Budapest University of Technology and Economics
Faculty of Transportation and Vehicle Engineering
Department of Automotive Technologies

Master thesis

Development of a V2X test application for proving ground use.

Author:
Afrifa Ebenezer Kwaku Oppong

Supervisor:
Dr. Árpád Török

Budapest, 2023.

Budapest University of Technology and Economics, Department of Automotive Technologies
Budapest, XI. Stoczek utca 6. J. épület, Levélcím: Budapest, Postafiók 1521, tel. 06-1-463-1615, fax: 06-1-463-3978
E-mail: titkarsag@gjt.bme.hu, web: www.gjt.bme.hu

BME DEPARTMENT OF AUTOMOTIVE TECHNOLOGIES

SPECIFICATIONS OF MASTER'S THESIS

Name: Afrifa Ebenezer Kwaku Oppong (BAXU7J) **Form**: Auton. Vehicle Control Eng. MSc
Course code: BMEKOGGM554

Title: Development of a V2X test application for proving ground use

No.: *GJT-M-G-2023-3* **Accessibility:** public

Tasks:

- Performing literature review
- Planning and creating the test software architecture
- Creating driving scenarios in OpenSCENARIO format (optional)
- Developing the Python HMI application to support V2X testing procedures.
- The HMI application should be capable of the following features:
  - -Load the OpenSCENARIO file.
  - - Compare the actual vehicle state to the predefined scenario
  - - Instruct the driver to follow the planned trajectory
  - - After the test, evaluate the test's success with numerical results
- Evaluation of results (edited)

Supervisor at the Department: Dr. Török Árpád

Host institution: BME Department of Automotive Technologies

Industrial/External Supervisor:

Subjects of Final Examination:

1. Automotive network and communication systems, Vehicle testing and validation - BMEKOGGM709, BMEKOGGM406, 7, Differentiated
2. Safety and reliability in vehicle industry - BMEKOKAM703, 3, Basic
3. Automotive environment sensors-BMEKOKAM708, 5, Differentiated

Deadline date of Thesis submission: 2nd June 2023

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
GÉPJÁRMŰTECHNOLÓGIA TANSZÉK
KÖZLEKEDÉSMÉRNÖKI ÉS JÁRMŰMÉRNÖKI KAR
1.

P.H.

Zsolt Szalay Ph.D.
associate professor
head of department

Declaration

I, Afrifa Ebenezer Kwaku Oppong, hereby declare that the present master's thesis was composed by myself and that the work contained here in is my own. I also confirm that I have only used the specified resources. Any part that I have taken from another source, either verbatim or in the same content, but reworded, has been clearly marked with the source indicated.

.....

Afrifa Ebenezer Kwaku Oppong

Abstract

This project aims to develop an advanced application for use on test tracks, with the specific objectives of ensuring compliance with prescribed vehicle dynamics, reliability, and safety criteria, as well as providing a graphical, intuitive, and user-friendly interface (HMI) for test drivers. The application utilizes GNSS positioning data (latitude, longitude, elevation) and IMU sensor data (acceleration, heading angle) to guide the test driver and evaluate test results numerically.

The developed application serves as a valuable tool to assist test drivers on proving grounds, which are specialized facilities designed to evaluate the performance of vehicles and ensure their adherence to specific criteria. By providing guidance to the test drivers, the application aids in navigating the proving grounds and conducting tests according to established procedures.

Furthermore, the application offers comprehensive evaluations of test results, highlighting any areas where the vehicle fails to meet the prescribed criteria. It suggests potential improvements or modifications based on the evaluation, empowering test drivers to identify and address issues promptly. This approach contributes to enhancing overall vehicle performance and safety.

The project's significance lies in its ability to provide an effective and efficient system for evaluating test driver's performance on test tracks. The application's intuitive interface and numerical analysis enable test drivers to make informed decisions and take necessary actions to improve the vehicle's compliance with desired standards.

Future enhancements may include integrating machine learning algorithms that could further improve the application's decision-making abilities, enabling it to adapt dynamically to varying test conditions.

Contents

Contents

1 Introduction

Automobile manufacturers and government agencies have been looking for ways to improve road safety, manage traffic more efficiently, and make it more comfortable in the nearby future for many years long. The all-encompassing phrase for the connected communications of a vehicle is known as vehicle-to-everything, or V2X for short. A car can use its onboard communication facilities to provide real-time traffic information, pre-emptively react to changing road conditions, recognise road signs and warnings, and more. This is the underlying notion. This capability will be available in the future. Even though V2X functions by itself and will not be able to control a vehicle that doesn't have a driver on board, they are essential building blocks that enable an autonomous vehicle to put together a picture of its surroundings [1].

V2X information and communications technology provide low-latency vehicle-to-vehicle communications, bringing a new aspect to future driver assistance systems [2]. The V2X communication will make it feasible to develop a wide variety of new applications and services in the areas of vehicle safety, comfort, and information and entertainment. For the purpose of covering future cooperative traffic scenarios, which will involve a large number of cars and infrastructure equipment, new testing and assessment techniques are necessary [3]. These help in scenario generation for testing.

With these vehicle driver assistance systems, the need for rigorous testing becomes quite important. Road testing is the prevalent and fundamental testing approach to validate Advanced Driver-Assistance Systems (ADAS) and Automated Driving Systems (ADS) in the automobile industry. Testing in real-time is nearly impossible since validating the failure rate of an ADS requires a significant number of operational hours. Virtual testing or hardware-in-the-loop testing is one method that can be utilised to lighten this testing load. The most difficult aspect of this task is coming up with test scenarios that are both realistic and accurate representations of what might happen, out on the road [4].

Any new vehicle, whether a car, a race car, a tank, a ship, or a motorcycle, propelled by gasoline, diesel, electric or hybrid electric vehicles, or alternative fuels, must undergo

exhaustive testing before it can be introduced to the public. The automotive industry uses testing grounds, sometimes known as proving grounds or test tracks, to evaluate both on-road and off-road vehicles.

During the process of a vehicle's development, its capabilities are evaluated in a range of different environments and under a variety of driving conditions. The dynamic features of vehicles are typically the primary focus of test cases. Vehicles are frequently put through rigorous testing, during which they are subjected to challenging environments and speeds. Up to the 1920s [5], testing of automobiles was placed on public roadways, including those in cities and rural areas. Vehicle testing is relocated from public roads to proving grounds, which are controlled, secure, and safe testing environments. These environments simulate a wide variety of road types and events, many of which are reflective of or related to how customers would use the vehicle.

Bringing all this together, that is, the importance of V2X, scenario generation and moving to proving grounds for vehicle testing, Human Machine Interface (HMI) must be employed. No human-machine interface (HMI) means drivers cannot interact with autonomous vehicles (AVs). In addition, a successful human-vehicle dialogue is impossible to achieve without an appropriately designed human-machine interface system [6]. Level 3 autonomous driving necessitates especially clear and explicit human-machine interface design. Improvements in an automotive HMI design first appeared in ADAS capabilities like adaptive cruise control (ACC) and lane change assist (LCA) in the first stages of autonomous driving. Lack of consistency is the greatest challenge in developing an HMI for an autonomous vehicle design since it can lead to potentially fatal incidents of driver confusion and errors. Examples include automatic functions like cruise control and lane change assistance. Manufacturers utilise a variety of HM design features to alert drivers, for example, when they are veering dangerously close to the left lane boundary. When designing a system's human-machine interface, consistency is key to ensuring usability.

So, for testing purposes, we need an efficient and user-friendly HMI interface of a V2X application that bridges communication between simulation testing and proving ground testing. This study focuses on that. We are proposing a design and implementation of a V2X application to assist in testing on grounds by providing information about the test scenario from a simulated test.

1.1 Related Work

Related work from Alam [7] concerning the industry's rapid advancement, there are currently not enough tools available for systematic testing. Testing in the real world consumes time and, above all, entails risks. Additionally, there is no infrastructure in place to automatically produce crucial scenarios for testing autonomous vehicles.

A general framework for testing an autonomous vehicle from beginning to end in a simulated environment is developed in this thesis. The framework offers the capacity to design and carry out a variety of traffic scenarios in a trustworthy manner. The criticality of a transportation scenario can be calculated using either one of a given two ways.

The probability distribution of the crucial scene is repeatedly learned using a so-called critical value. To test and compare various autonomous vehicles, one can utilize the resulting probability distribution to sample key scenarios. Open Drive and Open Scenario standards are used to identify the static and moving players in the simulator's simulation of urban traffic.

Two separate criticality metrics are used while utilizing the same architecture to verify the resilience of the design. To comprehend how the criticality metric affects the convergence and characteristics of the probability distribution, the performance of the metrics, time to collision were examined. Large critical scenarios for testing an autonomous agent and setting benchmarks in a virtual environment can be produced using the probability distribution that was constructed. Through constant resampling and optimization, the probability distribution is learned. The findings thus support the claim that learning-based techniques can be applied to discover the distribution of key scenarios. The suggested architecture quickly converges and is very effective.

Tamilarasan, Jung, and Guvenc [8] research explain why it is not possible to utilize the standard drive cycle profiles and suggests a driving scenario-based framework for assessing the speed optimization algorithm's fuel efficiency. The approach for creating driving scenarios described in this work can be used as a general tool for creating tests, and analysis metrics can be utilized to determine which driving scenarios are pertinent for assessing the test strategy for the fuel economy of connected and autonomous vehicles.

Future work may incorporate data from the chosen route, such as the locations of stop signs, traffic signals, traffic Signal Phase and Timing (SPaT), speed restrictions, gradients, etc., as constraints for a dynamic programming technique that will compute the ideal speed under these constraints.

Wen, Park, and Cho [9], the study suggested a pipeline that creates multiple simulation situations for autonomous vehicles. The training of a Convolutional Neural Network (CNN) for the selection of appropriate agents, the key contributions of the proposed system are the production of realistic situations including humans, animals, and cars, and the deployment of an event-centric action dispatcher module to create associated actions for agents near the event location in real-time. To guarantee that the simulation matched reality, they generated not just the scenarios but also the associated actions of the surrounding agents. The scenario generated a scenario map to recreate only the area around the autonomous vehicle for real-time performance. The trials revealed that the CNN-based scenario agent selection may reach a high accuracy of 92.67%.

The scenario-generating pipeline created believable scenarios in the virtual world for autonomous driving with the help of the event-centric action dispatcher module. The experimental findings demonstrate that this pipeline can be used to create scenarios for autonomous driving in a virtual environment. In the end, the activities sent out by the event-centric action dispatcher are categorized using human experience. Manual summarizing takes time, though, if further events are required. Their focus is currently on methods that automatically identify the action set associated with the associated event when applied to video data that has been classified based on event categories using an action recognition model applied via deep learning.

Menzel, Bagschik, and Maurer [10] using the ISO 26262 standard's development process as a guide, examined the viability of a scenario-based approach for the design of vehicle guiding systems. To accomplish this objective, the researchers have carefully identified the specific stages in the process where scenarios can play a crucial role in generating the desired outputs for each step. In addition, they have proactively addressed any conflicts that may arise when fulfilling the requirements of different process steps, particularly in terms of how scenarios are portrayed, and have come up with effective solutions. To ensure the success of their approach, the authors have proposed three different levels of abstraction for scenarios. Moreover, they have provided clear demonstrations of how these levels of abstraction can be effectively applied to generate high-quality work products for various process stages, as specified in the ISO 26262 standard. Additionally, each level of abstraction that has been added has been defined. Future functional scenario generation and conversion into physical scenario creation may require new approaches and technologies as the ISO 26262 standard is developed. A companion submission with a knowledge-based method for developing functioning situations with a wide variety was also made to the

2018 IEEE Intelligent Vehicles Symposium in addition to this one. The suggested degrees of abstraction can therefore incorporate the current data formats for situations. A test concept for automated cars can then be used to build new techniques and tools for scenario formulation and scenario concretization.

For Karunakaran et al. [11], a framework for scenario extraction for lane-change operations was proposed. It first creates a road network using lidar data, and then it extracts lane change scenarios from that network using object tracking data. Road networks and extracted scenarios were represented in OpenX formats. The suggested framework is an automated pipeline that can produce OpenX files without requiring the pipeline's content to be manually created. The outcome demonstrates that the framework can recreate real-world scenarios in a simulation. They were expanding on this effort to create an urban validation dataset with realistic traffic participant behaviour in an urban setting. They examined the SUT in more dangerous circumstances by using the validation dataset to produce numerous versions of specific scenarios and analysed it in genuine scenarios with only minor changes to the dataset playback portion, the framework could be applied to more datasets, probably adapting the framework to work with more well-known datasets by changing the playback section. Many datasets give point cloud with intensity, IMU, and tracking data.

