MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

MULTI-USER REAL-TIME CONTROLLABLE CONNECTED CARS

TESTING PLATFORM

BURAK ŞENKUŞ

MASTER THESIS

Department of Computer Engineering

Thesis Supervisor
Assoc. Prof. Müjdat SOYTÜRK

ISTANBUL, 2023

ACKNOWLEDGEMENT

First and foremost, I extend my heartfelt appreciation to my teammates Berkay Yaman, Hüseyin Aydın and Halil Hakan Çopur. Their dedication, expertise, and meticulous attention to detail greatly enhanced the quality of this work. Their insightful suggestions and thoughtful discussions were instrumental in shaping the direction of our research.

And I would like to acknowledge the continuous support and understanding of my family and friends throughout this endeavor. Their unwavering encouragement and belief in my abilities have been a constant source of motivation.

This article would not have been possible without the combined efforts and dedication of VeNIT Lab. I am truly fortunate to have had the opportunity to collaborate with such an exceptional group of individuals.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. RELATED WORK 3

2.1. V2X Communications Testing Methodologies 3

2.2. Differences of Our Test Platform 7

3. METHODOLOGY 9

3.1. System Architecture 11

3.1.1. User 12

3.1.2. Controller 12

3.1.3. CARLA Server 12

3.1.4. CARLA Client 13

3.1.5. Protocol Stack 14

3.1.6. 802.11p Interface 16

3.1.7. GPS Daemon 16

3.1.8. CAN Interface 16

3.1.9. Communication Controller 17

3.1.10. GPS Feeder 17

3.1.11. CAN Feeder 17

3.1.12. Applications 18

3.1.13. MQTT Broker 18

3.1.14. HMI 19

3.2. Packages Used 21

3.2.1. CARLA 21

3.2.2. Docker 21

3.2.3. mac80211_hwsim 22

3.2.4. Mosquitto 22

3.2.5. gpsd 22

3.2.6. Vanetza 23

4. EXPERIMENTAL EVALUATION 24

4.1. Performance Evaluation.....24

4.2. Longitudinal Collision Risk Warning Application.....37

4.2.1. Frontal Collision Risk Test 38

4.2.2. Forward Collision Risk Test 39

4.2.3. Different Lane Test 41

5. CONCLUSION AND FUTURE WORK.....44

ÖZET

V2X haberleşmesi, araçlar, altyapı bileşenleri ve trafikteki diğer istasyonlar arasında güvenliği ve bilgi paylaşımını arttırmaktadır. Bağlantılı araçlardan ve altyapı sistemlerinden paylaşılan bilgiler birçok uygulamanın geliştirilmesine olanak sağlamaktadır. Bu alanda geliştirilen yazılım ve donanımın gerçek dünyaya uygulanması için kapsamlı testlere ihtiyaç vardır. Geliştirilen yazılım ve donanımın doğrulanması ve geçerliliğinin sağlanması için birçok test yöntemi mevcuttur. Her bir yöntemin gerçekçilik, maliyet, tekrarlanabilirlik, güvenilirlik ve zaman kriterleri açısından artı ve eksileri bulunmaktadır. Bu çalışmada, bahsedilen kriterleri optimize etmeyi amaçlayarak, bağlantılı araçların uygulama testlerinin insanların sürece dahil olarak gerçekleştirilebileceği bir test platformu geliştirilmiştir. Bu platform, birden fazla kullanıcının kontrolcü yardımıyla kendi sanal araçlarını sürerek V2X uygulamalarını test etmelerine olanak sağlar. Test platformunun geliştirilmesi sürecinde, araçların yazılım bileşenlerinin gerçek dünyadaki karşılıklarına mümkün olduğunca yakın bir şekilde tasarlanması için çaba harcanmıştır. Test platformunun stabilitesini ve performansını göstermek amacıyla birkaç sistem içi gecikme ölçümü yapılmıştır. Ölçümler, ihmal edilebilir düzeyde gecikmelerin olduğunu ve test platformunun yüksek performans sergilediğini göstermiştir. Ayrıca, platformda uygulama testlerinin yapılabilirliğini göstermek için geliştirilen basit bir LCRW uygulaması denenmiştir. Uygulama testi, geliştirilen algoritmanın beklendiği şekilde tepki verdiğini ve çarpışma riskleri konusunda kullanıcıya gerekli uyarıları sağladığını göstermiştir.

ABSTRACT

V2X communication improves safety and information sharing among vehicles, infrastructure components, and other stakeholders in the traffic. Information shared between connected vehicles and infrastructure systems enables the development of numerous applications. The application of the software and hardware developed in this field to the real world is not possible without comprehensive testing. Many testing methodologies exist for verifying and validating the developed software and hardware. Each of these methodologies has trade-offs in terms of realism, cost, repeatability, reliability, and time. With the aim of optimizing this trade-off, we developed a human-in-the-loop test platform where application tests of connected vehicles can be conducted. This platform enables multiple users to drive their virtual vehicles through controller assistance and test V2X applications. In the development of the test platform, efforts have been made to isolate and design the software components of vehicles as closely as possible to their real-world counterparts. Several internal delay measurements were conducted to demonstrate the stability and performance of our test platform. The measurements revealed negligible delays, indicating high performance of the test platform. Additionally, to demonstrate the feasibility of application testing on our platform, a simple LCRW application was implemented. The application test demonstrated that the developed algorithm responded as expected, providing necessary warnings to the user regarding collision risks.

SYMBOLS

p_1 : Processing delay of CARLA Client

p_2 : Processing delay of Network Emulator

p_3 : Processing delay of CA Service

p_4 : Processing delay of 802.11p Interface

t_1 : Transmission delay between Controller and CARLA Client

t_2 : Transmission delay between CARLA Client and Network Emulator

t_3 : Transmission delay between Network Emulator and CA Service

t_4 : Transmission delay between CA Service and 802.11p Interface

t_5 : Transmission delay between 802.11p Interface and Communication Controller

ABBREVIATIONS

BTP **:** Basic Transport Protocol

C-ITS : Cooperative Intelligent Transportation Systems

CA : Cooperative Awareness

CAM : Cooperative Awareness Message

DCC : Decentralized Congestion Control

DEN : Decentralized Environmental Notification

DENM : Decentralized Environmental Notification Message

ETSI : European Telecommunications Standards Institute

FPS : Frame per second

GN : GeoNetworking

GPS : Global Positioning System

GPU : Graphics Processing Unit

HiL : Hardware-in-the-loop

HMI : Human Machine Interface

IEEE : Institute of Electrical and Electronics Engineers

ITS : Intelligent Transportation Systems

ITS-S : Intelligent Transportation Systems Station

LCRW : Longitudinal Collision Risk Warning

MQTT : Message Queuing Telemetry Transport

NMEA : National Marine Electronics Association

OBU : On-board Unit

OSM : OpenStreet Map

RSU : Roadside Unit

SUMO : Simulation of Urban Mobility

TCP : Transmission Control Protocol

TTC : Time-to-collision

USB : Universal Serial Bus

V2X : Vehicle-to-everything

VANET : Vehicular Ad Hoc Network

ViL : Vehicle-in-the-loop

LIST OF FIGURES

Figure 2.1. Field testing of collision risk warning between two vehicles 6

Figure 3.1: High level goal of required system 9

Figure 3.2. The high-level architecture of the system which two users included. 11

Figure 3.3. An example CARLA Client UI. 14

Figure 3.4. An example HMI (a car dashboard with a gps device photo, 2022) 20

Figure 3.5. A screenshot of HMI was developed for this study. 20

Figure 4.1: Sequence diagram of control of a single vehicle 25

Figure 4.2: t_2 histogram for the first run with 2 vehicles 29

Figure 4.3: p_2 histogram for the first run with 2 vehicles 29

Figure 4.4: t_3 histogram for the first run with 2 vehicles 30

Figure 4.5: p_3 histogram for the first run with 2 vehicles 30

Figure 4.6: t_4 histogram for the first run with 2 vehicles 31

Figure 4.7: t_3 histogram for the second run with 2 vehicles 32

Figure 4.8: Comparison of first and second run with 2 vehicles 33

Figure 4.9: t_3 histogram for the first run with 3 vehicles 34

Figure 4.10: t_3 histogram for the second run with 3 vehicles 35

Figure 4.11: Comparison of first and second run with 3 vehicles 36

Figure 4.12. Red vehicle's perspective during a frontal collision risk 38

Figure 4.13. Collision risk warning on HMI during frontal collision risk 39

Figure 4.14. Black vehicle's perspective during a frontal collision risk 39

Figure 4.15. Red vehicle’s perspective during a forward collision risk 40

Figure 4.16. Collision risk warning on HMI during forward collision risk 41

Figure 4.17. Red vehicle’s perspective during driving on different lanes 42

Figure 4.18. No collision risk on HMI when vehicles are on different lanes 42

Figure 4.19. Black vehicle's perspective during driving on different lanes 43

LIST OF TABLES

Table 4.1. Hardware specifications of the computer 27

Table 4.2: Statistics of the first run with 2 vehicles 31

Table 4.3: Statistics of the second run with 2 vehicles..... 32

Table 4.4: Statistics of the first run with 3 vehicles 34

Table 4.5: Statistics of the second run with 3 vehicles..... 35

1. INTRODUCTION

Vehicle-to-everything (V2X) communication is a type of communication technology that allows vehicles to communicate with each other and with surrounding stations, such as infrastructure systems, pedestrians, and cyclists in real time. V2X can be used in various applications such as collision avoidance, traffic efficiency, and autonomous driving to improve road safety and transportation efficiency (Chen, Lu, Fang, & Chan, 2018). With the increasing number of connected vehicles on the road, V2X communication has become a crucial technology for future transportation.

V2X communication is generally realized between connected vehicles that use on-board unit (OBU) and roadside units (RSU). In addition to the aforementioned elements, various entities, such as pedestrians, cyclists, and public transportation vehicles present in traffic, can engage in information exchange through V2X communication. All entities equipped with communication capabilities are collectively referred to as Intelligent Transportation System Stations (ITS-S).

The advent of V2X communications has resulted in numerous scenarios and applications. These applications are typically categorized informally as Day 1, Day 2, and Day 3+ applications (CAR 2 CAR Communication Consortium, 2019). Day 1 applications primarily focus on station safety and human life. Day 2 and Day 3+ applications are more complex and have greater requirements. In addition to prioritizing safety, they encompass in-vehicle information, platooning, and entertainment applications.

Testing these applications is essential for ensuring their reliability and effectiveness. Field testing is an absolute requirement for application testing, but it has several limitations, including low repeatability, high cost, and limited testing scenarios (Karagiannis, ve diğerleri, 2011). To overcome these limitations and reduce the need for field testing, virtual testing frameworks have been developed to provide a more controlled and flexible environment for application testing. These frameworks include various components, such as a simulation engine, communication modules, and virtual vehicles that work together to create an integrated testing environment.

However, virtual testing frameworks also have limitations, particularly because of

limited human interaction. Most of these frameworks rely on static definitions of vehicle mobility. This situation significantly compromises the reliability of the tests because they are not sufficiently comprehensive.

To address the limitations of current studies, this paper presents a human-in-the-loop testing platform that enables users to control vehicles similar to the real world. The platform consists of several components, including a simulation engine, communication modules, and a human-machine interface (HMI) that work together to create a realistic and flexible testing environment.

In summary, this study aimed to provide a comprehensive testing platform for V2X communication that combines the advantages of virtual testing and human interaction. This platform enables users to test V2X applications in a controlled and realistic environment, thereby eliminating the majority of issues that may occur during field testing.

2. RELATED WORK

In this section, we discuss the fundamental test methods used for the validation of V2X communications and its applications, and provide some examples. In addition, we specify which test method(s) our study employs and how it differs from the example studies provided.

2.1. V2X Communications Testing Methodologies

V2X communication and application testing can be performed using various methods. Some of these testing methods were listed and described as follows.

- Simulation-based testing
- Hardware-in-the-loop (HiL) testing
- Human-in-the-loop testing
- Vehicle-in-the-loop (ViL) testing
- Field test

Simulation testing: This methodology involves the creation of a virtual environment that simulates real-world scenarios to test the V2X communication and applications. This methodology allows the testing of various scenarios in a controlled and repeatable environment. This is a cost-effective approach and enables the evaluation of scenarios that may be too dangerous or difficult to test in the real world. However, it may not fully reflect real-world conditions, and the accuracy of the simulation can be limited by the quality of the models used.

Veins (Sommer, German, & Dressler, 2011), is a simulation-based testing framework that provides simulation models for vehicular networking. It allows writing application-specific simulation code and takes care of modeling lower protocol layers, node mobility, and setting up, executing, and collecting results during and after the simulation. Veins also includes many simulation models applicable to vehicular network simulation, which can be used as a toolbox to build comprehensive and detailed simulations.

