                   self._car.get_light_state())
180                            }
181
182             print("LIGHT CURR STATE: ",
                    self._car.get_light_state(), '\n\n')
183             return SimulatorStatus.RUNNING, msg_to_send
184
185         elif user_defined_message['msg_type'] == 'LIGHT_UPDATE':
186             curr_light_state = self._str_to_light_control_enum(
187                 user_defined_message['light_curr_state']
188             )
189             next_light_state =
                    self.remote_agent.calc_next_light_state(
190                 curr_light_state
191             )
192             print("LIGHT CURR STATE: ", curr_light_state, "LIGHT
                    NEXT STATE: ", next_light_state, '\n')
193
194             msg_to_send = {'msg_type': 'LIGHT_COMMAND',
195                            'light_next_state':
                                  self._light_control_enum_to_str(
196                                    next_light_state
197                            )
198                           }
199
200             # GET locations and speed and save it to csv
201             print("#" * 50)
202             print("getting sensor data from car")
203             self.remote_agent.process_vehicle_data(self.vehicle,
                    self.carlanet_actors)
204
205             ### CONTROL STAGE
206             if not AUTO_PILOT:
207                 pass
208
209             # send spectator to camera position
210             spectator = self.sim_world.get_spectator()
211             transform = self.vehicle.get_transform()
212             spectator.set_transform(transform)
213
214             return SimulatorStatus.RUNNING, msg_to_send
215         else:
216             raise RuntimeError(f"I don\'t know this type
                    {user_defined_message['msg_type']}")
217
218     def simulation_finished(self, status_code: SimulatorStatus):
219         super().simulation_finished(status_code)
220
221     def simulation_error(self, exception):
```

```
222         traceback.print_exc()
223         super().simulation_error(exception)
224 
225 
226 class RemoteAgent:
227 
228     def __init__(self):
229         self.light_state = carla.VehicleLightState.NONE
230 
231     def calc_next_light_state(self, light_state:
        carla.VehicleLightState):
232         return carla.VehicleLightState.NONE
233 
234     def generate_carla_nodes_positions(self, carlanet_actors):
235         nodes_positions = []
236         for actor_id, actor in carlanet_actors.items():
237             transform: carla.Transform = actor.get_transform()
238             velocity: carla.Vector3D = actor.get_velocity()
239             position = dict()
240             position['actor_id'] = actor_id
241             position['position'] = [transform.location.x,
                transform.location.y, transform.location.z]
242             position['rotation'] = [transform.rotation.pitch,
                transform.rotation.yaw, transform.rotation.roll]
243             position['velocity'] = [velocity.x, velocity.y,
                velocity.z]
244             position['is_net_active'] = actor.alive
245             nodes_positions.append(position)
246         return nodes_positions
247 
248     def process_vehicle_data(self, vehicle, carlanet_actors):
249         file_name =
            f"sensors_high_traffic/gnss/vehicle-id-{vehicle.id}.csv"
250 
251         if not os.path.exists(f"sensors_high_traffic/gnss/"):
252             os.makedirs(f"sensors_high_traffic/gnss/")
253 
254         positions_obj =
            self.generate_carla_nodes_positions(carlanet_actors)[0]
255         x_speed = positions_obj["velocity"][0]
256         y_speed = positions_obj["velocity"][1]
257         z_speed = positions_obj["velocity"][2]
258         speed = math.sqrt(x_speed**2 + y_speed**2 + z_speed**2)
259         current_time = datetime.datetime.now()
260 
261         save_obj = [{
262             "id": vehicle.id,
263             "timestamp": str(current_time),
264             "lat": positions_obj["position"][0],
265             "long": positions_obj["position"][1],
266             "alt": positions_obj["position"][2],
267             "speed": speed
```