In Bengler et al. [12], this research offered information on the many HMI kinds found in AVs, grouped according to whether they are used for internal or external communication. This study analysed and summarized aspects that affect the choice of HMI type and HMI content in addition to providing a thorough description of which information could be assigned to which HMI. The HMI framework incorporates the many HMI kinds, the factors that influence them, and how these three components interact. Additionally, the relationship between the various HMI components because of variations in the features of the influencing factors was shown. When exploring the interaction tactics of AVs with their passengers or the surrounding HRU, it is crucial to consider all HMI forms, which is why we built the framework. The influencing elements may change because of changing the design of just one interface, which will then have an impact on the HMI choice. Therefore, while investigating interaction techniques in the context of automated driving, a comprehensive approach considering internal and external communication is essential.

Naujoks et al. [13], in their research, they evaluated whether speech output could improve Conditional Automated Driving (CAD) human-machine cooperation. Improved system openness by reporting forthcoming automated moves has not been examined

thoroughly. Seventeen (17) participants completed the same driving simulator course twice, once with general aural feedback and once with voice output. Whether adding more speech output will improve human-machine interaction by effectively alerting the driver to impending automated driving manoeuvres and, as a result, would interfere less with the execution of an NDRT, that is, National Disaster Response Team. Automated driving might be particularly advantageous and alluring due to the potential for engaging in NDRTs without the requirement for interruptions.

2 Literature Research

The next generation of driver assistance systems in vehicles will be built on top of cutting-edge technologies such as V2X communication methods [3]. Application developers are better able to participate in approaches known as test-based development, in which they may test and evaluate their work as it is being completed. The incorporation of wireless communication into automobiles paves the way for several new avenues of improvement in the realms of vehicular safety, mobility, energy efficiency, and overall driving experience. Because of this, the requirements for testing and evaluating procedures are going to be affected.

Existing limitations on the capabilities of driver assistance systems mean that on a road with heavy traffic, if a car breaks down in front of oneself but is concealed by a slope and is quite challenging to see, there is a risk that drivers who are following you will not acknowledge the situation until it is too late or that they will not recognise it at all, which will result in a collision between you and the car that broke down in front of you [14]. This is due to the fact that driver assistance systems do not have the capability. In these kinds of circumstances, even vehicles equipped with sophisticated radar-based, ADAS reach the limitations of their capabilities, as these systems are unable to obtain any information about traffic occurrences that occur outside of their field of vision. Radio signals can be transmitted from vehicles equipped with V2X to other vehicles and road users in the immediate area, alerting them to potential hazards and allowing them to respond in a timely manner. Managing V2X messages and the contents of those messages on the V2X can be done on electronic control units (ECUs) of the first generation, Each have a transmit and receive unit, the V2X communication stack, and driver assistance capabilities that are based on V2X specifications. Another requirement for test tools is that they need to be able to comprehend application-specific messages. These messages are what carry the actual useful information, such as the position of the vehicle and the information from the sensors.

V2X communications sent from the test tool have to be digitally certified based on the

certificates provided by the Public key infrastructure (PKI) before they can excite ECUs. The testing tool needs to have the capacity to manage certificates, provide support for the corresponding cryptographic algorithms, and build the appropriate security header for the V2X package. The capacity to create the whole data traffic for all networks that are needed by the ADAS ECU in a time-synchronous manner is necessary for the stimulation of ADAS functionalities. The time stamps of the data obtained by measuring interface need to be synchronised across all networks in order for the various data to be analysed in connection to one another. This is necessary in order for the data to be analysed in connection to one another. This is another problem for V2X test systems. This is of the utmost importance when doing an analysis of the positions of objects observed by the various sensors at a certain point in time and having the fusion algorithm generate location findings based on those positions. Users are given the ability to load their own test scenarios into the V2X test tool, which then generates legitimate V2X communication. In conclusion, when it comes to evaluating, stimulating, and testing a V2X-based driver assistance function, test engineers are faced with a number of unique obstacles. The testing tool needs to understand messages from applications in a certain domain and be able to deal with the special communication protocols used by V2X. As the assistance functions' behaviour is affected by the internal vehicle networks' behaviour, it is crucial that the test tool be capable of processing data from network sources so that they may be incorporated into the testing. Allowing for automated testing is crucial because it ensures that the existing tests can be increased in the long term and that the ECU can always be triggered and tested with a wide range of parameters.

As discussed by Naujoks et al. [13], the evaluation of human-machine interfaces (HMIs) has a deep-rooted history in the automotive industry. In the context of manual driving, the evaluation is concentrated on the potential for distraction posed by in-vehicle information systems. As a result, the evaluation's primary objective was to determine the amount of visual workload associated with in-vehicle information systems. In the context of automated driving, a number of theoretical constructs related to safe driver-vehicle interaction (DVI) such as trust, controllability, mode awareness, or usability could be used as criteria. This is possible as it has been shown by research that, these constructs present a unique set of difficulties when it comes to the design and evaluation of HMIs for ADAS.

2.1 Language for Scenario Specification

A probabilistic programming language with domain-specific features for simulating the surroundings of cyber-physical systems is quite crucial. A scenic program defines a distribution over scenes, object configurations, and agents, for example, a program describing bumper-to-bumper traffic may specify a specific distribution for the space between cars while leaving the scene's location to be uniformly random across all 3-lane roads in the city.

Declarative restrictions and the easy geometry syntax offered by scenic allow for the concise and clear definition of such complex scenarios. Scenic has many uses in the design of ML-based systems. For instance, one can create a scenic program that describes an uncommon traffic situation, such as a disabled automobile blocking the road, and then sample from it to produce customized training data that can be added to an existing dataset. More generally, the language's formal semantics enable its use as a precisely described model of a systems surroundings [14]

Figure 2.1: The Autonomous Vehicle (AV) and pedestrian dummy used for track testing [14]

over the AV's surrounding agents, such as inflated autos and pedestrian dummies used for accident testing as seen in Figure 2.1 above. The ability to operate the genuine AV with its real hardware and software systems in locations that can be intended to replicate some difficult driving conditions is known as track testing. Track testing, however, may be exceedingly costly, labour- and time-consuming to set up. Which tests ought to be conducted considering these difficulties is a valid question to consider. The ability to conduct tests that will be most successful at spotting errors or odd behaviour, revealing defects, and boosting confidence in the safety of the AV and ADS is essential for testing AVs with complicated ML-based components. Automatic Lane Change scenario extraction

and synthesis from real-world data into scenarios in OpenX format. Although there are still numerous safety issues to be overcome, the widespread deployment of autonomous vehicles (AV) appears to be imminent. Machine learning and probabilistic techniques, which significantly increase the complexity of the conventional methods of verification and validation, will surely be used in modern autonomous vehicles. Prior to deployment, road testing is crucial, but since most scenarios may be repeated, it might be tricky to gather demanding events, exploring various, different, and important scenarios requires time and money. Scenario-based testing has recently gained widespread acceptance in both the research community and industry. It can lessen the testing effort needed because it is specifically focused on the pertinent important road circumstances. For the simulation's scenario-based testing to be effective, the traffic participants' behaviour must be realistic. To encode the behaviour of genuine traffic participants, it is crucial to record scenarios from the real world. The point-cloud data and object-tracking information used to capture the lane change scenarios are innovative scenario extraction techniques. Relatively, to other methods in this field, this one allows for entirely automatic scenario extraction. The created scenarios can be simply reused in the System Under Test (SUT) evaluation because they are represented in OpenX format. The goal of this framework was to provide a validation dataset that will enable the generation of several important real scenarios [11]. Road testing is a common and crucial testing method used in the auto industry to validate ADS [15]. Though most of the driving information gathered during road testing is regarded as non-critical. Therefore, most of the scenarios don't have an adequate bearing on the evaluation criteria [16] and in addition, every software upgrade will erode accumulated confidence [17].

This paper by Ponn et al. [18] presents a converting driving scenario to generate a format suitable for LG Silicon Valley Lab (LGSVL) simulator. Since autonomous vehicles are developed worldwide, to reduce the test cost, virtual simulators are used in the automotive industry. However, virtual simulators require simulation environments, such as roads, pedestrians, and vehicles. Configuring numerous test case parameters manually is not an efficient approach for developers. MATLAB/Simulink provides a graphical interface for scenario creation but has its limitations. Hence, the proposed solution was to develop a framework that converts scenarios designed in MATLAB/Simulink into a compatible format for the LGSVL simulator, an autonomous vehicle simulator. The converted scenarios can then be executed in the LGSVL simulator, similar to scenarios created directly in MATLAB/Simulink. And also, the LGSVL simulator has collaborated with Autowave, an

open-source self-driving system. This collaboration enables convenient testing of self-driving systems. By using this framework, developers can effectively generate scenarios for testing purposes [19].

2.2 Scenario-based approach

ADS cannot be normally validated by test drives anymore, as Kalra and Paddock [20] already noted in 2016. The scenario-based technique, commonly referred to as the "database approach", was used to address this issue [21]. The test drive was condensed into scenarios to eliminate from the validation process any sections that are irrelevant or do not involve an action or occurrence (such as boring straight driving). ADS development can already be completed using scenario-based testing, as shown by Sippl et al. [22].

Scenarios can be separated into many different groups. Detail level depends on the amount of detail. Scenarios are classified by functional, logical, and concrete scenarios as proposed by Menzel, Bagschik, and Maurer [10]. While language descriptions of functional scenarios are possible, logical scenarios already include parameter spaces and concrete scenarios' precise parameter values. This gradually lowers the level of abstraction and expands the range of potential outcomes from functional to tangible scenarios. A bird's-eye view of the scene and its contents can best convey the level of observation. Differentiating between driving and traffic situations is done by Rodarius et al. [23]. While driving scenarios only involve a small number of vehicles, traffic scenarios take place in a larger setting that also includes a range of driving scenarios.

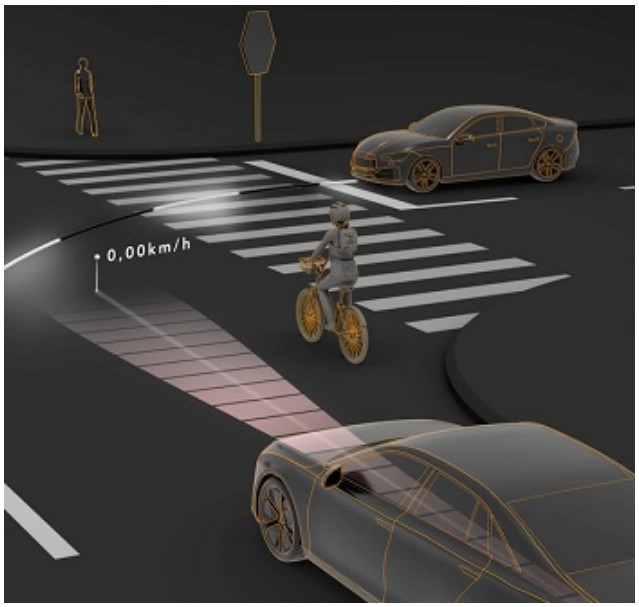


Figure 2.2: Scenario-Based ADAS/AD Testing, Design and Editing [24]

Driving on a highway segment is a typical traffic scenario. In addition, Elrofai, Paardekooper, et al. [25] added scenario classes that include scenarios that are related, such as "urban scenarios". This allows for the classification of a scenario into various classes level of information because a scene includes a variety of elements, including dynamic ones, scenery, information about the performers, and information about the audience. Bagschik, Menzel, and Maurer [26] developed five layers to characterize the essential data. According to Sauerbier et al. [27], a sixth layer of digital information can be added to the five fundamental layers.

2.3 Securing Platforms for V2X Communication

Vehicle-to-everything (V2X) communication platforms are made possible by contemporary vehicular wireless technology, which enables information to be sent between cars at any time, from any location to any network. Despite the advantages, V2X apps nevertheless face significant security and privacy challenges, which is a very reasonable concern, that intrusions in automotive communication networks and applications are prevalent, given a thorough overview of the V2X ecosystem. Also, over key security and privacy concerns, ongoing standardization efforts, and current defence methods that should be put out for the V2X sector. Then outlined potential opens vulnerabilities and discovers semantic gaps in the current security solutions [28]. V2X communication technology is anticipated to

drastically alter the current ground transportation system in the not-too-distant future. V2X applications may be targeted by malicious entities because of the development of this contemporary technology (as demonstrated by recent real-world attacks on automotive systems [29, 30], hence layered defence mechanisms are needed to increase the resilience of these systems. In the survey [31], a general review of the V2X security standards in use today, along with information on potential security risks and various detection techniques were explored. Even though the major focus of this study is on V2X technology, other safety-critical cyber-physical domains could benefit from the unique security techniques created for V2X applications [31].