Another study (Riebl, Günther, Facchi, & Wolf, 2015) discussed the development of a modular simulation environment for testing new Advanced Driver Assistance Systems (ADASs) based on V2X communications. The proposed framework, named Artery, is an extension to Veins and incorporates an implementation of the European Telecommunications Standards Institute (ETSI) ITS-G5 protocol stack. This study highlights the requirements for a modular simulation environment, including reusability, isolation, rapid prototyping, and extensible message formats. The proposed framework aims to assist the development and analysis of VANET applications in a network simulation environment.

HiL testing: This methodology involves the use of physical hardware components to test the V2X communication and applications in a simulated environment. This methodology allows for the testing of real hardware components under controlled conditions. This is an effective approach for identifying and diagnosing hardware-related issues. However, this can be expensive and may not fully replicate the complexity of real-world conditions.

The CarTest (Wang & Zhu, 2022) study discussed the development of a HiL V2X simulation framework. The framework was designed to test the functionality of V2X applications, such as Forward Collision Warning (FCW), Intersection Collision Warning (ICW), and Cooperative Adaptive Cruise Control (CACC), in a laboratory setting. This study highlights the challenges and solutions of HiL testing, and the benefits of HiL testing over field testing. A framework was created to simulate dangerous scenarios in a reproducible manner.

Another study (Szendrei, Varga, & Bokor, 2018) based on Simulation of Urban Mobility (SUMO) discussed the need for testing and prototyping Cooperative Intelligent Transportation Systems (C-ITS), which relies on V2X communication. The authors proposed a HiL V2X simulation framework that allows for cost-efficient and simple testing and prototyping of cooperative vehicular solutions. The framework was designed to replace costly, time-consuming, and dangerous field tests with an easy-to-install tabletop laboratory test and development suite. This article provides an overview of the proposed simulation framework and its building blocks, and highlights the benefits and limitations of the scheme. The article also surveys related work on V2X HiL simulation

and identifies the need for simulation experiments in V2X development and performance evaluation.

Human-in-the-loop testing: This methodology involves the use of human testers to simulate real-world scenarios and test V2X communication and applications. This methodology allows for the evaluation of user experience and identification of potential issues that may arise in real-world scenarios. This is an effective approach for assessing the usability and effectiveness of V2X applications from the user's perspective. However, this method may not perform as fast as simulation testing because it needs human interaction.

The article (Olaverri-Monreal, ve diğerleri, 2018) discusses the need for evaluating in-vehicle applications that are based on V2X communication technologies under lab-controlled conditions before performing field tests. The article presents a driver-centric traffic simulator built over a 3D graphics engine that creates a driving situation as it communicates with a traffic simulator to simulate real-life traffic scenarios. The TraCI as a Service (TraaS) library was implemented to perform the interaction between the driver-controlled vehicle and the SUMO. The article also presents a use case to evaluate the platform, which is related to the efficiency of a Traffic Light Assistant (TLA). The simulation platform's performance is evaluated through this use case with traffic lights.

ViL testing: This methodology involves the use of a physical vehicle or a simulated vehicle to test V2X communication and applications. This methodology allows the testing of the performance and effectiveness of V2X applications in a real-world scenario. This is an effective approach for evaluating the interaction between a vehicle and a V2X system. However, it can be costly and complex to set up, and may not fully replicate the complexity of real-world conditions.

A study (Chen, ve diğerleri, 2020) discusses the challenges of testing autonomous vehicles and proposes a new testing approach called mixed test environment-based ViL validation. This approach combines virtual and real-world scenarios to create a more comprehensive and safe testing environment. This method allows for the simulation of complex traffic scenarios and the integration of real sensors and test fields. The virtual perceptional results are simulated directly and delivered to a real vehicle, thereby reducing resource consumption. The approach is configurable and reproducible with

OpenStreet Map-based (OSM) HD maps, enabling the simulation to be decoupled from specific test fields or traffic facilities. The proposed method is a drive-safety testing technique that can be used before actual road testing. This paper provides a diagram illustrating the framework of the proposed method, with a closed loop consisting of a real autonomous vehicle and a mixed test environment.

Field test: This methodology involves testing V2X communication and its applications in real-world conditions. This methodology allows the testing of V2X applications in a real-world scenario, providing accurate and reliable results. This is an effective approach to assess the real-world performance and effectiveness of V2X applications. However, it can be time-consuming, costly, and difficult to control the environment, leading to potentially inconsistent results.

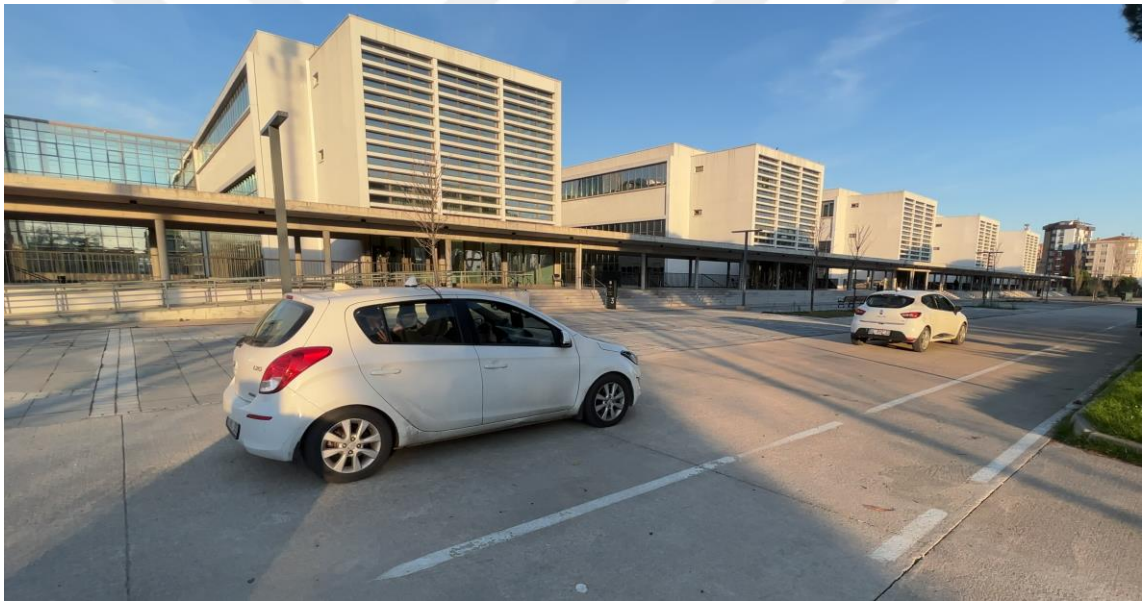

Figure 2.1. *Field testing of collision risk warning between two vehicles*

The article (Xu, ve diğerleri, 2021) discusses the testing process and results of the communication performance of V2V at an obstructed intersection in a closed test field. The focus is on the application requirements of Intersection Collision Warning (ICW) to V2X. The paper proposes a message forwarding mechanism based on vehicle-to-vehicle to infrastructure to vehicle (V2I2V), and the test results show that V2I2V performs better than V2V at an intersection with obstacles. The article also discusses the application scenarios of ICW and the requirements of 3GPP for road safety communication and warning. The article highlights the importance of connected cars in

reducing traffic accidents and improving the safety of intersections.

2.2. Differences of Our Test Platform

In the previous section, we discussed the current test methods of V2X communication and applications. Each method has its own advantages and disadvantages, and one or more of these methods can be applied depending on the requirements of the software or system to be tested.

In the proposed test platform, all vehicles and environmental components are created virtually, resembling simulation-based test environments. However, in simulation-based test environments, many structures are modeled simpler to reduce complexity or improve performance. For example, static predefined data can be used instead of modeling Global Positioning System (GPS) receivers and providers for feeding the location information of vehicles. To increase realism, such important components have also been modeled in our test platform. In this regard, it can be said that there is a transition from simulation to emulation. On the other hand, in simulation-based test environments, channel models are simulated, where inter-station communication is manipulated by calculating the paths that signals follow in the real world and the situations they encounter under different transmissions like interference, path loss, etc. Our test platform does not inherit this feature, so inter-station communication is carried out without any loss.

In HiL test methods, real OBU, RSU, or other C-ITS hardware are used. Since these devices will also be used in the real world, including these devices directly in the tests significantly increases the realism of the test. Our test platform does not use real devices, which is an important difference. Especially since OBUs and RSUs are quite expensive, it is not practical to set up such a testing environment. In addition, the use of these devices is not included in our study. Instead, our test platform performs the functionality that these devices provide in a containerized structure.

The ViL test method is more comprehensive than the HiL test method. Here, experiments are carried out by directly integrating a vehicle with the virtual

environment, rather than just using hardware. This method is more expensive than the HiL method and requires additional hardware knowledge. This method is also not included in our study. In our study, the vehicle dynamics were realized in a virtual manner.

The field test method is the most realistic test method and is closest to end-use. In the V2X context, it requires the use of all components (vehicle, OBU, RSU, and other hardware) as in reality. Naturally, this is the most expensive method. Because repeatability and low cost are important, this method was also not included in our study.

In summary, our test platform can be considered as a harmonized test platform that integrates the specific features of other test methodologies. It can be categorized as a human-in-the-loop test platform because it relies on human interaction to control vehicles.

3. METHODOLOGY

Although field test is the most reliable method for the verification and validation process of V2X applications, it is not efficient in terms of cost, repeatability, scope, and time. Therefore, a significant portion of the tests must be conducted in a virtual environment.

As mentioned in the previous sections, there are many different virtual testing methods available. These methods have their own advantages and disadvantages. When examining existing solutions, no test platform was found that actively involved users in the role of the driver during testing for V2X applications. User involvement plays a critical role in identifying scenarios that may be overlooked during static operation.

Based on these known facts, verifying the developed V2X applications by allowing users to participate in the testing process and control their vehicles in a virtual environment will greatly increase the test scope. Such a test platform should have four fundamental components, as shown in **Figure 3.1**. Firstly, a controller (hardware and software combination) that allows the user to control his/her vehicle, secondly, a vehicle dynamics software that translates the inputs created for vehicle control into meaningful mobility actions, thirdly, a software that monitors the dynamics of vehicles and performs V2X communication with these data, and finally, some interfaces that visualize the dynamic operation environment generated by this entire test suite for the user.

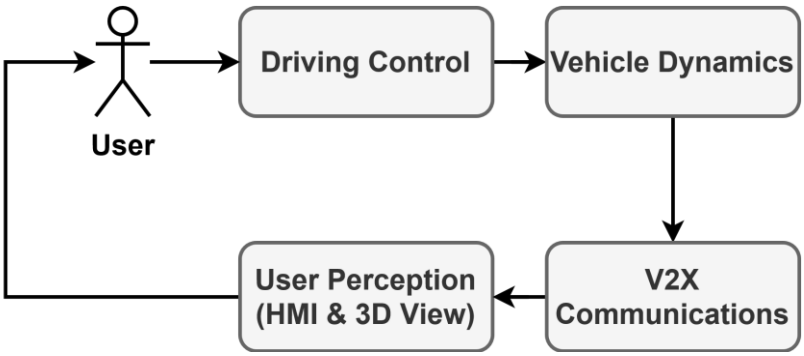

***Figure 3.1**: High level goal of required system*

Each of these fundamental features requires a comprehensive implementation when

included in the test platform, as they contain a lot of details on their own. At least one module is required to fulfill each feature. These modules can be developed using simple methods when considered independently from V2X communication and the real world. However, transitioning to field tests can yield significantly different results compared to virtual environment testing. For example, it is known that a GPS receiver operates by converting satellite signals into periodic NMEA sentences. If, instead of implementing a similar approach in the test platform, direct feeding of some predefined positions over time is preferred, it can lead to the omission of many values within the NMEA sentences, including confidence. This can result in issues with data that lacks sufficient confidence in the real world. Therefore, it is essential to consider the real-world counterparts of these modules and design them as closely as possible to ensure accurate representation.

With the knowledge of those needs, this study proposes a test platform with a highly modular structure, aiming to align with software and hardware components used in the real world. Transforming this design into software entails considerable effort, therefore some modules were implemented using existing libraries, while others were developed by us. As a result, while Section 3.1 discusses the modules we developed ourselves, which form the foundation of the platform, Section 3.2 provides information about the open-source tools we utilized.