```
        }]

        if os.path.exists(file_name):
            df = pd.read_csv(file_name)
        else:
            df = None

        df_new_data = pd.DataFrame.from_dict(save_obj)

        if df is None:
            df_new_data.to_csv(file_name, index=False)
        else:
            df = pd.concat([df, df_new_data])
            df.to_csv(file_name, index=False)


if __name__ == '__main__':
    my_world = B5GCyberTestV2X('localhost', 2000)
    my_world.start_simulation()
```

This code implements a more advanced CARLA simulation that integrates with OMNET++/SIMU5G for network simulation capabilities. The architecture creates a bridge between two simulators: CARLA handles the vehicle physics, sensors, and urban environment, while SIMU5G/OMNET++ simulates 5G network communication between vehicles and infrastructure. The integration is managed through a custom CarlanetManager class that keeps the two simulators synced.

The main controller class, B5GCyberTestV2X, implements the CarlanetEventListener interface to set up the simulation environment and handle event callbacks between CARLA and OMNET++. It connects to the CARLA server on localhost, initializes a CarlanetManager for communicating with OMNET++, and sets up synchronization between the simulators with configurable timesteps. Unlike the previous code that used Town10HD_Opt, this implementation loads the simpler Town01 map and configures traffic manager and weather settings accordingly.

For vehicle management, the code creates a Tesla Model 3 (instead of the Cybertruck in the previous example), places it at a random spawn point, and optionally enables autopilot through the AUTO_PILOT flag. It can also spawn additional NPC vehicles based on the NUM_VEHICLES setting. The network-vehicle integration is handled through a messaging system that supports two main message types: LIGHT_COMMAND for controlling vehicle lights based on network commands, and LIGHT_UPDATE for reporting the current light state back to the network.

Data collection is managed by the RemoteAgent class, which gathers vehicle telemetry data including position, velocity, and rotation. This data is saved to CSV files in a structured directory format, with each vehicle's information stored in a separate file with times-

tamps. The simulation maintains synchronization by having each simulator wait for the other before advancing to the next time step.

The key differences from the basic code include network integration through OMNET++/SIMU5G, more structured data collection with CSV storage, an event-based architecture using listeners for inter-simulator communication, and a more sophisticated vehicle control mechanism that can receive commands over the network. This implementation represents system where vehicle behavior can be influenced by network communications.

The next piece of code represents one of the use cases with data collection, now with the network and communication already abstracted in utility objects:

Code A.3 – Example Use case where data is collected from 3 cars and a drone may be used to perform attacks to the communication channels

```
import carla
import time

BAD_SCENARIO = True

from utils import (
    attach_all_sensors
)

def set_spectator(vehicle):
    spectator = world.get_spectator()
    transform = vehicle.get_transform()
    transform.location.z += 2.0
    spectator.set_transform(transform)

# entity used to preserve sensors in scope
simulation_actors = {}

client = carla.Client('localhost', 2000)
client.set_timeout(10.0)

# world = client.get_world()
world = client.load_world("Town01")
world.set_weather(carla.WeatherParameters.CloudyNoon)

time.sleep(5)

# Get blueprint library and filter only cars
# vehicle_blueprints =
   world.get_blueprint_library().filter('*vehicle*')
cyber_blueprints =
   world.get_blueprint_library().filter('vehicle.tesla.cybertruck')
c3_blueprints =
   world.get_blueprint_library().filter('vehicle.citroen.c3')
model_3_blueprints =
   world.get_blueprint_library().filter('vehicle.tesla.model3')
```

```
#drone_blueprint = world.get_blueprint_library().filter('vehicle.micro.microlino')[0]
drone_blueprint = world.get_blueprint_library().filter('vehicle.drone.drone1')[0]

# Get maps spawn points
spawn_points = world.get_map().get_spawn_points()

camera_init_trans = carla.Transform(carla.Location(z=2.7,x=1.0))
back_camera_init_trans = carla.Transform(carla.Location(z=2.7), carla.Rotation(yaw=180))
lidar_init_trans = carla.Transform(carla.Location(z=3.0))

# wanted positions: 181, 183, 177, 219
# wanted positions other lane: 65, 163

blueprint_list = [
    list(cyber_blueprints)[0],
    list(c3_blueprints)[0],
    list(model_3_blueprints)[0],
]
selected_locations = [181, 177, 163]
hero_list = []
for idx, location in enumerate(selected_locations):
    hero_n = world.try_spawn_actor(blueprint_list[idx], spawn_points[location])
    simulation_actors[f"vehicle_{hero_n.id}"] = hero_n
    if BAD_SCENARIO and idx == 2:
        attach_all_sensors(world, hero_n, simulation_actors, drone_attack=True)
    else:
        attach_all_sensors(world, hero_n, simulation_actors)
    hero_list.append(hero_n)