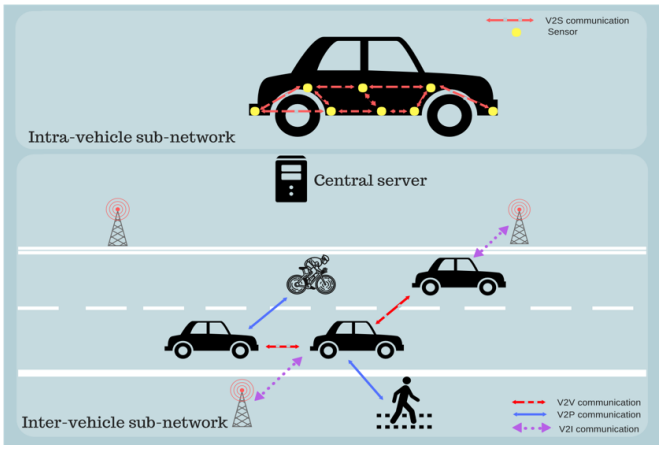


Figure 2.3: General Structure of intelligent transportation system [32]

- Roadside users like roller skaters, motorcycles, bicycles, and pedestrians.
- Roadside Units (RSUs) are pieces of transportation infrastructure that are located alongside roads.
- It contains data about neighbourhood topology that helps in giving road entities many functions.
- All road entities, traffic, and roadways are controlled centrally by a central/cloud server.

2.3.1 V2X Communication Evolution for Security Enhancements

Intelligent and autonomous vehicles are being introduced thanks to the rapid advancements in wireless communications and autonomic driving [33]. Vehicle to Everything (V2X) communications enable real-time road information, traffic efficiency, and driving safety in

vehicular networks. To meet the ever-changing vehicular application, communication, and service demands of connected vehicles, such as ultra-low latency, ultra-high bandwidth, ultra-high dependability, and security, 5G NR V2X, also known as 5G, has evolved. There is a backlash in deployment and management due to scalability, inadequate security, and less flexibility, though intelligent, and autonomous vehicles proliferate, and their safety standards increase. To address the scalability and flexibility challenges, multi-access edge computing (MEC) is crucial in bringing cloud services closer to vehicle nodes. In addition, blockchain has developed into an efficient technology enabler to address a number of security, privacy, and networking concerns that the present MEC systems in automotive networks based on 5G confront.

In addition to MEC, 5G V2X can be managed and secured using blockchain as a powerful security mechanism. Great depth about cutting-edge V2X was employed in this survey [33], including how it has developed using both 802.11bd and 5G cellular technologies for security, privacy protection, and content caching. They investigated integrating blockchain into 5G-based MEC vehicular networks. The problems and difficulties associated with current edge computing and 5G V2X should outlined before shedding some light on potential future research avenues for these converged and developing technologies [33].

2.3.2 V2X COMMUNICATION SYSTEMS AND NEXT-GENERATION NETWORKS

The primary objectives of the new generation of vehicle-to-everything (V2X) communication systems are to improve public safety to reach the accident-free milestone, increase traffic efficiency and reduce carbon emissions for a cleaner environment, improve traffic flow at intersections and on highways, more efficiently manage parking spaces, increase fuel efficiency, and ultimately enable self-driving cars. Due to numerous research and standardization organizations in Europe and the US, two distinct technologies—C-ITS (Europe - ITS-G5) and DSRC (US - WAVE) based on 802.11p—have emerged, both have been thoroughly tested, standardized, and prepared for wide-scale deployment.

Costandoiu and Leba [34] discussed that, a new technology, C-V2X (LTE/Cellular based V2X), is being created as science advances at an ever-increasing rate, and it has begun to eclipse more established and well-proven technologies with less convergence towards 5G. An ideal technology that is scalable and can be develop over time will serve as the basis for connected and automated vehicles. Utilizing both the current LTE infrastructure

and the upcoming 5G mobile network, C-V2X enables communication between its users. In this study [34], a quick overview of the available technologies, show, and asses the standardization landscape, and discuss the capabilities of the technologies now in use. The benefits and cons of old and new V2X technologies are then discussed, with an emphasis on the pressure from various automotive and industry associations to direct the creation and implementation of these V2X communication systems. They also reviewed the ideas for spectrum harmonization.

2.4 HMI applications

In recent years, the market for human-machine interfaces, sometimes known as HMIs, has shown phenomenal growth in terms of popularity. HMIs, which should not be confused with user interfaces (UIs), are primarily geared for applications and uses in the industrial sector. Having said that, this does not in any way rule out the possibility that they are employed for other functions. HMIs can be found in a variety of applications, including commercial and consumer ones, some of which we may be already familiar with. A number of the most reputable automobile manufacturers in the world are currently integrating HMIs into their products [35].

A conventional human machine interface (HMI) for use within a motor vehicle typically takes the form of a touchscreen-enabled user interface. Using this interface, the driver or a passenger can control various aspects of the vehicle, including the temperature control, the audio system, the turn-by-turn navigation, and more. And this is not a trend that will be going away any time soon, as industry analysts expect that more automobile manufacturers will join on board with HMI technology in the years to come. When reduced to its most elemental form, a human machine interface (HMI) is nothing more than an interface through which a human operator operates a machine. Therefore, a human machine interface, or HMI, could basically just be a touchscreen interface that's used to show e-ink text. Electronic HMI displays similar to this one are used rather commonly in workplaces such as offices and other types of business establishments.

2.4.1 Utilizing an ARM processor, implementing an HMI-based smart-mirror

The Human-Machine Interface (HMI) mirror is intelligent and able to communicate with users right away. The impact of intelligent mirrors on future technology that offers services to its customers is notable. The visible mirror, ARM CPU, camera, and LED screen make up the HMI smart mirror project. The ARM Processor can connect to the cloud and using the internet to access data to display on a mirror. The weather forecast, time and date, clock, location data, news feeds, and user data may all be retrieved via a web browser using Python, which offers software attributes and controls the display, as part of the HMI intelligent mirror system in this study [36]. Intelligent mirrors have been created with face and speech recognition technologies for security purposes.

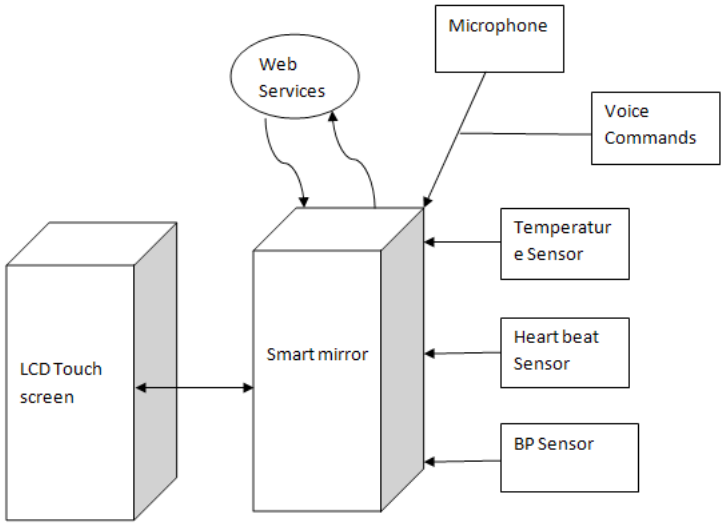


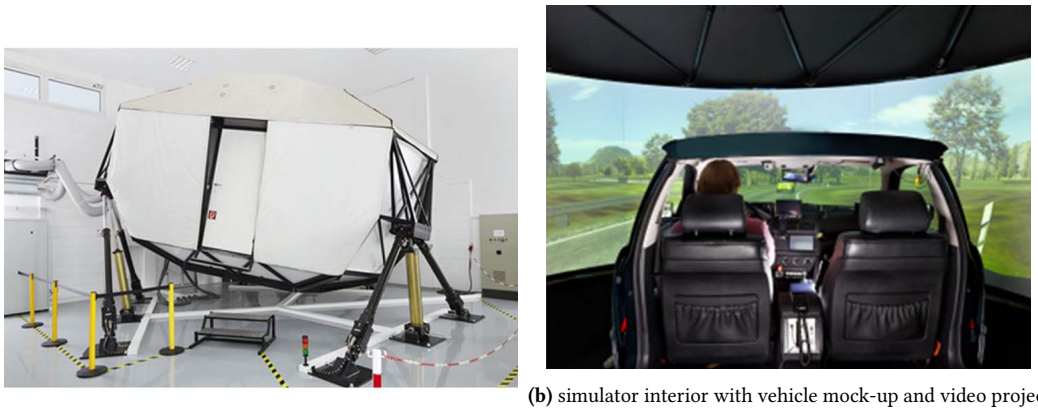
Figure 2.4: Figure Modules on the mirror [37]

The Raspberry Pi 3 has an impact on the HMI-based smart mirror, and the outcome is real-time data feeds that are shown on an LED panel. The facial recognition and speech recognition algorithms used in the HMI are based on CNN.

The smart mirror's architecture offers a setup that may be expanded in the future to accommodate even more capabilities. A designed intelligent mirror with the development of automotive systems and for commercial purposes in mind is vital.

2.4.2 Increasing the Practicality of Automated Driving by Reducing Primary Task Interference with HMI Design

Driving time can be put to other uses when conditionally automated driving (CAD) is in effect (Non-Driving-Related Tasks- NDRTs). A driver may be informed in advance of impending automated manoeuvres, such as lane changes or speed adjustments, to promote safety and comfort during an automated trip, but since completing NDRTs is the driver's main responsibility, they would prefer to be told in a non-distracting manner. In this study [13], the possibility of enhancing human-automation interaction through speech output was investigated. In a motion-based driving simulator, a sample of 17 participants conducted various tasks that required communication between the automation and the driver.



(a) Hexapod movement system

(b) simulator interior with vehicle mock-up and video projection

Figure 2.5: The WIVW dynamic driving simulator [38]

The autonomous driving system's Human-Machine Interface (HMI) was a visual-auditory HMI with either extra voice output or general auditory feedback (i.e., information tones that are used for all machines). Over the course of the drive, the drivers were instructed to complete a standard NDRT. Self-reported visual workload and monitoring of the visual HMI decreased because of communicating forthcoming automated motions in addition by speech as opposed to generic aural output.

Additional speech output did not, however, have an impact on NDRT interruptions [13]. The fact that participants overwhelmingly preferred the HMI with additional speech-based output shows the potential for speech to increase the utility and adoption of driverless vehicles [35]. Whether speech output could enhance human-machine collaboration in the field of CAD was the focus of the current investigation. Enhancing system transparency by communicating impending automated manoeuvres has not yet been thoroughly researched,

even though communication of transitions from CAD to manual driving has drawn significant study interest. Seventeen (17) individuals in all completed the same driving simulator course twice, once with a system that just applied generic audio input ("generic") and once with a system that added speech output in addition to the generic auditory output ("speech + generic"), it was investigated whether adding more speech output will make it easier for humans and machines to work together by effectively alerting the driver about impending automated driving manoeuvres and, as a result, causing less disruption during the execution of an NDRT. The very ability to engage in NDRTs without interruptions may be what makes automated driving practical and appealing [39].

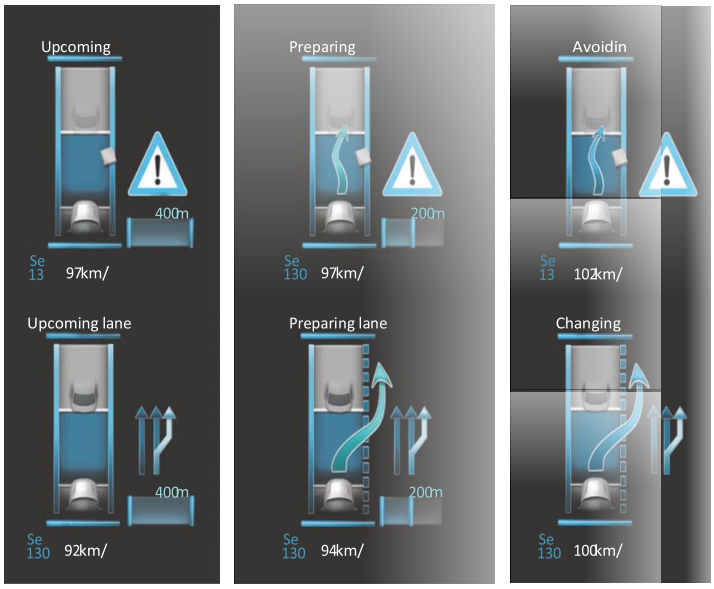


Figure 2.6: HMI for communication of speed limits [38]

The problem is explained to the driver as they get closer to the manoeuvre (Figure 2.5(a)). The same blue colours that are utilized in the regular working state are displayed to make it apparent that no manual intervention is required.