3.1. System Architecture

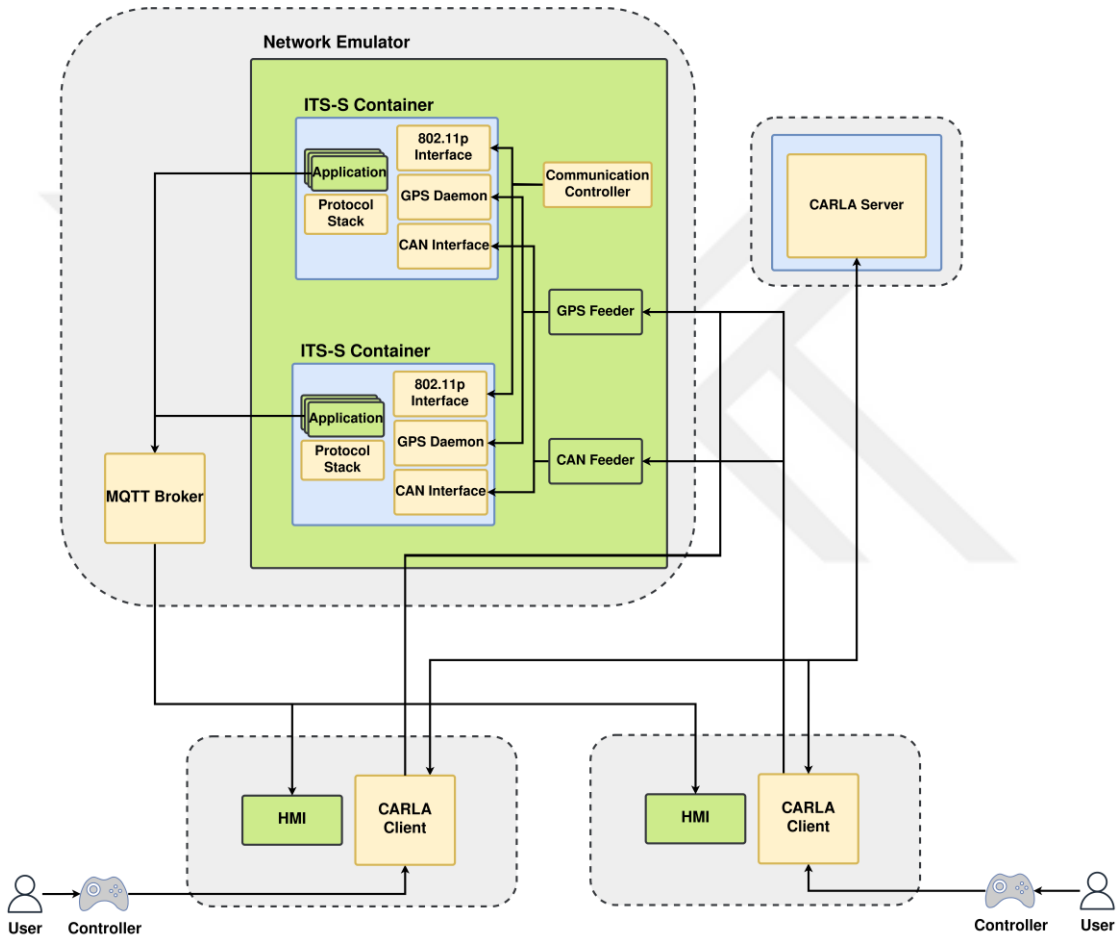

Figure 3.2. *The high-level architecture of the system which two users included.*

The proposed framework is composed of several components as illustrated in **Figure 3.2**. This figure depicts a scenario in which two users are included in the framework.

The grey background areas represent independent computers. In the presented scenario, four different computers are connected via a local network, which allows different software applications located on different computers to easily exchange data. While the figure represents the ideal situation, having exactly four computers is not mandatory.

Some components can be merged into a single computer based on their performances. The blue components in the figure are Docker containers, which are used in the test platform for environment isolation and deployment facilitation purposes. The yellow components shown in the figure are open-source software components that were used directly or with minor modifications. The green components represent those developed for this study.

The entire platform was designed to operate on a Linux infrastructure. The development and testing processes were performed using the Ubuntu 22.04.

3.1.1. User

The user refers to the person using the platform by controlling the vehicle model in the simulator based on their perception. In the real world, the user corresponds to the driver of the vehicle.

3.1.2. Controller

A controller is a device that contains units such as buttons that can be interacted with and connected to the computer -mostly- via USB. As shown in **Figure 3.2**, the controller can be a gamepad, but also a keyboard, steering wheel, etc.

Each action generated from the controller is expressed with certain numerical values in the computer. This ensures that each button on the controller can be assigned a specific function. In the proposed framework, the controller corresponds to all mechanical or electronic components with which the driver can interact with the vehicle in the real world. For example, while a button on the controller works like a throttle pedal, another button can control the turn signals of the vehicle. Almost all buttons in the controller can be mapped to an actuator of a real vehicle. If this controller is a steering wheel, mapping can be performed more realistically and easily.

In the real world, the controller corresponds to all the electronic and mechanical equipment of the vehicle, which allows the driver to perform driving actions. For simplicity, only the throttle, brake, and steering functionalities were implemented in this study.

3.1.3. CARLA Server

As described in Section 3.2.1, CARLA has a client-server architecture. The server side initializes the simulation nodes, controls their interactions, and owns the nodes' life cycles.

Our test platform relies on the CARLA simulator to generate vehicles in the scenario. CARLA server is initialized with the given number of vehicles and specifications. Also, any custom map with "xodr" extension can be used for environment generation.

The CARLA server is deployed on a computer with a high-performance GPU for a smooth simulation run. It runs inside a Docker container to simplify and isolate the server-deployment process. The Docker image is provided by CARLA's development team. After invoking the server process, it is possible to access the CARLA server process by creating a local network of computers.

3.1.4. CARLA Client

The implementation of the CARLA client was achieved through the utilization of CARLA's Python API, thereby producing a Python script that establishes a connection with the CARLA server. The client can be executed on a distinct computer from the one hosting the server, although the latter must be accessible to the former. The server has the capacity to support multiple clients, subject to the performance of its host computer.

Upon initialization, the client generates a 3D simulation environment display using the pygame library, which depicts the viewpoint of the vehicle similarly to that of a typical racing game.

Figure 3.3. *An example CARLA Client UI.*

The client performs the processing of controller inputs, converting user-generated inputs into meaningful data. Upon pressing the throttle button, for example, the client obtains the corresponding numerical value and directs the server to accelerate the relevant vehicle. The processing of other inputs follows a similar process.

Commands executed by the CARLA client cause the state of the vehicle to change (such as accelerating, decelerating, and turning). This dynamic state is transmitted instantaneously to the GPS Feeder and CAN Feeder. In this manner, vehicle software replications (ITS-S container) created in Network Emulator work with the most up-to-date data.

The principal objective of the client is to enable the user to control their vehicle in a manner similar to real-world scenarios.

3.1.5. Protocol Stack

Protocol stacks are defined for the communication of connected vehicles and the management of services that work on top of them. These protocol stacks are developed and made available for use based on the regulations and infrastructure technologies in the country where they will be used. Therefore, many definitions of protocol stacks have emerged at the continental and national levels. Examples of these include ETSI

ITS-G5 and IEEE WAVE. In this study, the ITS-G5 protocol stack, which is mainly used by European Union countries, has been preferred.

Implementing a protocol stack from scratch with all of its components is a cumbersome task. Therefore, and because it is not within the scope of this study, it is necessary to use a ready-made library as the protocol stack. For this purpose, an open-source ITS-G5 library called Vanetza was preferred.

A detailed description and features of Vanetza are provided in Section 3.2.6. The library has the necessary base definitions for creating and sending various message types defined by ETSI (e.g., CAM[1], DENM[2]), as well as GN and BTP implementations.

Cooperative Awareness (CA) and Decentralized Environmental Notification (DEN) services located in the facilities layer of the ITS-G5 architecture are defined in ETSI EN 302 637-2 (European Telecommunications Standards Institute, 2014) and ETSI EN 302 637-3 (European Telecommunications Standards Institute, 2014) standards, respectively. These services control the sending and receiving operations of CAM and DENM packets, respectively. These services are not available as ready-made services in Vanetza. Therefore, the basic versions of these services were also developed within the scope of this study.

CA Service is an essential component of the ITS-G5 protocol stack and is responsible for exchanging basic safety messages between vehicles, infrastructure, and other connected entities. CAMs contain essential information about the current status of a vehicle, such as its position, speed, and acceleration. The CA service is critical for enabling connected vehicles to exchange information in real time, allowing them to detect potential hazards and take appropriate actions to avoid collisions. CAMs are transmitted frequently and rapidly to ensure that all connected vehicles are aware of their surroundings, and they can be used by a wide range of ITS applications, including collision avoidance, traffic management, and cooperative driving.

DEN Service is another crucial component of the ITS-G5 protocol stack and is

[1] The type of message published by vehicles (or other ITS-Ss) to create environmental awareness by sharing location, speed, direction, and many other status data.

[2] The type of message that ITS-Ss broadcast to notify short or long-term environmental events (such as roadwork warning, emergency vehicle approach warning).

responsible for transmitting environmental information to connected vehicles. DENMs contain information about potential hazards or other environmental conditions that could affect vehicle safety, such as roadwork, accidents, or adverse weather conditions. The DEN service enables connected vehicles to receive this information in real-time, allowing them to adjust their driving behavior accordingly. The DEN service is particularly useful for improving road safety and traffic efficiency because it allows connected vehicles to react quickly to changing conditions and avoid potential hazards.

3.1.6. 802.11p Interface

All stations (vehicles) had an 802.11p interface for transmitting and receiving messages. These interfaces are created by Communication Controller. The messages specified in Section 3.1.5 are transmitted and received through this interface.

3.1.7. GPS Daemon

The GPS Daemon is an integral component that is facilitated by the operation of a *gpsd* process (daemon) in the background. During the initialization stage of the Docker container, a *gpsd* process is initiated to enable the reception of NMEA sentences. By means of a UDP socket created by the *gpsd* process, it becomes possible to transmit NMEA sentences to any other process within the system, thereby enabling the integration of the GPS Daemon with other processes that require access to location data.

3.1.8. CAN Interface

The ITS-S incorporates a CAN Interface which is created by the CAN Feeder. The creation of the interface is facilitated through the utilization of the built-in CAN interface definitions of the Linux kernel. The primary function of this interface is to receive CAN frames transmitted by the CAN Feeder. Subsequently, related services or applications can access CAN frames through this interface.

In practice, the data format of the CAN frame is determined by the respective vehicle manufacturer. Consequently, various vehicle brands or models can utilize different formats. Therefore, the data format of the CAN frames in our ITS-Ss is configured in accordance with the specific requirements of our system.

To ensure isolation, the CAN interface for each ITS-S is moved into a distinct network namespace. Subsequently, these network namespaces are relocated to the corresponding Docker container for the ITS-S.

3.1.9. Communication Controller

In the realm of vehicular communication emulation, the task of generating 802.11p interfaces using the mac80211_hwsim library falls under the responsibility of the Communication Controller. This involves invoking the appropriate functions of the mac80211_hwsim library to provide an interface for each station. In addition, an extra interface is created to enable vehicular communication monitoring in the host computer, which runs the Network Emulator.

To ensure isolation, the interfaces for the stations are sorted into distinct network namespaces. Subsequently, these network namespaces are relocated to the corresponding Docker container for the ITS-S.

3.1.10. GPS Feeder

The GPS Feeder is responsible for controlling GPS-based information, such as the location and direction received from the CARLA client.

During the initialization stage, the GPS Feeder creates a GPS agent for each ITS-S. These GPS agents can be considered as virtual GPS receivers. A UDP socket is created to communicate with the *gpsd* service created within each ITS-S container using these GPS agents. Using this method, the *gpsd* services provide the most up-to-date GPS data to their respective ITS-S containers.

The GPS Feeder was developed by us along with several colleagues and was published as a separate scientific study (Senkus, Yaman, Aydin, & Soyturk, 2022). Additional improvements were made to utilize it in the test platform.

3.1.11. CAN Feeder

In the initialization stage, the CAN Feeder generates CAN interfaces for each ITS-S and moves them into the corresponding ITS-S Docker containers using separate network namespace.

During runtime, the CAN Feeder receives real-time status updates for all vehicles currently running on the CARLA test platform via the CARLA Client. It then sends the data for each car to the corresponding CAN Interface, based on the received status information. This enables the ITS-S container to utilize the car's up-to-date data.

3.1.12. Applications

V2X applications are software programs that run above the Protocol Stack, and their primary purpose is to exchange information between vehicles and the infrastructure. Information exchange is essential for providing safety and convenience services in V2X environments. Various V2X applications are defined in ETSI standards.

Security applications are among the most important types of V2X applications. These applications ensure that the information exchanged between vehicles and infrastructure is secure and protected from cyber-attacks. Safety applications are another critical type of V2X applications that aim to reduce the number of accidents and fatalities on roads. They provide warnings to drivers about potential hazards such as collisions or hazardous road conditions.

Other V2X applications include traffic efficiency applications, which aim to improve the traffic flow and reduce congestion. These applications provide real-time traffic information to drivers, allowing them to select the best route to their destination. Additionally, infotainment applications provide drivers and passengers with entertainment and other useful information during their journeys.