# set_spectator(hero_list[0])
print("Simulation actors:", simulation_actors)

if BAD_SCENARIO:
    # Malicious Drone
    drone_location = carla.Location(x=392.791443, y=107.608482, z=14.855516)
    drone_rotation = carla.Rotation(pitch=-27.212101, yaw=-89.532944, roll=-0.025725)
    drone_spawn_point = carla.Transform(drone_location, drone_rotation)
    drone_1 = world.try_spawn_actor(drone_blueprint, drone_spawn_point)
    drone_1.set_simulate_physics(False)


tm = client.get_trafficmanager()
```

```
tm_port = tm.get_port()

for hero_n in hero_list:
    hero_n.set_autopilot(True, tm_port)

if BAD_SCENARIO:
    tm.distance_to_leading_vehicle(hero_list[0], 0)
    tm.vehicle_percentage_speed_difference(hero_list[0], -80)
    tm.vehicle_percentage_speed_difference(hero_list[1], -75)
    tm.vehicle_percentage_speed_difference(hero_list[2], -80)
    tm.collision_detection(hero_list[0], hero_list[2], False)
    tm.collision_detection(hero_list[0], hero_list[1], False)
    tm.collision_detection(hero_list[2], hero_list[0], False)

spectator = world.get_spectator()
# Camera take - drone view
# spec_location = carla.Location(x=389.039062, y=102.294785,
   z=19.252405)
# spec_rotation = carla.Rotation(pitch=-31.561518, yaw=73.702972,
   roll=-0.018310)
# spectator.set_transform(carla.Transform(spec_location,
   spec_rotation))

# Camera take - lateral view
spec_location = carla.Location(x=377.332733, y=125.954506,
   z=35.793575)
spec_rotation = carla.Rotation(pitch=-54.568348, yaw=-0.546539,
   roll=-0.018311)
spectator.set_transform(carla.Transform(spec_location,
   spec_rotation))

starting_time = time.time()
print("current time", starting_time)
triggered = False
while True:
    # transform = camera_1.get_transform()
    # # transform.location.z += 1.0
    # spectator.set_transform(transform)
    time.sleep(0.1)

    ## DATA COLLECTION HAPPENS AUTOMATICALLY DURING THIS LOOP

    # find out locations
    spectator = world.get_spectator()
    print(spectator.get_transform())
```

This code creates a more scenario-focused CARLA simulation designed to test specific traffic situations, particularly centered around a potential malicious attack scenario. The implementation begins by connecting to a local CARLA server and loading the "Town01" map with cloudy noon weather conditions, allowing time for the world to properly initialize.

Rather than using random vehicles, this scenario have specific vehicle models - a Tesla Cybertruck, Citroen C3, and Tesla Model 3. The vehicles are placed at predetermined spawn points for reproducibility reasons (positions 181, 177, and 163) to create a controlled traffic scenario. Each vehicle is added to a "simulation_actors" dictionary to maintain references and has sensors attached through the abstracted "attach_all_sensors" function, with special configuration for the third vehicle when the BAD_SCENARIO flag is enabled.

We then implement a potentially hazardous situation when BAD_SCENARIO is set to true. In this case, a drone actor is positioned at specific coordinates above the road and configured not to simulate physics (remaining stationary). The traffic manager is then set up with aggressive parameters: following distance for the first vehicle is minimized, all vehicles are instructed to drive much faster than normal, and collision detection between specific vehicle pairs is deliberately disabled.

The simulation's viewpoint is precisely positioned using a spectator camera at coordinates that provide a lateral view of the upcoming scenario. If the BAD_SCENARIO is set to true, the drone will change the data that reaches vehicle 2 (the one trying to do the outtake in the road), making the data arrive with a big delay. With no visibility, vehicle 2 tries to outtake vehicle 1 and ends up coliding with vehicle 3.