The driver is also informed of the sort of traffic event [40] and the remaining distance to the event. The drive is informed by the preparatory step (Figure 2.5(b)). regarding the precise manoeuvre the system intends to do. The text message above the main state and the blue arrow show that the automation is preparing to perform the manoeuvre. Additionally, a text message and the situation-specific arrow's blue colouring are used to signal the execution of the manoeuvre (Figure 2.6). A Head-Up Display (HUD) shows visual information [13].

2.5 Analysing the software architectures (HCI/HMI) Operations

Several researchers have focused on Human-Computer Interaction or Human-Machine Interaction by constructing or designing software architectures that either make it easier for users to interact with one another or recognize the inputs users provide to produce the appropriate responses. There have been many studies that have utilized these methodologies in a variety of study fields, including healthcare, mobile environments, robots, and other fields [41].

Interaction is recognized as an essential notion, and the work that is put into enhancing it is an important consideration for a wide variety of platforms, systems, and business fields. The purpose of this manuscript is to present a systematic review of the literature to identify, analyse, and classify the published approaches to support or enhance Human-Computer Interaction or Human-Machine Interaction from the perspective of software architectures.

This review will be presented in the context of software architecture. The approach that was used was the systematic review, and it adhered to the principles that are associated with Systematic Literature Reviews methods, such as the one that was proposed by Cruz-Benito, Garcia-Penalvo, and Theron [41]. This research found a total of 39 publications that contained software architectures to either enhance or assess human-machine or human-computer interaction (HCI or HMI). Regarding software architectures, researchers identified three primary approaches: layered architectures, modular architectures, and architectures that were founded on software agents. However, none of these structures was standardized and most of them were ad hoc solutions. The most important interfaces that were discussed are those associated with Graphical User Interfaces (GUIs) as well as multimodal and natural ones.

2.5.1 Python applications

Python is a high-level programming language that has become increasingly popular in recent years. It has a wide variety of applications and may be used in a number of different contexts. Python has a considerable proportion of 27.95% of the programming language market, which places it at the top of the Popularity of Programming Language (PYPL, 2022) rankings as the most popular programming language. In the past five years, Python has experienced the biggest growth, amounting to 11.5%. Python has surpassed SQL to become

the third most popular technology [42]. Python is presently utilised for the majority of the operations carried out by IT organisations. The user is able to develop programmes that are understandable on both a small and a large scale thanks to the language constructs.

Python's ability to support multiple programming paradigms, including object-oriented, procedural, functional, and formal programming, is without a doubt the most astounding feature of this exceptional programming language. In addition to its robust architecture and memory allocation that is handled automatically, Python's standard library is sizable and has been around for a long time. There are Python interpreters available for a number of different operating systems. With this, it has a wide range of applications. Web development frameworks like as Django, Pyramid, Flask, and Bottle, in addition to content management systems such as Plone and Django CMS, can be used to develop web applications in a short time interval. The standard library that comes with Python can be used to manage a wide number of internet protocols, such as HTTPS, FTP, and SSL, as well as the processing of JSON, XML, and Email, amongst other things. Python is used to implement cryptographic functions, which encompass a wide variety of techniques for developing secure applications.

These algorithms can be found in the Python standard library. Users of Python are able to construct clients and servers for connection-oriented and connectionless protocols using the programming language. Python is utilised extensively in the fields of empirical and statistical computing, featuring tools such as SciPy for Engineering, Mathematics and Science, Pandas for data research and forecasting, IPython for efficient editing and recording of work sessions, as well as visual representations and parallel processing. Python is also utilised in the development of Enterprise resource planning (ERP) and eCommerce systems. Odoo is an all-in-one management software for enterprise administration applications, whereas Tryton is a three-tier high-level general-purpose application platform. Tryton was designed to support a wide variety of applications. Everything is an object in the Python programming language.

Object-oriented programming, sometimes known as OOP, is a method that facilitates the intuitive resolution of complex issues. Object-oriented programming (OOP) may be utilised to cut down these massive challenges into more manageable bits.

2.6 Socket Programming

All languages that allow for network communication can be used to write socket programmes, but Python is the most popular choice due to its platform independence, its

exception mechanisms for robustly handling common problems that arise during I/O and networking operations, and its threading facilities for implementing powerful servers [43]. Python's extensive network support is one of its primary strengths as a language for client- server application development. Understanding how internet-based asynchronous communication works requires knowledge of socket programming, but not at the level of the programme that is produced; rather, it must be understood at a higher level that is compiled into a set of socket programmes.

A network is made up of computer systems, each of which can act in one of two roles: client or server. A programme that is providing some service is referred to as a server, while a programme that is making a request for some service is referred to as a client. Clients are personal computers or workstations on which users run applications. Servers are powerful machines or processes that are specialised in handling file servers, printers, or network traffic. Clients are dependent on servers for many resources, including files, devices, and even processing power. When these programmes are run, the end effect is the simultaneous creation of a client process and a server process. These two processes communicate with each other via receiving from and sending to sockets, as shown in Figure 2.7.

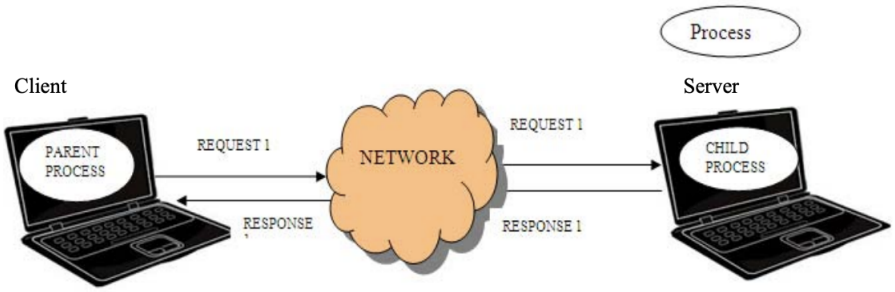


Figure 2.7: Client-Server communication [43]

These sockets serve as the programming interfaces that are made available by the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) protocols for stream and datagram communication, respectively, on the transport layer, which is a component of the TCP/IP stack. Writing code for both the client and the server programmes is the primary responsibility of the developer when it comes to the process of developing a network application. Both the client and the server programmes are created by a single developer or by a development team; however, the individual developer has full editorial control over the content of the code. Because the code does not implement a protocol that

is available to the public, it will not be possible for other independent developers to create code that is compatible with the programme and works together with it. When designing a proprietary application, the developer must exercise extreme caution to ensure that they do not utilise a port number that has already been established and is widely recognised.

2.6.1 TCP and UDP

Since TCP relies on connections between clients and servers, it offers a service that is focused on those connections. When two processes need to link to one another in order to share information, the system is said to be connection-oriented. TCP is reliable since clients always request confirmation of server receipt of data sent [44]. TCP will automatically resend data and wait for a longer amount of time before giving up on getting an acknowledgement if it doesn't receive one. The processes on the various computers exchange data by sending messages across the sockets. The socket of a process is like the door of a house. The socket, depicted in Figure 2.8, is the interface between applications and TCP. While the application developer can dictate nearly every aspect of the socket's application layer, it has much less control over the transport layer of the Open Systems Interconnections (OSI) model.

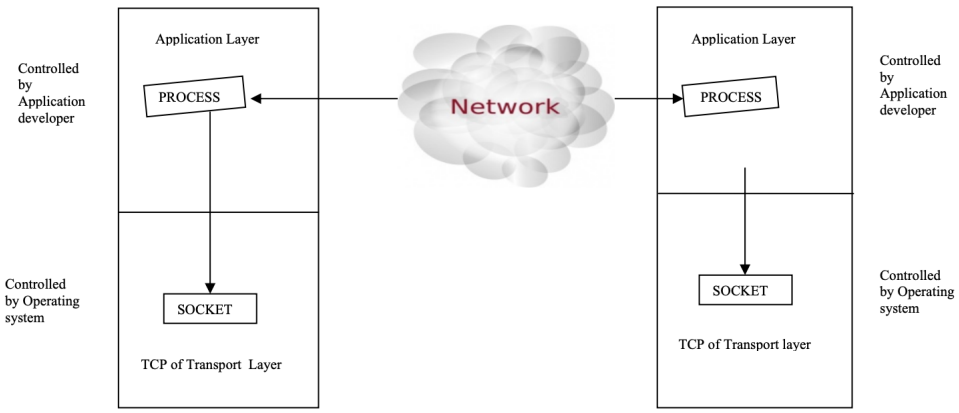


Figure 2.8: Process communicating through TCP sockets [43].

The TCP connection appears to the application as a virtual pipe connecting the sockets on the client and server ends. TCP ensures that the server process will receive (through the connection socket) each byte in the order sent by the client process. Like a door through which people can enter and exit, a connection socket between a client process and a server process can both receive and transmit data. In contrast to the TCP protocol, UDP does

not require the client to initially establish a connection with the server. Instead, the client simply transmits a datagram to the server by using the send to function, which needs the address of the destination as an argument in order to work properly. In a similar manner, the server will not allow a connection to be made from the client. Instead, the server will simply call a function called "recvfrom", which will wait in a holding pattern until data is received from one or more clients. Along with the datagram, the IP address of the client is returned by "recvfrom" to the server.

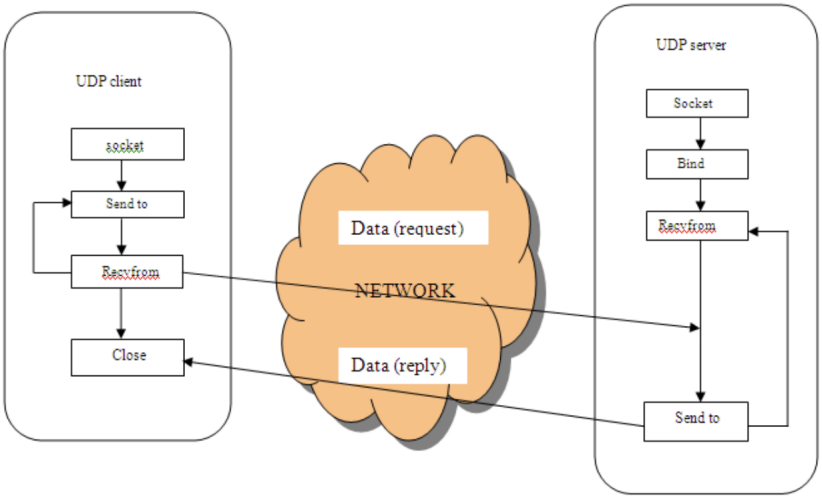


Figure 2.9: UDP client-server [43]

This enables the server to respond to the client in the manner depicted in Figure 2.9. There is no phase that begins with people shaking hands. It is not reliable because when you transmit information or a message, you do not know if it will arrive at its destination; it is possible that it will be lost along the route. During the process of message transfer, there is a possibility of data corruption.

3 OBJECTIVE

The development of an application designed specifically for use on a test track presents itself as a daunting but significant endeavor when considering its purpose: testing advanced driver assistance systems (ADAS) found in vehicles today. These systems, including those that emit warnings when vehicles veer out of their lanes or halt unexpectedly in emergencies- ultimately play pivotal roles in increasing vehicle safety while also decreasing accident rates.

Thus, developing an efficient software system that permits tests of such technological advancements boosts its importance further within the automotive industry. An ideal ADAS test track app warrants user-friendliness at utmost priority; ease-of-use being pivotal to its nature causing undemanding maneuvers on navigation with concise instructions accompanying each function present within it shall prove beneficial towards achieving optimal outcomes from each trial conducted using this app's aid effectively rendering no room for misunderstanding or confusion.

Another critical aspect needs consideration during conducting tests is ensuring maximum safety measures are applicable throughout designated areas where high-speed testing occurs still avoiding any untoward incidents caused by inaccuracies resulting from using such technology driven tool. To sum up; developing an ADAS test track application follows rigorous processes demanded by true functionality allowing accurate implementation of tests in a scenario that replicates real-world driving situations fostering the growth and development of the automotive industry.

3.1 SPECIFIC OBJECTIVE

The specific objectives of this project are:

- Developing an application that can be used on a test track.
  - Verifies the compliance of the performed test with the prescribed vehicle dynamics, reliability, and safety criteria.

  - Graphical, intuitive, user-friendly interface (HMI)
- Guide the test driver and evaluate test results numerically.
  - Using GNSS positioning data (lat, lon, elevation).
  - Using IMU sensor data (acceleration, heading angle).

This is a tool that can be used to assist test drivers on proving grounds, which are specialized facilities where vehicles are put through a series of tests to evaluate their performance and ensure that they meet certain criteria. The system will provide guidance to the test drivers, helping them navigate the proving grounds, and perform the necessary tests according to prescribed procedures.