V2X applications receive data from the Protocol Stack and can generate outputs that can be sent to an MQTT Broker or used internally within the ITS-S Container. Using the test platform, any V2X application defined in the ETSI standards can be implemented and tested to ensure that they function as intended in real-world V2X environments.

3.1.13. MQTT Broker

The MQTT Broker is an essential component of the V2X test platform, which serves as an intermediary between the ITS-S containers and the external software that needs to access the data generated by the applications. The Mosquitto package is used as the MQTT Broker, which receives the output data generated by V2X applications running

in ITS-S containers.

The MQTT Broker is responsible for receiving data from various applications and sharing them with the external software. The applications send their output data to the MQTT Broker, which forwards it to any external software that subscribes to the respective topics. This design allows the V2X test platform to share the output data generated by the applications with multiple external software components.

We selected the MQTT protocol because of its ability to receive information from multiple ITS-S applications from a single source. This ensures that the data generated by different applications can be consolidated and easily shared with external software as required.

3.1.14. HMI

In the real world, almost every new-generation vehicle has a screen that displays the status of the vehicle and allows control of multimedia operations. Although these screens are rarely found on the windshield of the vehicle, they are usually located on the front dashboard next to the steering wheel, as shown in **Figure 3.4**.

The functions of these screens vary significantly from vehicle to vehicle. The variety of these functions is directly proportional to the electronic and software capacity of the vehicle. Moreover, connected vehicles are capable of visualizing received and transmitted information through V2X communications.

Figure 3.4. *An example HMI (a car dashboard with a gps device photo, 2022)*

A simple HMI (**Figure 3.5**) was implemented in this study. This HMI has the features that serve the purpose of the basic scenarios to be shown in this study.

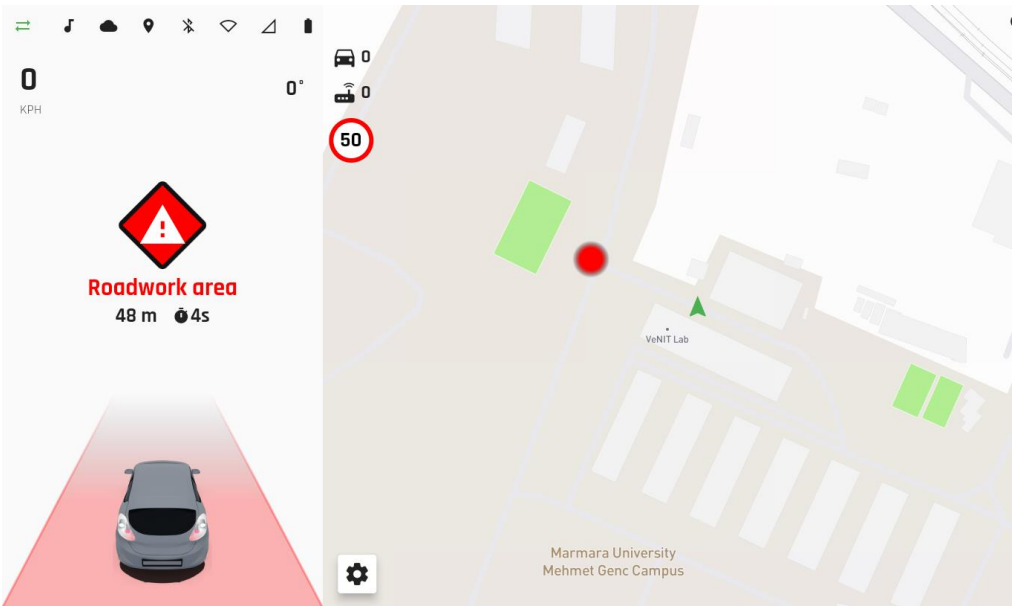

Figure 3.5. *A screenshot of HMI was developed for this study.*

HMI application has been developed by using the Flutter framework (Google, 2017). Flutter is an open-source mobile UI framework that is available at no cost. It was

developed by Google and made its debut in May of 2017. With Flutter, it is possible to build a native mobile application using only one code base.

3.2. Packages Used

For the test platform proposed in this study, a variety of diverse components must be created and combined. However, attempting to create all of these components in-house during this study would detract from its intended goals and necessitate additional resources. Consequently, certain modules have been selected by identifying the software that is commonly employed within the software industry and academia.

3.2.1. CARLA

CARLA (Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017) is a simulation platform that has been designed to facilitate the development, training, and testing of autonomous driving systems. Its open-source code and protocols, combined with open digital assets such as urban layouts, buildings, and vehicles, are specifically tailored to support these endeavors, and can be utilized without any restrictions. The platform also offers various features such as the ability to specify sensor suites, control both static and dynamic actors, generate maps, and customize environmental conditions as per user requirements.

The CARLA simulator utilizes a client-server architecture that can be scaled to meet varying requirements. The server is responsible for all aspects of the simulation, such as sensor rendering, physics computation, and updates to the world state and its actors. To achieve realistic results, it is recommended to use a dedicated Graphics Processing Unit (GPU) when running the server, especially when working with machine learning. On the other hand, the client side is composed of various client modules that control the logic of the actors on the scene and set the world's conditions. This is accomplished using the constantly evolving CARLA API, which is available in Python or C++. The API serves as a mediator between the server and client and offers new functionalities as it progresses.

3.2.2. Docker

Docker (Merkel, 2014) is an open-source platform that facilitates the development,

shipping, and deployment of applications. This enables users to detach their applications from their infrastructure, which allows for quicker software delivery. By leveraging Docker's techniques for swift shipping, testing, and deploying code, users can similarly manage their infrastructure as they manage their applications. This approach helps to significantly reduce the time between writing the code and running it during production.

3.2.3. mac80211_hwsim

mac80211_hwsim (Malinen, 2008) is a module in the Linux kernel that enables simulation of an arbitrary number of IEEE 802.11 radios for mac80211. It allows for the comprehensive testing of mac80211 functionality and user space tools (such as wpa_supplicant and hostapd) that closely mimic the standard use of WLAN hardware. Mac80211_hwsim can be viewed as another hardware driver by mac80211, which means that there is no need to make any changes to mac80211 to utilize this testing tool.

3.2.4. Mosquitto

Mosquitto is a message broker that employs the Message Queuing Telemetry Transport (MQTT) protocol, a lightweight messaging standard created for low-bandwidth and high-latency devices. Through a publish-subscribe model, Mosquitto allows networked devices to exchange messages while providing features, such as message persistence, quality of service, and security. The software is open-source and licensed under the Eclipse Public License.

3.2.5. gpsd

gpsd (gpsd — a GPS service daemon, 1995) is a service daemon that runs in the background and monitors one or more GPSes or AIS receivers that are connected to a host computer via serial or Universal Serial Bus (USB) ports. The daemon makes all data related to the location, course, and velocity of the sensors available for querying on Transmission Control Protocol (TCP) port 2947 of the host computer. One of the significant benefits of using *gpsd* is that it enables multiple location-aware client applications to access supported sensors simultaneously without data loss or contention. Additionally, *gpsd* provides a format that is more accessible to parse than the National Marine Electronics Association (NMEA) 0183, which is commonly used by most

GPSes. The *gpsd* package includes a linkable C service library, a C++ wrapper class, and a Python module that developers can use to communicate with *gpsd* when building *gpsd*-aware applications.

3.2.6. Vanetza

Vanetza (Riebl, Vanetza - Your open-source ETSI C-ITS protocol stack, 2013) is an open-source software implementation of the ETSI C-ITS protocol suite. Its implementation includes various protocols and features, such as GeoNetworking (GN), Basic Transport Protocol (BTP), Decentralized Congestion Control (DCC), Security, and support for ASN.1 messages (Facilities) such as Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM). Although it was originally developed to work on ITS-G5 channels in a Vehicular Ad Hoc Network (VANET) using IEEE 802.11p, Vanetza and its components can also be integrated with other communication technologies, such as GN over IP multicast.

4. EXPERIMENTAL EVALUATION

The test platform developed in this study enables the testing of V2X protocol stacks and applications to be developed for connected vehicles. Unlike static test frameworks, the platform aims to expand the scope of testing of the developed software and increase repeatability by allowing users to control virtual vehicles created in real time, which closely resembles reality. This reduces the need for real-world field testing.

To demonstrate the reliability and performance of the platform, firstly, some performance evaluation experiments needed to be done. These experiments aimed to show that the modules of the test platform communicate and operate with acceptable delays in terms of user perception.

In order to show the feasibility of V2X application tests, which is the main purpose of the platform, as well as the measurement of delay times, as an example, the Longitudinal Collision Risk Warning (LCRW) application was developed and run on the platform. There are many different applications that organizations such as ETSI, SAE, C2C-CC standardize or define as recommendations. These applications are divided into different categories according to their priorities and ease of implementation. In this context, applications for the solution of emergency and vital situations in traffic are called safety applications. Since LCRW is an application with high priority, it was preferred for the demonstration of this study.

In the following sections, the details of the experiments on performance evaluation and LCRW application were mentioned, and the results of the measurements are given.

4.1. Performance Evaluation

One of the critical requirements for the developed platform to be usable is to work with smoothness that is close to the real world. When a driver presses the gas pedal in the real world, the resulting acceleration in the vehicle occurs instantaneously. Similarly, the movement of a vehicle can be perceived immediately by other drivers in the surrounding area. The test platform should also meet the user perception as close to reality as possible in such actions.

The platform consists of many modules and subcomponents. As mentioned in the

previous sections, while some modules were developed by us, open source or free tools were used for some modules. In order to make performance measurements comprehensively and accurately, it was important to which category the module used was included. The timestamp logging method was used to calculate the delay values in the performance measurement. For this reason, although the measurements were made as desired in our own modules, it was not so simple to measure the delay values in 3rd party tools. This issue was experienced specifically between the CARLA client and server. Details about the measurements were given in the next sections.

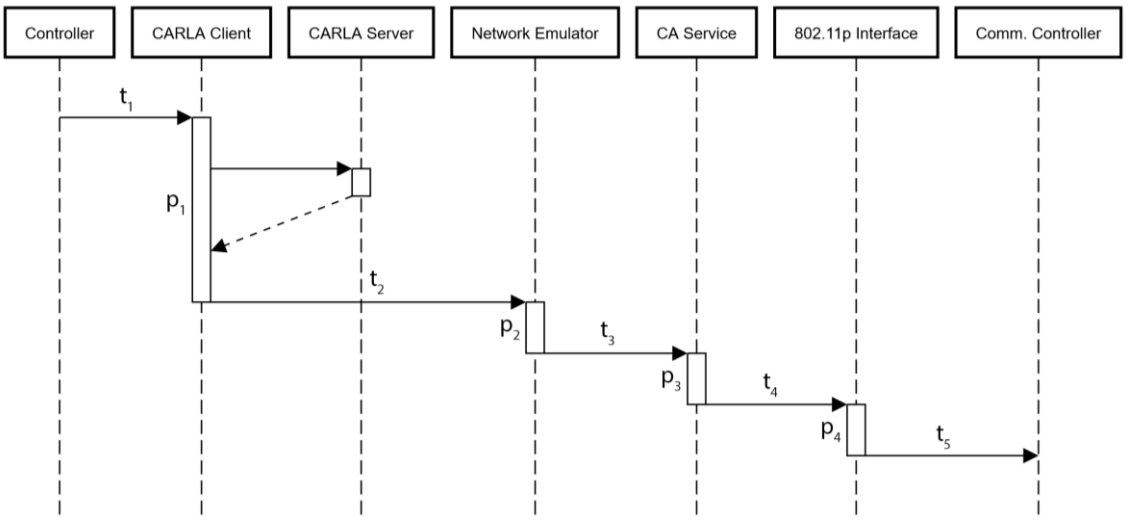

Figure 4.1: *Sequence diagram of control of a single vehicle*

In the sequence diagram in **Figure 4.1**, it is shown through which basic modules an input that occurs while the user is controlling his/her vehicle in the platform. Although there are many more internal modules compared to high level architecture, the measurements were made on the main modules. In the diagram, the times prefixed with ***t*** represent the transmission delays between the two modules, while the times prefixed with ***p*** represent the processing delays.

While controlling the vehicle, the user sends various commands with the buttons on the controller. These commands are read by the CARLA client. The elapsed time is called t_1. CARLA client works asynchronously with a periodic interval. It checks the pressed keys of the controller at approximately 16 ms intervals (to work with an average of 60 FPS). Theoretically, t_1 can be at most 16 ms. Measuring the exact time here is not very easy as it is difficult to find the exact moment when the keys are pressed. For this

reason, time t_1 was ignored in the measurements. However, 16 ms can be considered as the maximum theoretical threshold. Although this value varies according to the hardware specifications of the computer that the server and the client is running, it has not been seen that it exceeds 16 milliseconds at the scale (number of vehicles) our platform operates.