Additionally, the system will provide evaluations of the test results, highlighting any areas where the vehicle did not meet the prescribed criteria, and suggesting potential improvements or modifications. This can help test drivers to identify issues with the vehicle and take steps to address them, ultimately improving the overall performance and safety of the vehicle.

4 METHODOLOGY

This chapter presents the methodology used in developing an application that verifies the compliance of performed tests with the prescribed vehicle dynamics, reliability, and safety criteria on a test track. The objective of the application is to provide a graphical, intuitive, and user-friendly interface (HMI) that guides the test driver and evaluates test results numerically. The application uses GNSS positioning data (lat, lon, elevation) and IMU sensor data (speed, heading angle) to accomplish its objectives.

The first step in developing the application was to identify the requirements and objectives of the application. This involved researching the prescribed vehicle dynamics, reliability, and safety criteria and understanding how to verify compliance with these criteria on a test track. The team then identified the necessary data that needed to be collected and processed to accomplish the application's objectives.

The second step was to choose the appropriate tools and technologies to build the application. After evaluating various options, the team decided to use Esmini scenario player API and PyQT5 to build the application. Esmini scenario player API is a powerful tool for playing scenarios on a test track, and PyQT5 is a popular and widely used framework for building user interfaces.

The third step was to design and develop the application. This involved creating a user-friendly interface that guides the test driver through the test and provides numerical evaluation of test results. The application collects data from the GNSS positioning system and IMU sensors and compares it to the data from the XOSC file. The application instructs the driver how to proceed and evaluates performance after the test is complete.

The fourth step was to test and evaluate the application. The team conducted multiple tests on a test track to ensure that the application was functioning as intended and accurately evaluating test results. The team also gathered feedback from test drivers and made necessary improvements to the application based on their feedback.

In conclusion, the development of the application involved identifying the requirements and objectives of the application, choosing the appropriate tools and technologies, designing

and developing the application, and testing and evaluating the application. The result is a powerful tool that verifies compliance with prescribed vehicle dynamics, reliability, and safety criteria on a test track, providing a graphical, intuitive, user-friendly interface that guides the test driver and evaluates test results numerically.

4.1 Esmini Scenario Player API

The Esmini Scenario Player API is an open-source software library that provides a flexible and customizable platform for scenario playback. It enables the application to simulate complex scenarios by allowing it to read and interpret xosc files. These files contain the required parameters for creating various scenarios, such as vehicle positions, speeds, and trajectories. The Esmini Scenario Player API offers several features such as scenario looping, variable time-step simulation, and dynamic object control. These features will be utilized to ensure that the application is able to simulate the scenarios accurately and in real-time.

The API is built on top of the Unreal Engine, a popular game engine used in the development of video games. This engine provides the underlying framework for rendering and simulating scenarios in the application. The Esmini Scenario Player API interfaces with the Unreal Engine through a set of well-defined APIs that allow for the creation of custom scenarios.

To integrate the Esmini Scenario Player API into the application, the API will first be downloaded from the official repository and built according to the instructions provided in the documentation. Once built, the API will be integrated into the application using the appropriate software development kit (SDK). The SDK will allow the application to access the API's functions and data structures required to create and run the scenarios.

The Esmini Scenario Player API also provides a rich set of tools for debugging and testing scenarios. The application will leverage these tools to ensure that the scenarios are functioning correctly and that the results are consistent with the expected outcomes. These tools will help in the identification and resolution of issues that may arise during scenario development and testing.

4.1.1 How Esmini works

Esmini is an open-source, event-driven simulation engine that enables users to simulate complex systems and study their behavior. The engine is designed to be modular and extensible, which means that users can easily create their own simulation models based on the Openscenario format and visualise them.

At its core, Esmini uses an event-driven simulation approach, where events are scheduled and executed in a chronological order. The engine maintains a priority queue of events, where each event is associated with a trigger indicating when it should be executed. The events can represent any action or change in the system being simulated, such as the arrival of a new entity, the departure of an entity, or the completion of a task.

During the simulation, the engine iterates through the priority queue, executing the events in order of their timestamps. Each event can modify the state of the system, such as updating the values of variables, adding or removing components, or scheduling new events.

Esmini also supports the concept of entities, which represent the objects or agents in the system being simulated. Each entity has its own state and behavior, and can interact with other entities and the system as a whole. Entities can be created and destroyed dynamically during the simulation, allowing users to model systems with varying numbers of agents.

To model the behavior of entities, users can define their own components, which represent the different aspects of an entity's behavior. For example, a customer entity might have components for its arrival process, its service process, and its departure process. Components can be implemented using a variety of programming languages, C++ and python.

So to summarise, Esmini provides a powerful and flexible platform for simulating complex systems and studying their behavior. Its modular and extensible design allows users to easily create and customize simulation models, while its event-driven simulation approach provides a high degree of accuracy and efficiency.

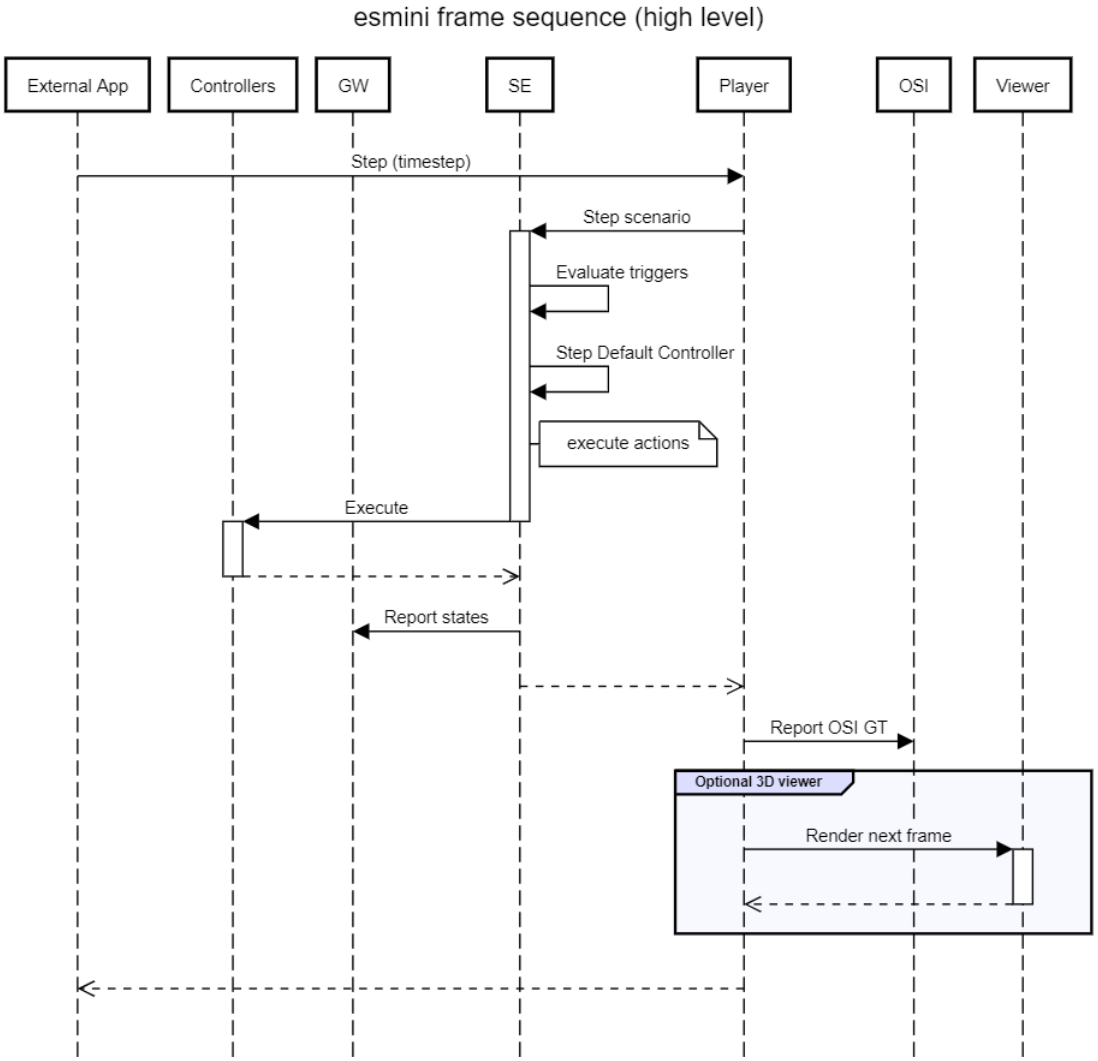


Figure 4.1: How the modules interact [45]

As shown in Figure 4.1, Esmini is a complex software system that consists of several components working together to achieve its intended purpose. These components include:

- Data Ingestion: this component is responsible for ingesting data from various sources such as APIs, databases, and flat files, and storing it in a central data store.
- Data Processing: this component processes the ingested data by performing transformations, cleaning, and normalization, to ensure the data is in a format that can be analyzed.
- Data Analysis: this component applies statistical and machine learning techniques to the processed data to derive insights and patterns that can be used for decision making.

- Reporting and Visualization: this component generates reports and visualizations that provide users with an easy-to-understand representation of the analyzed data.
- Deployment and Monitoring: this component handles the deployment and monitoring of the entire system to ensure it operates smoothly and efficiently.

All of these components work together to create a robust and effective data analysis system that can be used by businesses to gain insights and make informed decisions.

4.1.2 Initialization of the Scenario Engine

The SE_Init() function is a critical component of the esmini scenario player API, responsible for initializing the scenario engine and preparing it to execute a specified OpenSCENARIO (OSC) scenario file. This function takes in several parameters, including the name of the OSC file to load, a flag to disable controllers, and various parameters for the viewer, such as whether to enable off-screen rendering or record a playback of the scenario.

The 'oscFilename' parameter specifies the path to the OSC file to load, which contains the scenario definition and all necessary simulation and environment data. The disable_ctrls parameter is a flag that can be set to '1' to disable any controllers specified in the OSC file, effectively rendering the simulation in a "manual" mode. Alternatively, if this flag is set to 0, controllers will be applied according to the OSC file, allowing for fully autonomous simulation.

The 'use_viewer' parameter is a bitmask that determines how the viewer should be used in the simulation. Setting this parameter to '0' will prevent any viewer from being instantiated, improving performance in scenarios where a viewer is not needed. Other possible values for 'use_viewer' include '1' to instantiate a viewer with a window on the screen, '3' for off-screen rendering only, '7' for off-screen rendering with screenshots saved to file, and '11' for off-screen rendering with info-text disabled for better remote/virtual desktop support.

The 'threads' parameter controls the number of threads to use for simulation. If this value is set to '0', the simulation will run in a single thread. Alternatively, setting this parameter to '1' will allow the viewer to run in a separate thread, running parallel to the scenario engine for improved performance.

Finally, the 'record' parameter is a flag that determines whether a recording of the scenario should be created for later playback. If set to '0', no recording will be created.

However, setting this parameter to '1' will create a recording file that can be used to replay the scenario later.

The 'SE_Init()' function important part of the esmini scenario player API, providing a flexible and powerful interface for initializing and configuring the scenario engine for a wide range of simulation scenarios. This function is an essential component of any simulation project that uses the esmini scenario player, and its behavior and performance should be thoroughly tested and evaluated as part of the development process.

4.1.3 SE_GetObjectState

The SE_GetObjectState function is a method provided by the ESMini Scenario Player API that enables the retrieval of object state information for a specified object. This function takes two parameters, the object_id and a pointer to a SE_ScenarioObjectState structure that will be filled in with the object's state information upon completion.

The SE_GetObjectState function returns a value of 0 if the operation was successful, and -1 if not. This function is particularly useful in scenarios where real-time object state information is required for analysis or visualization purposes.

To use this function, the object_id of the desired object must be passed as a parameter, along with a pointer to an empty SE_ScenarioObjectState structure. Upon completion, the structure will be filled in with the relevant information, including the object's coordinates, orientation, speed, and other characteristics.

The SE_GetObjectState function provides a powerful tool for analyzing and visualizing complex scenarios, enabling real-time access to object state information. Its usage in conjunction with other functions provided by the ESMini Scenario Player API can facilitate the creation of sophisticated simulation environments for a wide range of applications.

4.1.4 SE_ScenarioObjectState structure

The SE_ScenarioObjectState structure is a data structure used in the scenario engine to store the state of objects in the simulated environment. The structure has several member variables, including id, model_id, ctrl_type, timestamp, x, y, z, h, p, r, roadId, junctionId, t, laneId, laneOffset, s, speed, centerOffsetX, centerOffsetY, centerOffsetZ, width, length, height, objectType, objectCategory, wheel_angle, and wheel_rot.

The id member variable is an automatically generated unique identifier for the object. The model_id variable is used to control which 3D model represents the object in the

simulation. The ctrl_type variable specifies the type of controller being used to control the object, with 0 representing the default controller and 1 representing an external controller. The timestamp variable is not currently used, but could potentially be used in the future to interpolate object positions for increased synchronization between simulators.

The x, y, and z member variables represent the global coordinates of the object's position in the simulated environment, while the h, p, and r member variables represent the object's heading, pitch, and roll in the global coordinate system. The roadId and junctionId member variables specify the road and junction that the object is currently on, with -1 indicating that the object is not currently in a junction. The t and laneId member variables represent the lateral position and lane that the object is currently in, while the laneOffset member variable represents the object's lateral offset from the center of the lane.

The 's' member variable represents the longitudinal position of the object on the road, while the speed variable represents the object's speed. The centerOffsetX, centerOffsetY, and centerOffsetZ member variables represent the coordinates of the bounding box center relative to the object reference point in the local coordinate system. The width, length, and height member variables represent the dimensions of the object.

The objectType and objectCategory member variables specify the main type and subcategory of the object, respectively. Finally, the wheel_angle and wheel_rot member variables represent the steering angle and rotation angle of the object's wheel, respectively. The SE_ScenarioObjectState structure is a crucial data structure used in the scenario engine to represent the state of objects in the simulated environment.

4.1.5 External UDPDriverController

The "UDPDriverController" is a tool within the "esmini" simulation environment that allows for an external interface with "driver models" or "vehicle simulators" through a UDP connection. This allows for the control of entities within the simulation to be delegated to an external device or program. To use the "UDPDriverController," it must be assigned to a specific vehicle within the "OpenSCENARIO" file, and input can then be sent to the "esmini" controller to update the vehicle's position and behavior. The external controller bypasses the default controller within "esmini" and allows for the control of a simulated vehicle using a variety of input modes.

There are four different input modes available within the "UDPDriverController." The "DRIVER_INPUT" mode allows for the control of a simple vehicle model in terms of pedals and steering input. The "VEHICLE_STATE_XYZHPR" mode reports the explicit complete

state of the vehicle, including its position, orientation, speed, and wheel angle. This mode is useful for advanced vehicle dynamics simulations where the vehicle may be pitching or rolling as a result of acceleration or steering. The "VEHICLE_STATE_XYH" mode reports the explicit partial state of the vehicle, including its position, orientation aligned to the road geometry, speed, and wheel angle. Finally, the "VEHICLE_STATE_H" mode reports only the minimal explicit state of the vehicle, including its heading, speed, and wheel angle. The "UDPDriverController" also allows for the reception of ground-truth data via OSI over UDP from "esmini" to enable evaluation and analysis of the simulated vehicle's performance. The Figure 4.2 below depicts the concept of the controller.

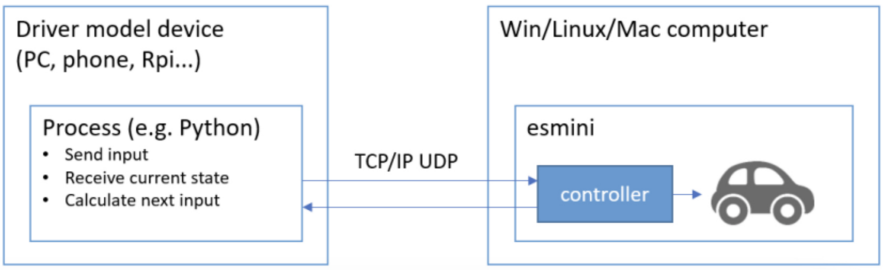


Figure 4.2: UDPDriverController Concept

[45]

So, the "UDPDriverController" is a powerful tool that enables the control of simulated vehicles within the "esmini" environment using external programs or devices. This functionality is essential for the testing and development of autonomous driving systems and for advancing the field of driving simulation research.

4.1.6 OpenSCENARIO file format

The OpenScenario (xosc) file format is an open standard for describing scenarios in driving simulation. It is an XML-based file format that allows users to specify a scenario by defining a series of events, maneuvers, and other environmental factors that can be used in a driving simulation. The xosc standard was developed by the non-profit organization, the Open Simulation and Modeling (OpenSaM) community, to provide a common language for driving simulation tools and to enable interoperability between different simulation tools. The Figure 4.3 depicts the entire OpenSCENARIO format.

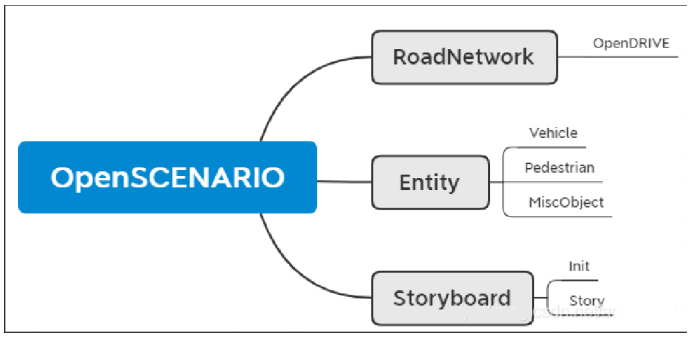


Figure 4.3: OpenSCENARIO format

[46]

The xosc file format provides a structured way to describe scenarios in a driving simulation. The file contains a hierarchical structure of XML elements, which can be organized into different categories such as metadata, road networks, objects, actors, events, maneuvers, and conditions. Each element has a set of attributes that define its properties and behavior.

The metadata element includes general information about the scenario, such as the title, author, and description. The road network element defines the topology of the road network, including lanes, intersections, and traffic signs. The objects element defines the objects that can be used in the simulation, such as buildings, trees, and other environmental features. The actors element defines the actors in the scenario, including the simulated vehicles and pedestrians.

The events element allows users to define specific events that can occur in the simulation, such as a vehicle entering an intersection or a pedestrian crossing the street. The maneuvers element allows users to define specific actions that actors can perform in the scenario, such as turning, changing lanes, or stopping at a traffic light. The conditions element allows users to define specific conditions that must be met for an event or maneuver to occur, such as a minimum speed or distance from another actor.

The xosc file format provides a powerful tool for testing and evaluating autonomous driving systems. In addition to describing the physical environment, the xosc format can also specify the behavior of simulated vehicles and other actors, allowing users to test and evaluate different scenarios and driving behaviors. The xosc file format enables the exchange of driving scenarios between different simulation tools, making it a valuable tool for researchers and developers in the field of autonomous driving.

Components of XOSC Format

The xosc file format consists of various components that work together to create a complete driving scenario.

Storyboard and Entities

At the highest level, an xosc file consists of a storyboard that defines the main actors or entities in the scenario, such as vehicles, pedestrians, and static objects like buildings and trees. Entities are defined using the entity tag and can include various attributes such as id, name, type, position, and orientation. The position and orientation can be defined using either absolute or relative coordinates.

The storyboard also specifies the duration of the scenario, which can be defined in terms of a start time and an end time. Additionally, the storyboard can specify the initial state of each entity at the start of the scenario, such as its position, speed, and other relevant attributes. Figure 4.4 shows the scene of the story the "xosc" file tells.

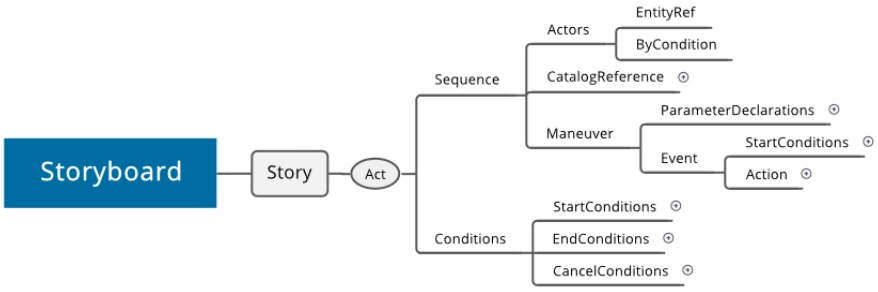


Figure 4.4: Storyboard Scene Description Structure

[46]

ManeuverGroups, Maneuvers, Events, and Actions

Within the storyboard, the behavior of each entity is defined using a series of ManeuverGroups, Maneuvers, Events, and Actions. A Maneuver Group is a collection of related maneuvers, which describe the actions of an entity over a specific period of time. A Maneuver is a specific action or movement that an entity performs, such as driving along a particular route or changing lanes. Each Maneuver can include various parameters such as the speed and acceleration of the entity. Events and Actions are used to trigger specific behaviors in an entity, such as changing its speed or turning its headlights on or off. An

Event is a specific condition that must be met before an Action can be executed. Events can be defined using various parameters such as time, distance, and location. An Action is a specific behavior that an entity performs in response to an Event. Actions can include various parameters such as changing the speed or direction of an entity or triggering an animation.

Conditions and Triggers

Conditions and Triggers are used to specify the circumstances under which a specific behavior should be executed. A Condition is a logical expression that must evaluate to true before an Action can be executed. Conditions can include various parameters such as time, distance, and location, as well as more complex expressions involving the state of other entities in the scenario.

A Trigger is a specific Event that is used to initiate a Maneuver or an Action. Triggers can include various parameters such as time, distance, and location, as well as more complex expressions involving the state of other entities in the scenario.

4.2 PyQt5 Application Framework

PyQt5 is a set of Python bindings for the Qt5 application framework developed by the Qt Company. It allows Python programmers to use the Qt framework for desktop applications with a Pythonic API. PyQt5 provides modules and tools to work with graphical user interfaces (GUI) such as buttons, windows, and dialogs, and also includes support for multimedia, network sockets, and SQL databases.

Qt is a popular cross-platform application framework [47] used for developing software applications with a graphical user interface. It is written in C++ and provides support for multiple platforms including Windows, Linux, and macOS. PyQt5 is a Python wrapper for the Qt framework, which allows developers to use the Qt toolkit with Python, instead of using C++.

PyQt5 provides a rich set of features for GUI development such as buttons, text fields, labels, menus, and dialogs. It also provides support for multimedia applications such as audio and video playback, network sockets for network communication, and SQL databases for data storage and retrieval.

One of the key advantages of PyQt5 is its compatibility with various operating systems. PyQt5 applications can run on Windows, Linux, and macOS, which makes it a popular choice for developing cross-platform applications. PyQt5 also provides support for various GUI design tools such as Qt Designer, which can be used to create and edit forms with graphical widgets, and convert them to Python code.

PyQt5 is licensed under the GNU GPL (General Public License) or commercial license. This allows developers to use PyQt5 for free under the GPL license, or purchase a commercial license for commercial applications. PyQt5 has extensive documentation and a large community of developers contributing to its development, making it a popular choice for creating GUI applications using Python.

In conclusion, PyQt5 is a powerful and flexible toolkit for creating GUI applications in Python [48]. Its compatibility with various operating systems and support for GUI design tools make it a popular choice for developing cross-platform applications. With its extensive documentation and large community of developers, PyQt5 is an excellent choice for developing GUI applications using Python.

4.2.1 PyQt5 Designer

PyQt5 Designer is a graphical user interface (GUI) design tool that comes bundled with the PyQt5 package. It allows developers to create and edit forms using a drag-and-drop interface, without the need for coding.

PyQt5 Designer provides a wide range of widgets such as buttons, labels, text boxes, and menus that can be easily dragged and dropped onto the form. These widgets can be customized by modifying their properties in the property editor. PyQt5 Designer also supports the creation of custom widgets using Python code. These widgets can be added to the form and used like any other built-in widget.

One of the main advantages of using PyQt5 Designer is that it generates Python code for the designed GUI, which can be easily integrated into the application's source code. The generated code can be easily modified to add custom functionality to the GUI. Another advantage of PyQt5 Designer is that it supports the use of style sheets, which are used to customize the appearance of the GUI. Style sheets allow developers to modify the font, color, and layout of the widgets in the GUI, making it easy to create a custom look and feel for the application.

PyQt5 Designer is a powerful tool for developers who want to create GUI applications with PyQt5. Its intuitive drag-and-drop interface, support for custom widgets and style

sheets, and ability to generate Python code make it an essential tool for creating modern and responsive desktop applications.