CARLA client and CARLA server work by communicating with each other. These two modules are asynchronous, that is, they have independent lifecycles. The CARLA client changes various dynamics of the vehicle such as gas, brake. These changes are read asynchronously by the server and the state of the vehicle (such as speed, acceleration, direction) changes. These changes are read back by the client and the latest status of the vehicles are obtained. Measuring the elapsed time (p_1) during these processes is also not very possible due to CARLA's asynchronous operation, because it is difficult to understand what input the state change on the vehicle occurs due to. For this reason, the p_1 value was excluded from the measurement scope.

Network Emulator has many submodules. Measuring the time elapsed in the use of these modules separately has not been consciously preferred, both in order not to increase the detail unnecessarily and because it would be sufficient to accept the Network Emulator as a main module. Therefore, the status information from the CARLA client directly reaches the Network Emulator. Here the elapsed time is called t_2. This time could be measured by the logging method.

After the Network Emulator receives the vehicle status changes, it processes these data and updates the sub-modules (such as GPS Feeder, CAN Feeder) within itself. The time elapsed during the updating of these modules is called p_2. This time was measured by timestamp logging.

Each of the vehicles run on the platform activates a CA service. This service was implemented according to the basic specifications given in the ETSI EN 302 637-2. Normally, the CA service determines the CAM broadcast frequency according to the changes in the variables such as speed, position and heading in the vehicle (considering the 100 ms lower and 1 second upper limits). Therefore, varying times may occur between each CAM packet. To minimize the differences during the measurements, the CAM broadcasting frequency was determined as static 10 Hz (100 ms interval).

Besides, GPS Feeder and the positioning service in Protocol Stack works in a scheduled manner. Therefore, gathering up-to-date data by CA Service takes a relatively long time t_3.

CA service constructs a CAM packet to send it to virtual interface for broadcasting. In this process the service gathers necessary data (position, speed, station type etc.) from related modules. The elapsed time from start of the packet generation and to forwarding it to the virtual interface was named as p_3.

The CAM packet produced by the CA service is sent to the virtual 802.11p interface in each ITS-S container. The transmission delay that emerged in this process was named as t_4. This delay was measured using packet capturing of related interface.

After a CAM packet is dropped on the 802.11p interface, the processing time of this packet on the interface and sending it to the Communication Controller are named p_4 and t_5, respectively. In the measurements made for these two operations, very low results were revealed in the unit of microseconds. Therefore, p_4 and and t_5 were accepted as 0.

It has been explained in the previous paragraphs which of the mentioned processing and transmission delays are measured and how they are measured. In summary, t_2, p_2, t_3, p_3, t_4 values were measured clearly. A theoretical value of 16 ms is accepted for t_1. No clear data were obtained for p_1. Since p_4 and t_5 values are in the order of microseconds in the measurements made, they have little or no effect on the platform. Therefore, these values were accepted as 0.

Table 4.1. *Hardware specifications of the computer*

CPU	Intel Core i7-9750H
GPU	Nvidia GTX 1650
RAM	16 GB
Operating System	Ubuntu 22.04 LTS

A single computer, whose specifications were shown in **Table 4.1**, was used for latency measurements. This computer can be considered a low-spec computer as of 2023. As

mentioned in the previous sections, some of the components on the platform could be deployed to run on different computers. The reason for using a single computer for measurements was to eliminate system time differences that may occur between computers. Also, since running all modules on a single computer would result more overhead on the system, measurements would give worse results compared to distributed work. This was deliberately chosen to show the worst-case.

The developed platform provides a multi-user V2X verification environment. By multi-user, we do not mean on a large scale, but rather a few users. Therefore, measurements were conducted in two scenarios involving 2 and 3 vehicles. In the 2-vehicle scenario, each run lasted approximately one minute and was repeated twice. Similarly, the 3-vehicle scenario was also run approximately twice, each lasting about a minute. In summary, a total of 4 minutes were spent running the scenarios: 2 minutes with 2 vehicles, 2 minutes with 3 vehicles, and the overall total. Although these durations may seem short, they resulted in the inclusion of over 60 thousand measurements in total. Furthermore, shortly after the scenarios started, the measurements stabilized within a few seconds and continued with a more normal distribution. Naturally, if the scenario duration were extended, values close to the existing measurements would be obtained.

The measurements of those 4 runs were given below using tables and charts.

2 vehicles – First Run:

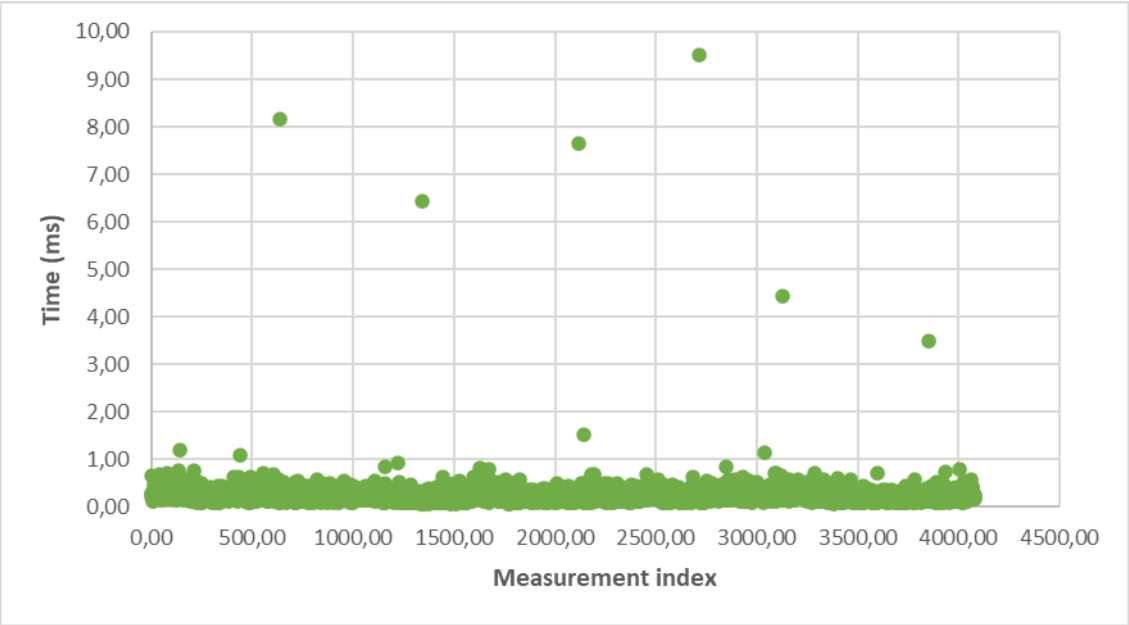

Figure 4.2: *t_2 histogram for the first run with 2 vehicles*

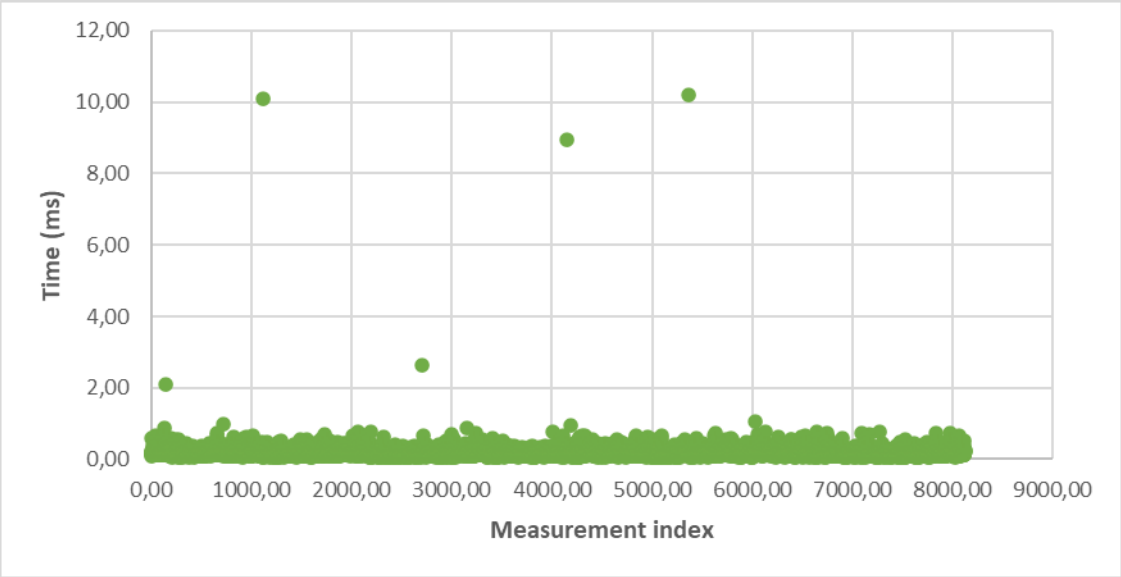

Figure 4.3: *p_2 histogram for the first run with 2 vehicles*

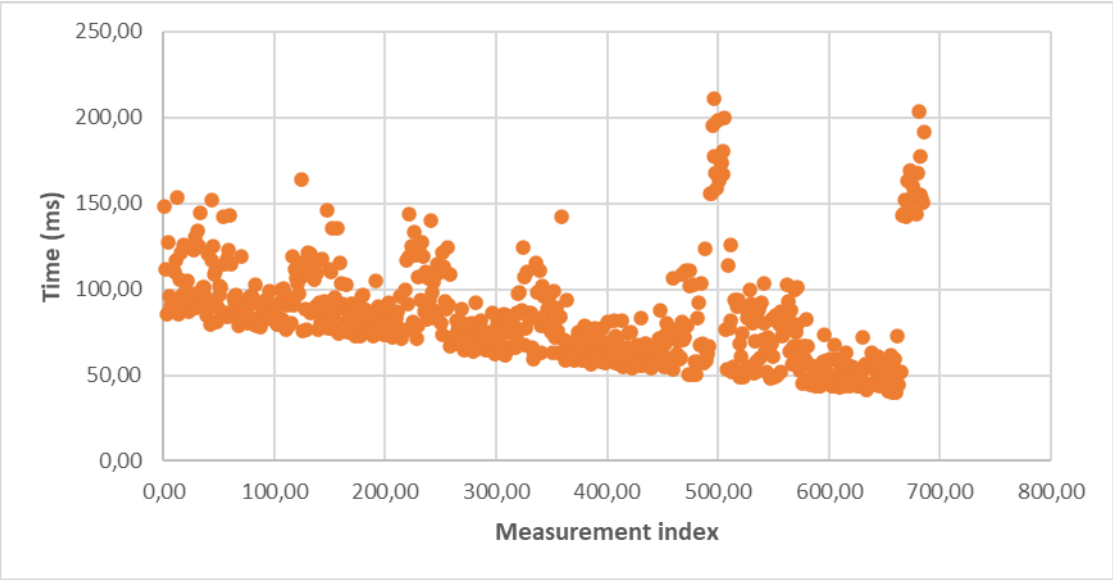

Figure 4.4: *t_3 histogram for the first run with 2 vehicles*

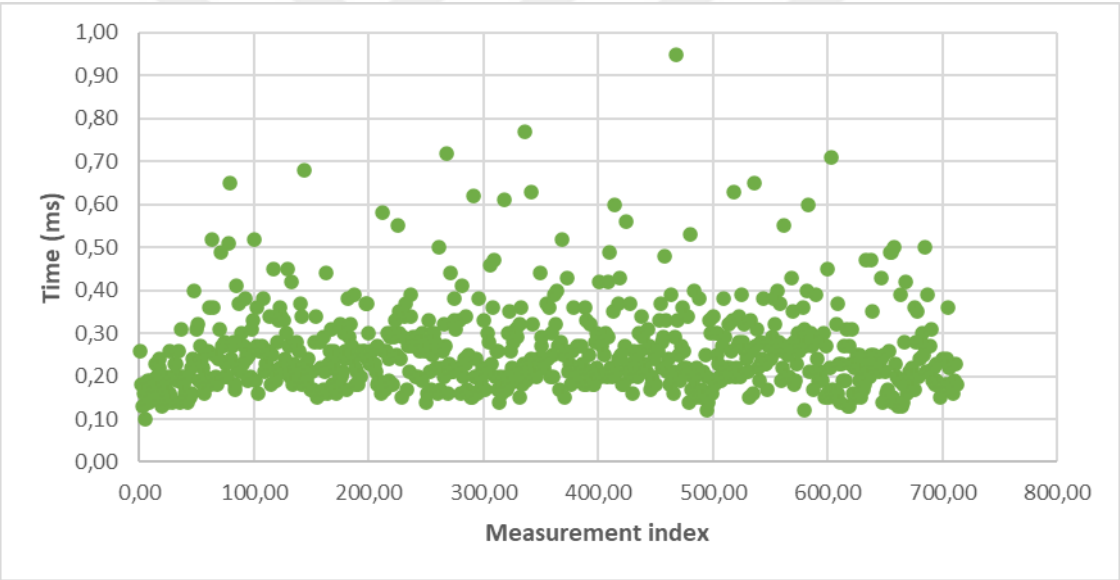

Figure 4.5: *p_3 histogram for the first run with 2 vehicles*

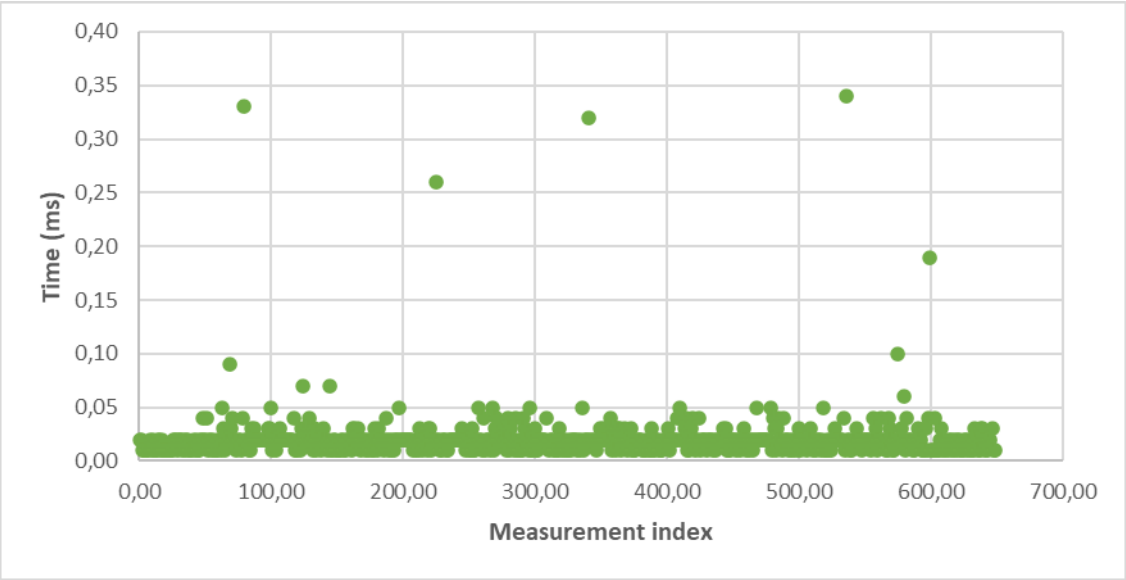

***Figure 4.6**: t_4 histogram for the first run with 2 vehicles*

The measurements for t_2, p_2, t_3, p_3, and t_4 were shown in **Figure 4.2**, **Figure 4.3**, **Figure 4.4**, **Figure 4.5**, **Figure 4.6**, respectively. All of the measurements indicate raw values with outliers included. The green measurements are actual delays that could be measured sequentially during normal runtime. The orange (t_3) measurements are results of sum of 3 periodic schedules. By looking the change trend in **Figure 4.4**, the effect of asynchronous schedules can be seen from a repeating pattern. t_2, p_2 and t_4 has stable trends except outliers. In contrast, the values of p_3 were scattered more. This is because the contents of the CAM packets are not fixed. Each individual CAM packet may contain different data from each other, naturally of a different size. This is a factor that affects the creation time of a CAM packet.

***Table 4.2:** Statistics of the first run with 2 vehicles*

	t_1	p_1	t_2	p_2	t_3	p_3	t_4	p_4	t_5	***TOTAL***
MIN	16	-	0,07	0,05	39,74	0,10	0,01	0,00	0,00	*39,97*
MAX	16	-	1,08	0,78	210,81	0,56	0,10	0,00	0,00	*213,33*
AVG	16	-	0,23	0,18	83,80	0,25	0,02	0,00	0,00	*84,48*
STD. DEV.	0	-	0,10	0,08	29,01	0,08	0,01	0,00	0,00	

The statistics for the measurements of the first run with 2 vehicles were presented in **Table 4.2**. The values in table were given by removing outliers. In the first run, a total of 84.48 ms was observed as the sum of the average values of all durations. There is a high difference between minimum and maximum values. By looking to the t3 column, it can be seen that almost all of the difference comes from that value. Therefore,

2 vehicles – Second Run:

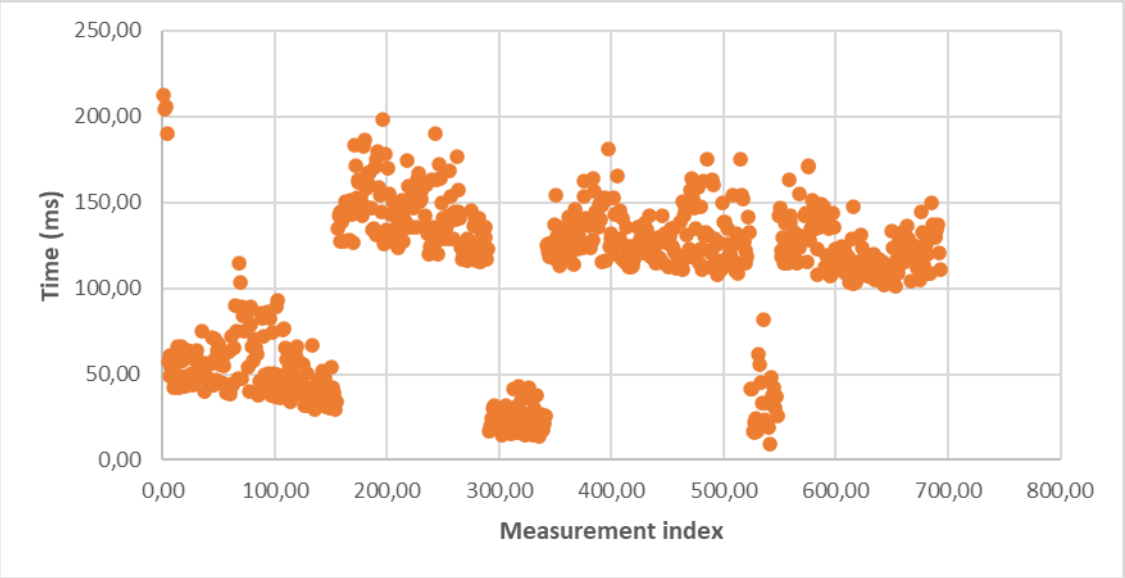

***Figure 4.7**: t_3 histogram for the second run with 2 vehicles*

Since the chart trends of individual time measurements were similar to first run, only the chart of t_3 was given in **Figure 4.7**. Asynchronous runtime pattern can be seen as in the first run.

***Table 4.3:** Statistics of the second run with 2 vehicles*

	t_1	p_1	t_2	p_2	t_3	p_3	t_4	p_4	t_5	***TOTAL***
MIN	16	-	0,07	0,04	9,58	0,11	0,01	0,00	0,00	*9,81*
MAX	16	-	1,20	0,85	212,45	0,87	0,10	0,00	0,00	*215,47*
AVG	16	-	0,22	0,16	105,51	0,25	0,02	0,00	0,00	*106,16*
STD.	0	-	0,10	0,08	41,26	0,10	0,01	0,00	0,00	

DEV.

The statistics of the measurements for the second run with 2 vehicles were presented in **Table 4.3**. In the second run, a total of 106.16 ms was observed as the sum of the average values of all durations. By comparing with the first run, all the statistics except t_3 are almost same.

2 vehicles – Comparison:

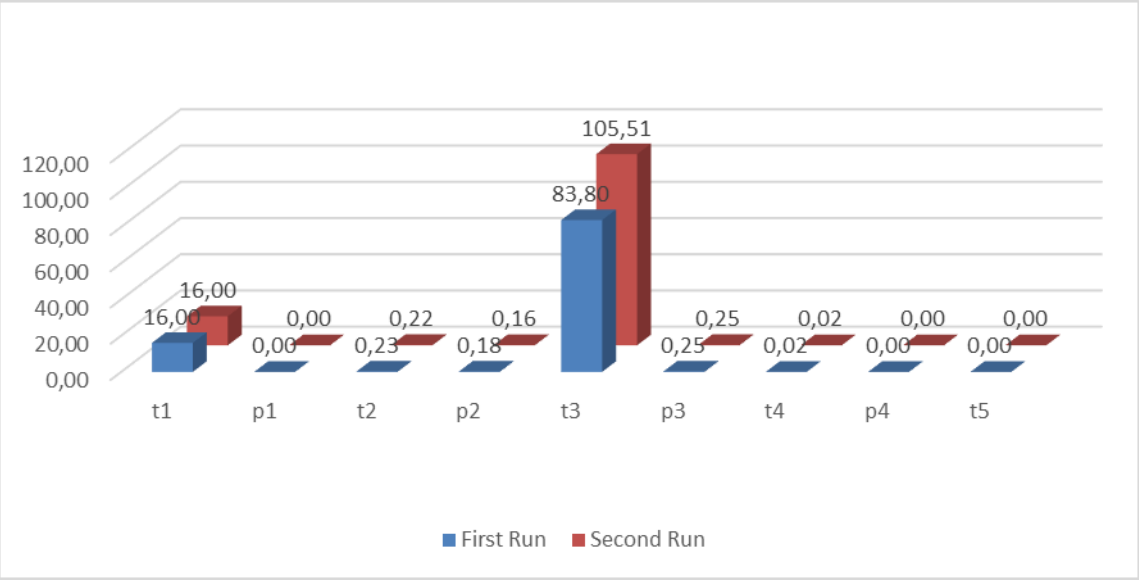

Figure 4.8*: Comparison of first and second run with 2 vehicles*

The comparison of statistics of both runs with 2 vehicles was given in **Figure 4.8**. All values except t_3 are similar to each other as expected. The reason for t_3 being significantly higher and different from the others is the presence of 3 different periodic operating mechanisms within it. Firstly, the GPS Feeder behaves like a real GPS provider and sends the current data it holds to the GPS Receiver every 100 milliseconds. Secondly, the service within the Protocol Stack collects up-to-date data related to location and time at intervals of 50 milliseconds. Finally, the CA Service generates CAM packets at intervals of 100 milliseconds. This system, consisting of 3 asynchronous components, was developed in order to be equivalent to real-life operation logic. Theoretically, a delay of up to 250 milliseconds is normal in this section. As can be seen in both tables, t_3 never exceeds the theoretical threshold of 250 milliseconds. If the duration of the runs were longer, the average values of t_3 could be

closer to each other.

3 vehicles – First Run:

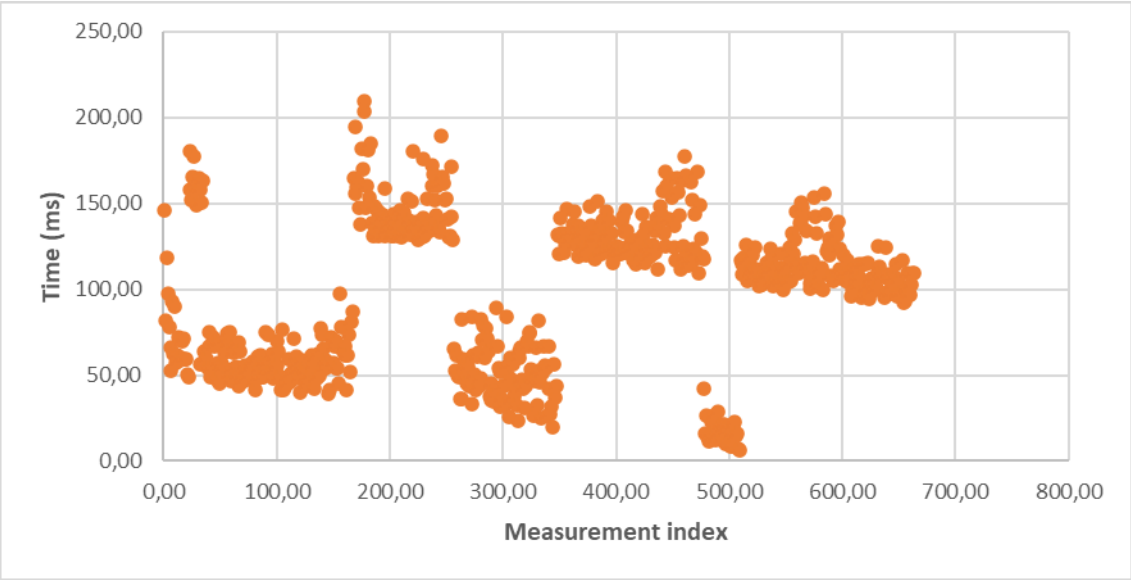

***Figure 4.9**: t_3 histogram for the first run with 3 vehicles*

Individual time measurements were like 2-vehicle case, so, only the chart of t_3 was given in **Figure 4.9**. Asynchronous runtime pattern can be seen as in the first run within the 6.41-212.70 ms range.