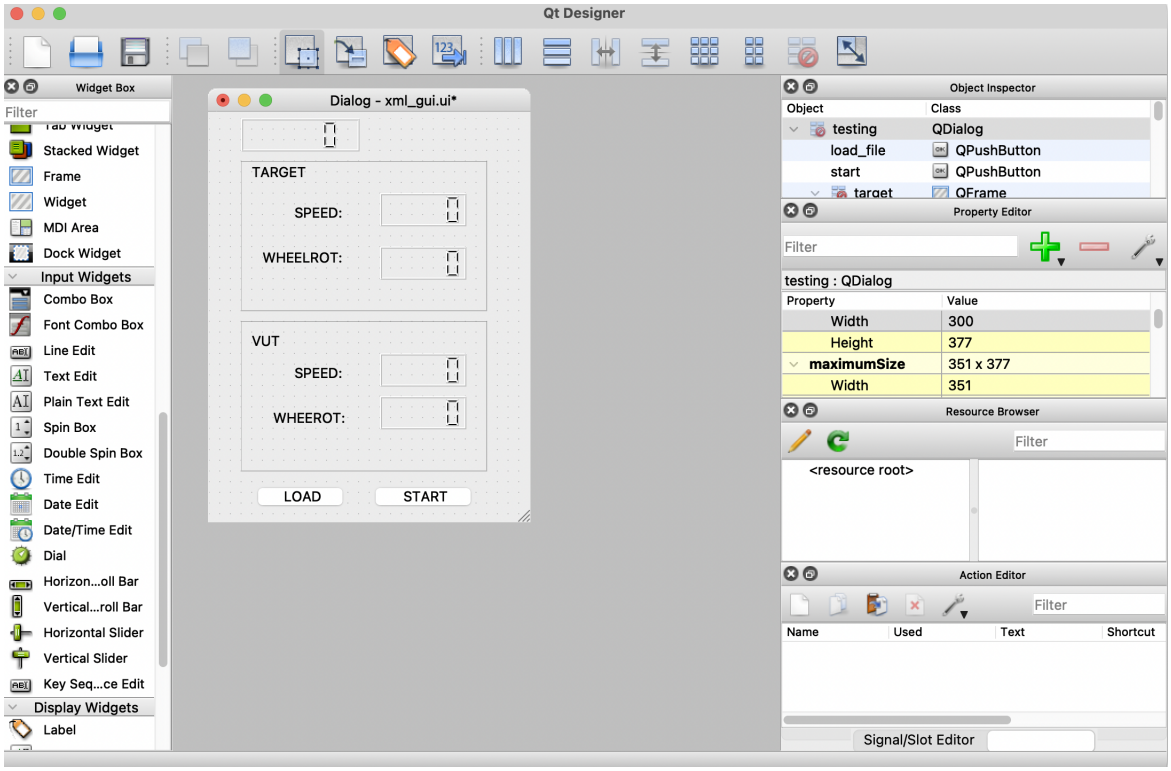


Figure 4.5: Designer interface

4.3 Architecture

The application follows a simple architecture for a demonstration purpose, that is divided into two main parts: the graphical user interface (GUI) and the back-end simulation engine. The GUI is developed using the PyQt5 library, which provides a set of graphical widgets that are used to create the user interface. The GUI is responsible for allowing the user to select a simulation file and for displaying the simulation results.

The back-end simulation engine is implemented in C++ and is wrapped using the ctypes library to be called from Python. The simulation engine is responsible for loading the simulation file and executing the simulation. The simulation engine is also responsible for providing the simulation results to the GUI for display.

The communication between the GUI and the simulation engine is done through a shared variable that contains the path to the simulation file. The GUI sets this variable when the user selects a simulation file, and the simulation engine reads this variable to

load the simulation file. The simulation engine provides the simulation results to the GUI by updating the display widgets on the GUI.

The architecture of the application is simple but effective in providing a user-friendly interface for loading and executing simulations.

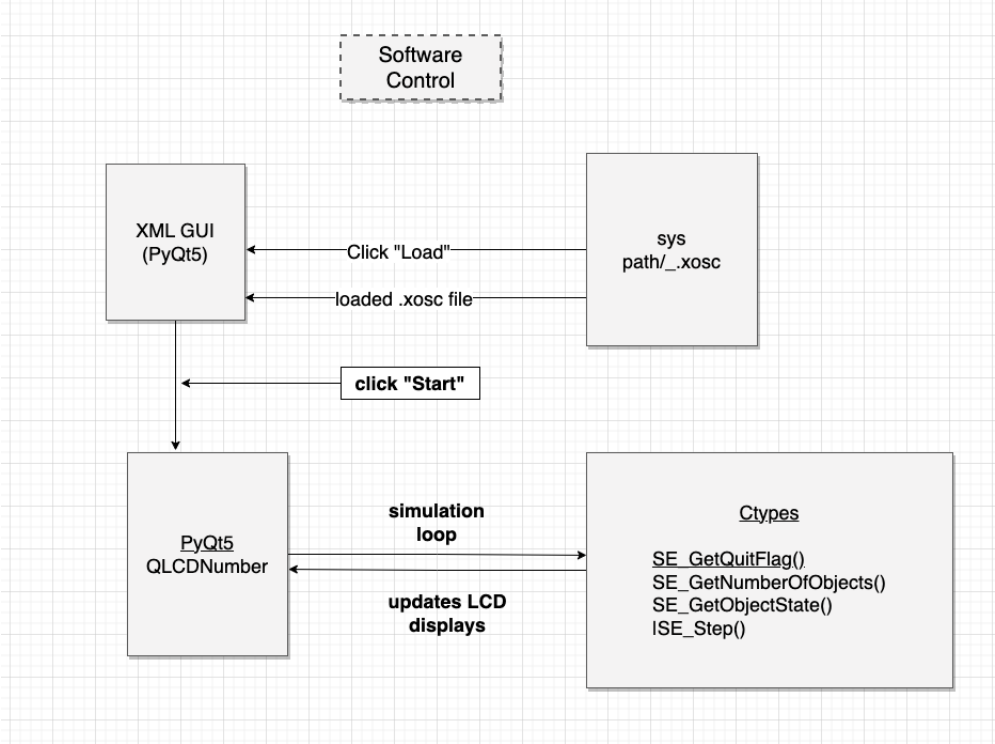


Figure 4.6: Architecture

The LCD updates and the ESMini window will provide information to the test driver about the current state of the simulated scenario. The LCD updates will display information about the current time, speed, and angle of the simulated vehicle(s), while the ESMini window will display the simulated scenario itself, including the positions of the vehicle(s) and any obstacles or road features. This information can help the test driver make decisions about how to interact with the simulated scenario and can provide feedback on the effectiveness of any actions taken.

5 Results

The development of a test track application for advanced driver assistance systems (ADAS) is a challenging but important task that can help improve vehicle safety and reduce the risk of accidents. The objective of this project is to develop an application that can be used on a test track to verify compliance with prescribed vehicle dynamics, reliability, and safety criteria. The application features a graphical, intuitive, user-friendly interface and utilizes GNSS positioning data and IMU sensor data to guide the test driver and evaluate test results numerically.

5.1 Interface Design

The interface design of the testing application is simple yet functional, with a main window that serves as a container for various widgets such as buttons, labels, and LCD numbers. The window has a fixed size of 300 x 377 pixels, with a minimum size of 300 x 377 pixels and a maximum size of 351 x 377 pixels.

The background of the window is set to a light grey color (#9A9996) using Cascading Style Sheets (CSS). This helps to differentiate the interface from the rest of the operating system and ensures that the user's focus remains on the application. The window also has a title bar with the text "Dialog" displayed.

The application contains two push buttons, "LOAD" and "START", that are positioned at the bottom-left and bottom-center of the window, respectively. The "LOAD" button has an image of a file folder next to the text, while the "START" button has an image of a play button. Both buttons are easily visible and accessible, and their functions are self-explanatory.

The main widget in the interface is a QFrame named "target". This frame contains several labels and LCD numbers that display the current status of the application. The "target" frame is positioned in the center of the window and has a raised border that gives it a

3D look. The frame is also styled using the "StyledPanel" frame shape to create a subtle shadow effect around the edges.

Within the "target" frame, there are four labels and two LCD numbers. The first label, named "SPEED:", is positioned in the top-left corner of the frame and displays the text "SPEED:". The second label, named "STEERING:", is positioned just below the first label and displays the text "STEERING:". The remaining two labels, named "TARGET" and "DISTANCE", are positioned at the top-right and bottom-right corners of the frame, respectively. The "TARGET" label displays the text "TARGET", while the "DISTANCE" label is left blank.

The two LCD numbers, named "t_speed" and "t_angle", are positioned next to the "SPEED:" and "STEERING:" labels, respectively. These LCD numbers display the current speed and wheel rotation angle, and update in real-time as the application runs. The "t_speed" LCD number is also positioned next to a smaller label that displays the text "KM/H", while the "t_angle" LCD number is positioned next to a smaller label that displays the text "DEG".

In a nutshell, the interface design is clean, simple, and functional. The use of contrasting colors and clear labeling make it easy for the user to understand the application's functionality and current status. The buttons are placed in easily accessible positions, and the use of the "target" frame helps to keep the user's focus on the most important information in the application. The Figure 5.2 shows the user friendly GUI describe above.

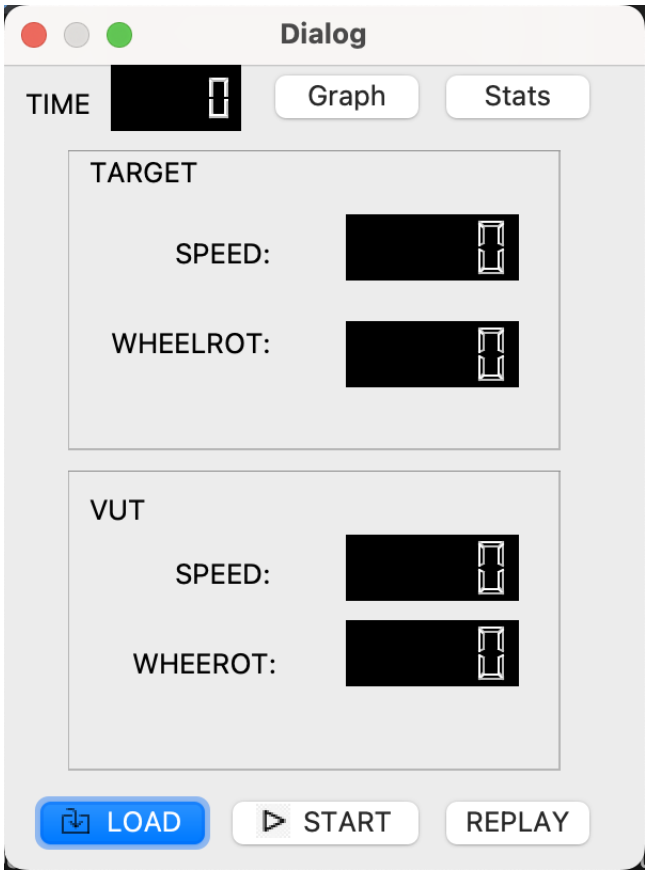


Figure 5.1: PYQT5 Interface

Once the application is executed and the "START" is pressed, the Esmini API engine generates a second graphical user interface (GUI) that appears on the screen. This GUI is specifically designed to aid the driver by providing detailed instructions and guidance. The interface is displayed in a user-friendly format, allowing the driver to easily understand the information and follow the directions provided. The accompanying Figure 5.2 demonstrates how the second GUI looks like and how it provides support to the driver during the journey.

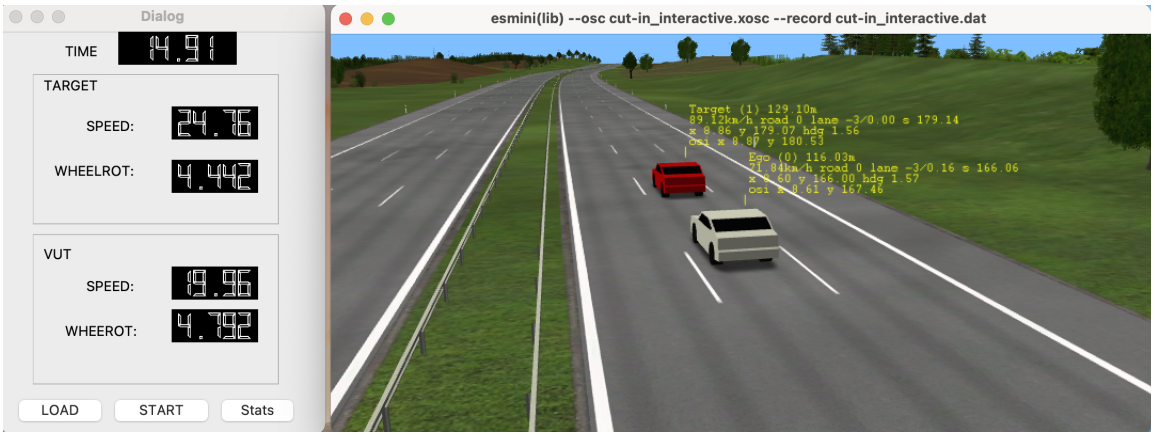


Figure 5.2: Interface With Guidance

5.2 Scenario

For the purposes of demonstration, the xosc file utilized in this project was sourced from the Emini Resource folder, specifically the "cut-in_interactive.xosc" file. However, certain modifications were made to the file in order to accommodate the specific scenario that we aimed to showcase. The starting positions of the involved vehicles were altered to ensure that they all commenced from the same point, which was a necessary adjustment for the purposes of the demonstration.