***Table 4.4:** Statistics of the first run with 3 vehicles*

	t_1	p_1	t_2	p_2	t_3	p_3	t_4	p_4	t_5	TOTAL
MIN	16	-	0,07	0,06	6,20	0,07	0,01	0,00	0,00	*6,41*
MAX	16	-	0,94	0,87	209,23	0,64	0,19	0,00	0,00	*211,87*
AVG	16	-	0,23	0,19	96,58	0,25	0,02	0,00	0,00	*97,27*
STD. DEV.	0	-	0,12	0,10	42,98	0,09	0,01	0,00	0,00	

The statistics of the measurements for the first run with 3 vehicles were presented in

Table 4.4. In the first run, a total of 97.27 ms was observed as the sum of the average values of all durations.

3 vehicles – Second Run:

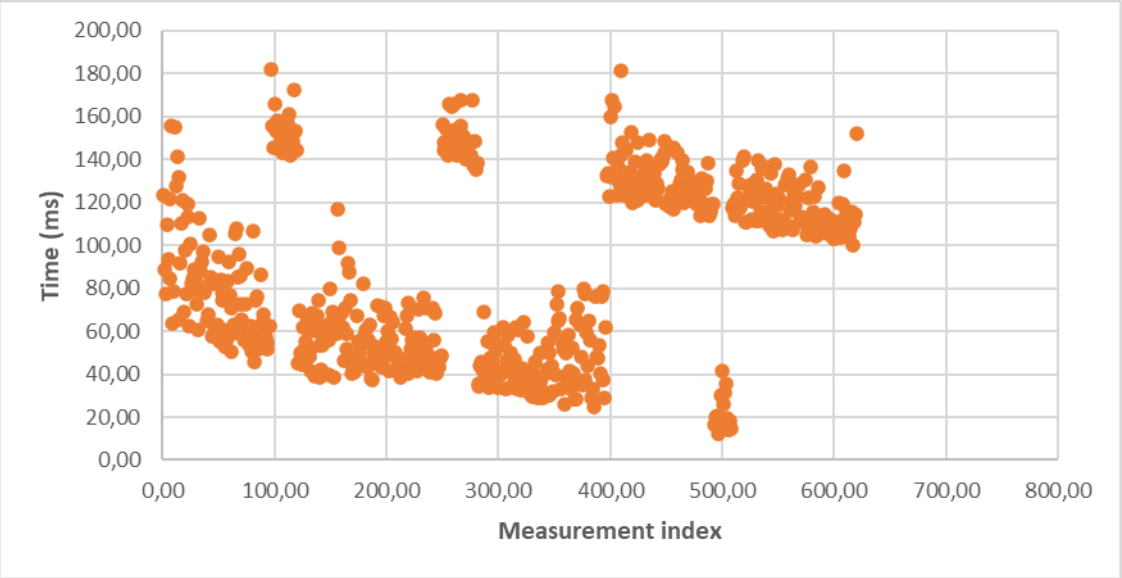

***Figure 4.10**: t_3 histogram for the second run with 3 vehicles*

Individual time measurements were similar to first run with 3 vehicles, so, only the chart of t_3 was given in **Figure 4.10**. Asynchronous runtime pattern can be seen as in the first run within the 12.75-185.83 ms range.

***Table 4.5:** Statistics of the second run with 3 vehicles*

	t_1	p_1	t_2	p_2	t_3	p_3	t_4	p_4	t_5	***TOTAL***
MIN	16	-	0,08	0,05	12,53	0,09	0,00	0,00	0,00	*12,75*
MAX	16	-	1,32	0,78	182,14	0,65	0,24	0,00	0,00	*185,13*
AVG	16	-	0,24	0,18	88,20	0,24	0,02	0,00	0,00	*88,88*
STD. DEV.	0	-	0,12	0,09	41,06	0,10	0,02	0,00	0,00	

The statistics of the measurements for the first run with 3 vehicles were presented in **Table 4.5**. In the second run, a total of 88.88 ms was observed as the sum of the average

values of all durations.

3 vehicles – Comparison:

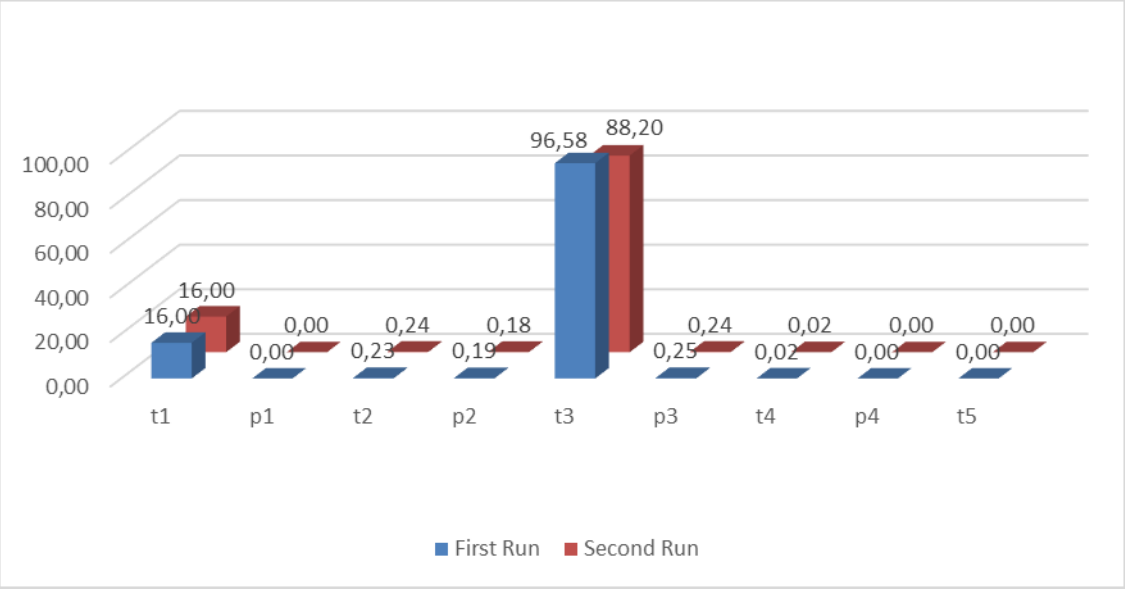

***Figure 4.11**: Comparison of first and second run with 3 vehicles*

Figure 4.11 shows that differences and trends of statistics of individual time measurements are similar to the 2-vehicle case. Therefore, similar comments can be made for 3-vehicle case.

General Overview:

In the total of 4 measurements conducted with 2 and 3 vehicles, the average duration of approximately 83-105 ms for the user-applied changes to the vehicle, which are then encapsulated in CAM packets and transmitted to other vehicles, is observed. These measurements encompass the process from the occurrence of the state change in the vehicle until it is transmitted to the Communication Controller. Therefore, the transmission of this message to the CA services of other vehicles is not included. However, upon examination of the logs, it was observed that after the Communication Controller, the time for this message to reach the CA services of other vehicles is approximately $\mathbf{t_4} + \mathbf{p_4} + \mathbf{t_5}$. This corresponds to a time well below 1 millisecond. Hence, it can be stated that the state change occurring in the vehicle reaches the CA services of other vehicles within an average of 83-105 ms. It is important to note that p1 is not

included in these measurements. Due to the asynchronous nature of the CARLA server's operation, a precise measurement could not be made. We are aware of the server's operating FPS. During our experiments, we achieved an average FPS value of 60, so we can make an approximate estimate of 16 ms for $\mathbf{p_1}$. As a result, it is anticipated that a duration ranging from 99 to 121 ms emerges. Additionally, in the case of runs with 3 vehicles, no significant performance loss compared to 2 vehicles was observed.

Mentioned delay values are totally acceptable since there exist asynchronous workflows in the platform. Ignoring those would result an end-to-end delay below 100 ms which indicates a responsive user perception experience.

4.2. Longitudinal Collision Risk Warning Application

The LCRW application is a collision risk calculation application for two vehicles, as detailed in the ETSI TS 101 539-3 (European Telecommunications Standards Institute, 2013) standard. This standard encompasses various collision scenarios. Some of these scenarios involve calculating the collision risk using CAM packets, whereas others use DENM packets.

In our simplified version of this application, the calculation of collision risk between two vehicles is based on examining the content of the transmitted CAM packets. During this calculation, fundamental variables, such as the position, speed, and heading of the vehicles, were utilized. The TTC is derived from these variables. In our application, an alert was generated when the TTC dropped below 3 s. This alert was shared with the user through the developed HMI.

To demonstrate the accuracy of the application, various snapshots were captured when collision risk alerts occurred. These snapshots were used for perspective validation. However, the purpose of developing this application and the verification method is not solely to showcase its success but rather to demonstrate the capability of the test platform for testing such applications.

Three example cases have been selected for this application. In the first case, the two vehicles approached each other in a face-to-face manner. In the second case, one vehicle approached the other from behind. In the last case, the vehicles were on different lanes to demonstrate a non-risky case.

4.2.1. Frontal Collision Risk Test

In this case, the red and black vehicles approach each other linearly. As the vehicles approached each other in a straight line, the collision risk increased. The application continuously provided warnings from the moment TTC dropped below 3 s until the risk completely disappeared, and these warnings were displayed on the HMI. For instance, when the distance between vehicles was approximately 25 m, there was a collision risk between the red and black vehicles. The perspectives of red and black vehicles are shown in **Figure 4.12** and **Figure 4.14**, respectively. Simultaneously, the HMI of the red vehicle displayed a warning, as shown in **Figure 4.13**.

Figure 4.12. *Red vehicle's perspective during a frontal collision risk*

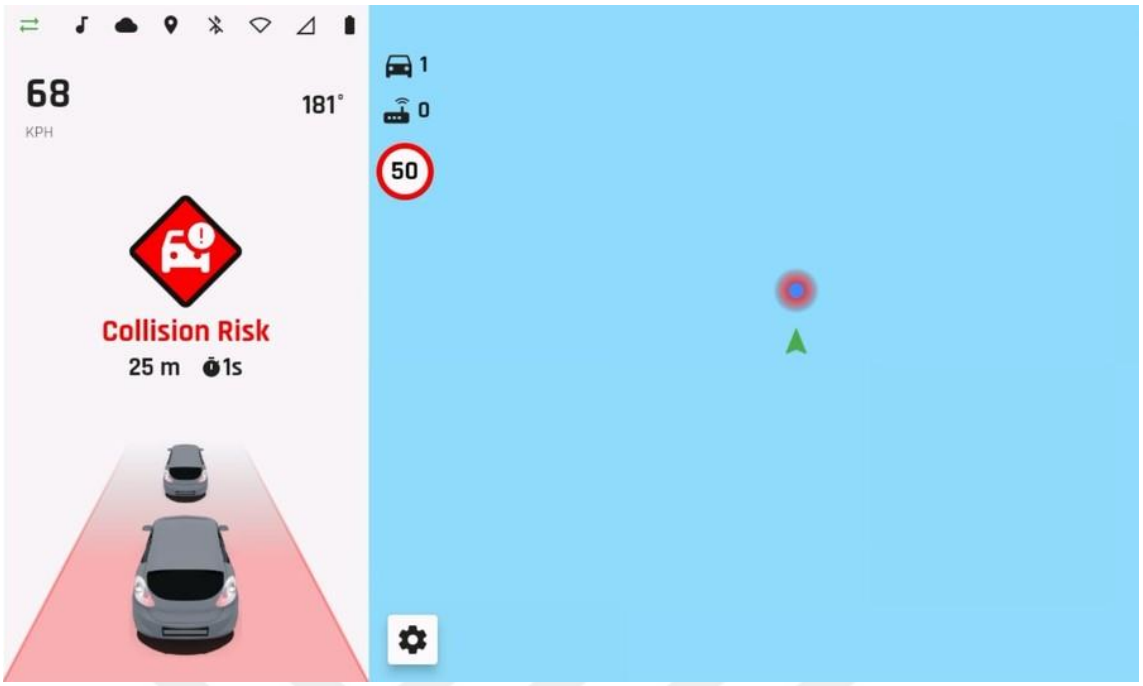

Figure 4.13. *Collision risk warning on HMI during frontal collision risk*

Figure 4.14 depicts the black vehicle at standstill. During these tests, the black vehicle was intentionally kept stationary, as it did not have any impact on the outcome.

Figure 4.14. *Black vehicle's perspective during a frontal collision risk*

4.2.2. Forward Collision Risk Test

In this case, the red vehicle dangerously approaches the black vehicle, which moves slower than itself, from the rear. An alert was displayed on the HMI from the moment the TTC dropped below 3 s until the risk was eliminated. The perspective of the red vehicle in this case is shown in **Figure 4.15**. The corresponding alert displayed on the HMI at that time is shown in **Figure 4.16**.

Figure 4.15. *Red vehicle's perspective during a forward collision risk*

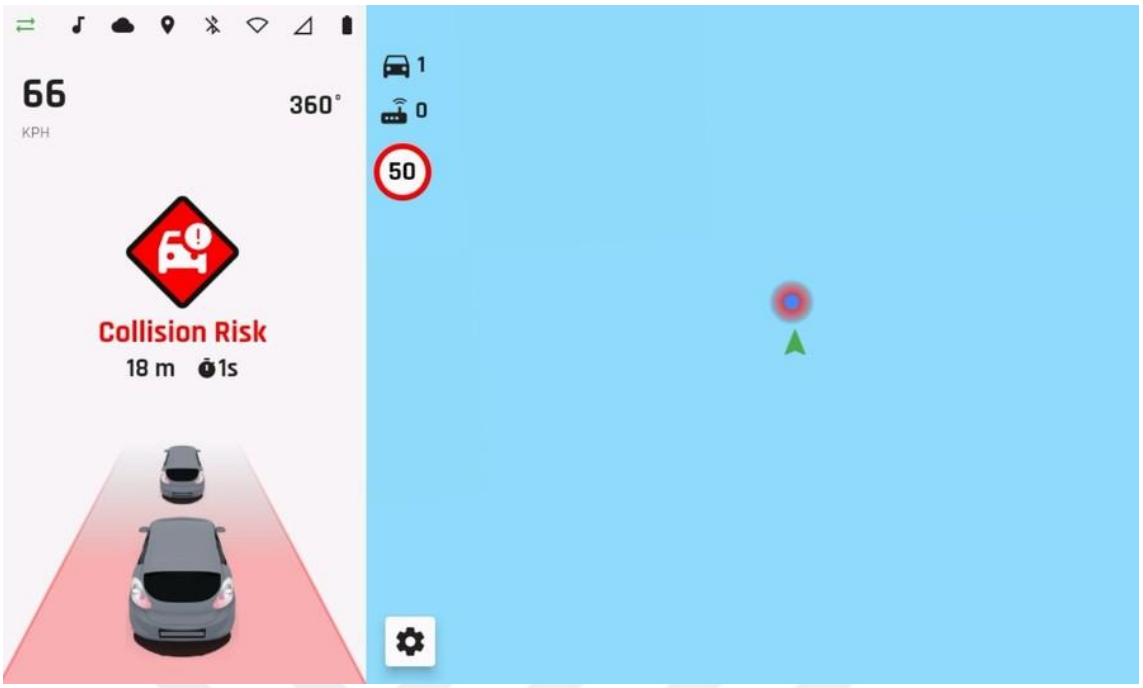

Figure 4.16. *Collision risk warning on HMI during forward collision risk*

4.2.3. Different Lane Test

In this scenario, the two vehicles approach each other in different lanes. Upon examining the trajectories of the two vehicles, no collision risk was observed. Consequently, the two vehicles safely passed each other without any risk. Therefore, no warnings were generated on the HMI of the two vehicles. The perspectives of red and black vehicles are shown in **Figure 4.17** and **Figure 4.19**, respectively. Simultaneously, the HMI screen displayed to the user by the red vehicle is shown in **Figure 4.18**.

Figure 4.17. *Red vehicle's perspective during driving on different lanes*

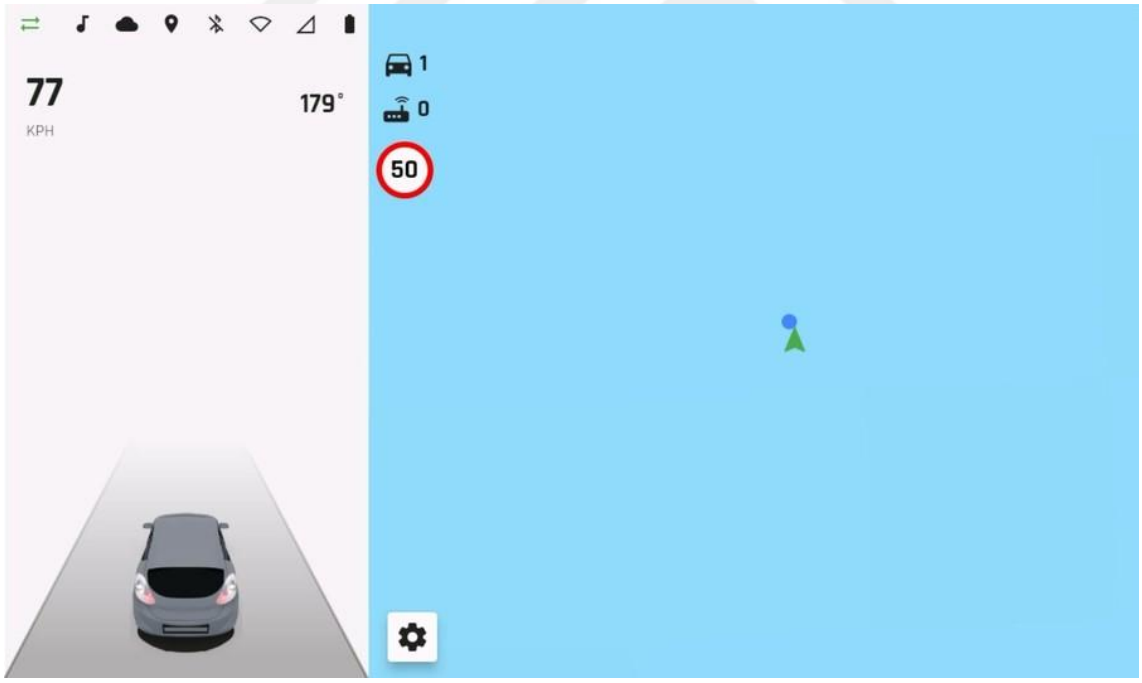

Figure 4.18. *No collision risk on HMI when vehicles are on different lanes*

Figure 4.19. *Black vehicle's perspective during driving on different lanes*

5. CONCLUSION AND FUTURE WORK

V2X communication utilizes wireless communication technologies such as WiFi and cellular networks to improve safety, information sharing, and entertainment among vehicles, infrastructure components, and other stakeholders in traffic. The connectivity of vehicles allows them to be more effectively and accurately informed of their surroundings. Information shared between connected vehicles and infrastructure systems enables the development of numerous applications.

The implementation of software and hardware developed in this field in the real world is impossible without comprehensive testing. The inclusiveness, repeatability, and cost-effectiveness of the tests are crucial for accelerating advancements. Therefore, it is essential to develop test environments tailored for this purpose.

Based on this need, this study aimed to create a test platform where users can act as vehicle drivers and control computer-generated vehicles, allowing for the demonstration and validation of V2X applications. In the development of the test platform, efforts have been made to isolate and design the software components of vehicles as closely as possible to their real-world counterparts. During the development of these components, we made use of open-source libraries and our own coded modules.

Several delay measurements were conducted to demonstrate the stability and performance of our test platform. These measurements aimed to show how quickly the user-provided control inputs are included in CAM packets and assess the efficiency of V2X communication. The measurements revealed negligible delays, indicating high performance of the test platform. Additionally, to demonstrate the feasibility of application testing on our platform, the LCRW application was implemented. Snapshots depicting the vehicle states during alert times were presented as part of the application validation process. The application test demonstrated that the developed algorithm responded as expected, providing necessary warnings to the user regarding collision risks.

Further research and development efforts on our test platform could lead to a more realistic and functional platform. For instance, in this study, a keyboard was used for vehicle control. Enhancements can be made to enable the use of steering wheel sets,

thereby increasing the realism of the inputs and vehicle control. Furthermore, the absence of channel modeling in our test platform reduces the realism of the V2X communication. Additionally, although this study focused on scenarios involving a maximum of three vehicles, conducting tests on a larger scale (involving more vehicles) would be beneficial. Based on these outcomes, the necessary optimizations can be made as required.

REFERENCES

a car dashboard with a gps device photo. (2022, June 26). Retrieved 03 16, 2023, from https://unsplash.com/photos/9aaD5DWs2_g?utm_source=unsplash&utm_medium=referral&utm_content=creditShareLink

CAR 2 CAR Communication Consortium. (2019, September 25). *Guidance for day 2 and beyond roadmap.* Retrieved 5 15, 2023, from https://www.car-2-car.org/fileadmin/documents/General_Documents/C2CCC_WP_2072_RoadmapDay2AndBeyond.pdf

Chen, M., Lu, X., Fang, Y., & Chan, S. (2018). Vehicle-to-everything (V2X) communication: Architecture, applications, benefits, and challenges. *IEEE Access*(6).

Chen, Y., Chen, S., Xiao, T., Zhang, S., Hou, Q., & Zheng, N. (2020). Mixed Test Environment-based Vehicle-in-the-loop Validation - A New Testing Approach for Autonomous Vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)* (pp. 1283-1289).

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). {CARLA}: {An} Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1-16.

European Telecommunications Standards Institute. (2013, November). *ETSI TS 101 539-3 V1.1.1.* Retrieved 5 1, 2023, from https://www.etsi.org/deliver/etsi_ts/101500_101599/10153903/01.01.01_60/ts_10153903v010101p.pdf

European Telecommunications Standards Institute. (2014). *EN 302 637-2 - V1.3.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service.* Retrieved 3 15, 2023, from https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.01_30/en_30263702v010301v.pdf

European Telecommunications Standards Institute. (2014). *EN 302 637-3 - V1.2.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of*

Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service. Retrieved 3 15, 2023, from https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.01_30/en_30263703v010201v.pdf

Google. (2017). *Flutter - Build apps for any screen*. Retrieved 3 11, 2023, from https://flutter.dev

gpsd — a GPS service daemon. (1995). Retrieved 3 15, 2023, from https://gpsd.gitlab.io/gpsd/

Karagiannis, G., Altintas, O., Ekici, E., Heijenk, G., Jarupan, B., Lin, K., & Weil, T. (2011). Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards, and solutions. *IEEE Communications Surveys & Tutorials*(16), 584-616.

Malinen, J. (2008). *mac80211_hwsim - software simulator of 802.11 radio(s) for mac80211*. Retrieved 3 15, 2023, from https://www.kernel.org/doc/html/next/networking/mac80211_hwsim/mac80211_hwsim.html

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, p. 2.

Olaverri-Monreal, C., Errea-Moreno, J., Díaz-Álvarez, A., Biurrun-Quel, C., Serrano-Arriezu, L., & Kuba, M. (2018). Connection of the SUMO Microscopic Traffic Simulator and the Unity 3D Game Engine to Evaluate V2X Communication-Based Systems. *Sensors*.

Riebl, R. (2013). *Vanetza - Your open-source ETSI C-ITS protocol stack*. Retrieved 3 12, 2023, from https://www.vanetza.org/

Riebl, R., Günther, H.-J., Facchi, C., & Wolf, L. (2015). Artery: Extending Veins for VANET applications. *International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. Budapest.

Senkus, B., Yaman, B., Aydin, H., & Soyturk, M. (2022). Implementation of High Performance Multi-Agent Position Feeding Framework. *2022 24th International Microwave and Radar Conference (MIKON)*. Gdansk.

Sommer, C., German, R., & Dressler, F. (2011). Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing (TMC)*(10), 3-15.

Szendrei, Z., Varga, N., & Bokor, L. (2018). A SUMO-Based Hardware-in-the-Loop V2X Simulation Framework for Testing and Rapid Prototyping of Cooperative Vehicular Applications. In *Vehicle and Automotive Engineering 2* (pp. 426-440). Springer International Publishing.

Wang, J., & Zhu, Y. (2022). A Hardware-in-the-Loop V2X Simulation Framework: CarTest. *Sensors, 22*(13).

Xu, Y., Ge, Y., Wei, D., Yu, R., Wang, J., & Yao, Z. (2021). V2V Test Scenario-Study on Intersection Collision Warning. *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring).*

ÖZGEÇMİŞ

I have completed my primary and high school education in Manisa. In 2015, I won the Department of Computer Science & Engineering at Marmara University and settled in Istanbul. After one year of language education, I began my own department. As a result of my next four years of education, I obtained my bachelor's degree with an average of 3.48/4.00. In the fall of 2020, I started my master's education at the Marmara University Computer Science & Engineering department from the 1st place among all candidates. I completed my master's courses with an average of 3.63/4.00 and started my thesis process.

Since the last year of my undergraduate education, I have been working at VeNIT Lab, which is located in the Department of Computer Science & Engineering at Marmara University. Here, I work on automotive, V2X communications, and mobile application development, and contribute to European Union funded projects.