Although this application is designed to be versatile and can collect data from any OpenScenario format scenario file, the aforementioned modifications were necessary in order to mimic the conditions of a test driver on a test track who is attempting to follow a target vehicle and perform a series of maneuvers. To facilitate this, keyboard control was incorporated into the application, allowing the user to simulate the actions of a test driver in a controlled and repeatable manner. This approach ensured that the demonstration accurately represented the capabilities of the application in a real-world context.

Furthermore, the control of the vehicle under test (VUT) is customizable and can be defined within the parameters of the openscenario format. In real-world testing scenarios, data from the VUT can be transmitted over User Datagram Protocol (UDP) to control the ego vehicle in the GUI guidance. This enables the simulation of complex driving scenarios that are representative of real-world situations.

5.2.1 Cut-in scenario

The "cut-in_interactive.xosc" scenario file contains a storyboard that demonstrates basic maneuvers and the use of an interactive controller. The scenario involves the user controlling a white car that is being overtaken by a red car.

The storyboard is divided into two parts: initialization and a story called "CutInAndBrakeStory." In the initialization phase, the cars are placed on the road and the interactive controller is activated for the white car. The white car is then set to start at a distance of 50 meters from the beginning of the road, and its speed is set to 60 km/h.

In the "CutInAndBrakeStory" story, the red car begins to overtake the white car. The red car is set to cut in front of the white car and brake suddenly. The maneuver group responsible for this sequence is called "CutInAndBrakeSequence." The "CutInManeuver" maneuver within this group has two events: "OverTakerStartSpeedEvent" and "CutInEvent."

In the "OverTakerStartSpeedEvent," the red car starts to accelerate towards the white car at a rate that is 1.2 times faster than the white car. This event is triggered when the "CutInAndBrakeAct" starts its transition. The "CutInEvent" event causes the red car to change lanes in front of the white car. This event is triggered when the "CutInStartCondition" is met.

The scenario also includes parameter declarations for the host vehicle (white car), target vehicle (red car), and various headway times. The road network is defined using a logic file and a scene graph file, while the entities in the scenario are defined using a vehicle catalog and a controller catalog.

In the revised version, in order to mimic the conditions of a test track, the starting positions of the ego vehicle and the target vehicle were adjusted so that they both begin at the same point. This change was made to ensure that the test application accurately reflects the scenarios that would be encountered in real-world testing.

5.2.2 Scenario Data Collection

Gathering scenario data is an essential part of numerous research projects, especially those that focus on simulations or modeling. This process involves collecting data to develop scenarios for testing hypotheses or assessing the performance of models and simulations. Typically, scenario data collection includes identifying relevant variables and parameters, choosing suitable data sources, and establishing data collection protocols. In this study, Python, Pandas, and the CSV module were utilized for scenario data collection.

The application recorded the scenario data in a CSV file named mylog.csv, containing columns for time, object ID, s, x, y, heading, and speed.

The the application's while loop persistently logged scenario data until the SE_GetQuitFlag() function returned True. Within the loop, the SE_GetNumberOfObjects() function determined the number of objects in the scenario, and the SE_GetObjectState() function obtained each object's state. The object state was then written to the CSV file using the csv.writer() function access through pandas dataframe.

The log file served as a comprehensive record of the collected scenario data, which was used for further analysis and modeling. For instance, the data was employed to generate visualizations of the scenario or to create predictive models of object behavior. The CSV file format facilitated easy data importation into other tools or programs for additional processing and analysis.

A significant advantage of using Python and the CSV module for scenario data collection is the flexibility and user-friendliness these tools offer. Python is a popular programming language well-suited for data analysis and modeling, while the CSV module enables simple and efficient data logging to a CSV file.

This method is particularly beneficial for projects involving simulations or modeling, as it allows researchers to effortlessly collect and analyze vast amounts of data. Another benefit of using Python and the CSV module for scenario data collection is the ability to customize the data collection process to align with the project's specific needs. For example, the application in this study could be adapted to collect extra data points or log data at varying intervals. This adaptability enables researchers to tailor the data collection process to their project's unique requirements, ensuring the collected data is relevant and valuable. The Python-generated log file provided an in-depth record of the collected scenario data, which was used for further analysis and modeling. For example, the data was employed to generate visualizations of the scenario or to create predictive models of object behavior. The CSV file format facilitated easy data importation into other tools or programs for additional processing and analysis.

However, there are potential limitations to using Python and the CSV module for scenario data collection. One limitation is that the data collection process might be constrained by the capabilities of the simulation or modeling software in use. For instance, if the software lacks access to specific variables or parameters, it may be impossible to collect all the necessary data for the project. Despite these potential limitations, Python and the CSV

module remain powerful and effective tools for scenario data collection in a wide range of projects. The flexibility and ease of use provided by these tools make them an ideal choice for researchers needing to collect and analyze large quantities of data for their projects.

Scenario data collection is a vital step in many research projects, particularly those involving simulations or modeling. Using Python and the CSV module for scenario data collection offers a flexible and efficient approach for collecting and analyzing large amounts of data. While there may be some potential limitations to this approach, the advantages of using Python and the CSV module for scenario data collection make it a suitable choice for numerous projects. By carefully selecting relevant variables and parameters, appropriate data sources, and developing data collection protocols, researchers can utilize scenario data collection to create scenarios for hypothesis testing, model or simulation evaluation, and informed decision-making [49].

Python CSV Module

The CSV module in Python is widely recognized and utilized for its extensive functionality in reading and writing CSV files. CSV files serve as a popular file format for storing tabular data, and Python's CSV module offers a convenient way to handle and manipulate this type of data.

One of the key advantages of the Python CSV module lies in its simplicity. The module provides straightforward functions that facilitate reading and writing CSV files. Its user-friendly interface makes it an ideal choice for beginners who are just beginning to work with CSV files, allowing them to quickly grasp the essentials

Flexibility is another notable characteristic of the Python CSV module. It offers a wide range of options for reading and writing CSV files, including support for different delimiters, quoting styles, and line endings. This adaptability enables developers to seamlessly work with CSV files that possess diverse formats or have been generated by various programs.

Efficiency is a crucial aspect of the Python CSV module. It is designed to effortlessly handle large CSV files, making it a powerful tool for managing big data. The module also provides options for performance optimization, such as buffering and chunking, which contribute to expediting data processing tasks.

In addition to its simplicity, flexibility, and efficiency, the Python CSV module also incorporates functions for parsing and formatting CSV data. These functions facilitate the conversion of CSV data into other data formats, such as lists or dictionaries. This feature

proves particularly valuable when working with data that requires further processing or analysis using alternative methodologies.

The Python CSV module benefits from comprehensive and accessible documentation available online. This documentation includes numerous examples illustrating the usage of the module, as well as detailed explanations of its functions and options. This wealth of information enables developers to swiftly learn and leverage the module, as well as effectively troubleshoot any potential issues.

These some examples of the functions provided by the Python CSV module, showcasing its versatility and usefulness:

- reader(): this function allows for reading data from a CSV file and iterate over its rows. It returns an iterable object that provides access to each row as a list of values. One can easily extract and manipulate the data within each row, making it convenient for data analysis or processing tasks.
- writer(): with the writer() function, One can create a CSV writer object that enables the possibility to write data to a CSV file. A list of values can be passed to the writer object for each row, and it will handle the formatting and writing of the data. This function simplifies the process of creating and writing CSV files with desired data.
- DictReader(): similar to reader(), the DictReader() function allows reading data from a CSV file. However, instead of returning rows as lists, it returns each row as a dictionary, where the keys correspond to the column names and the values represent the cell values. This function is particularly useful when working with CSV files that have headers, as it provides a convenient way to access and manipulate data based on column names.
- DictWriter(): the DictWriter() function complements DictReader() by providing a way to write data to a CSV file using dictionaries. One can pass a dictionary representing each row, with the keys mapping to the column names and the values containing the data. This function simplifies the process of writing structured data to a CSV file, especially when working with dictionaries or structured data sources.
- csv.fieldnames: this attribute holds the column names (or field names) of a CSV file. When using DictReader() or DictWriter(), this attribute can be used to access and manipulate the column names. It provides a convenient way to programmatically work with the headers of a CSV file.

To summarize, the Python CSV module stands as a powerful and versatile tool for handling CSV files. Its simplicity, flexibility, and efficiency render it an optimal choice for various scenarios involving data collection and processing. Whether a novice or an experienced programmer, the Python CSV module proves invaluable for numerous data-related tasks, solidifying its position as an indispensable asset in any programming toolkit.

Python Pandas Module

The Python Pandas module has gained widespread popularity as a crucial library for developers seeking efficient and versatile data manipulation and analysis capabilities. It serves as an indispensable resource for data scientists and analysts working with extensive datasets, enabling them to tackle complex data analysis tasks seamlessly.

One of the prominent advantages of the Pandas module lies in its ability to effortlessly handle large datasets. Through the employment of powerful data structures like Series and DataFrame, Pandas exhibits exceptional prowess in managing extensive volumes of data. This characteristic renders it an optimal choice for processing big data and undertaking intricate data analysis tasks. Furthermore, Pandas equips developers with functions tailored to handle missing data, effectively addressing a common challenge encountered when dealing with large datasets.

Flexibility emerges as another key asset of the Pandas module. Its rich repertoire encompasses an extensive range of functions dedicated to data manipulation and transformation. Filtering, sorting, and grouping data become effortlessly attainable, enabling complex data transformations and facilitating the extraction of valuable insights. Merging and joining data, a common necessity when working with data from multiple sources, are seamlessly accomplished through Pandas' provisions.

The module also extends its support to various data formats, including CSV, Excel, and SQL databases. This versatility simplifies the import and export of data from diverse sources, facilitating the manipulation of data in different formats. Additionally, Pandas boasts specialized functions for handling time series data, catering to scenarios where data changes over time.

Furthermore, the Pandas module boasts extensive documentation, ensuring a comprehensive resource for developers. This documentation comprises illustrative examples and detailed explanations of functions and options, empowering users to swiftly familiarize themselves with the module and effectively troubleshoot any encountered issues.

The strength of the Pandas module is further amplified by its vibrant community of developers. This community fosters collaboration, offering invaluable support to fellow developers while contributing to the continued advancement of the module. Developers leveraging the Pandas module can tap into this vast pool of knowledge, benefiting from a wealth of resources and insights.

The Python Pandas module stands as an indispensable and robust tool for data manipulation and analysis in Python. Its flexibility, efficiency, and user-friendly nature make it the go-to choice for data scientists and analysts working with substantial datasets and embarking on intricate data analysis endeavors. Regardless of one's skill level, the Pandas module proves to be a valuable asset in the developer's arsenal. Its array of functions streamlines data manipulation and analysis, while its adaptability accommodates diverse data sources and formats. By embracing the Pandas module, developers unlock a realm of possibilities in managing and deriving insights from their data.

5.2.3 Logged Data Structure

The logged data structure plays a crucial role in capturing and organizing the recorded data for analysis and further investigation. In this section, we will examine the structure of two key files, 'mylog.csv' and 'diff_df'. Understanding the data organization within these files is essential for comprehending the data processing and analysis procedures.

mylog.csv Structure

The mylog.csv file serves as a repository for the recorded data, encapsulating relevant information about various parameters at specific timestamps for ojects in the simulation. In this case, (ObjId0) and target vehicle (ObjId1) given time. It adheres to a structured format comprising distinct columns representing different data attributes. Figure 5.3 shows an example of how mylog.csv looks like.

	Time	Obj Id	S	X	Y	Heading	Speed
	0.49	0	50	8.17	49.97	1.57	0
	0.49	1	50	8.17	49.97	1.57	0
	1	0	50.01	8.17	49.98	1.57	1.1
	1	1	50.01	8.17	49.98	1.57	0.95
	1.52	0	50.86	8.18	50.83	1.57	10.4
	1.52	1	50.97	8.18	50.94	1.57	12.12
	2.02	0	52.95	8.18	52.92	1.57	19.4
	2.02	1	53.43	8.19	53.4	1.57	22.92
	2.54	0	56.42	8.2	56.39	1.57	28.69
	2.54	1	57.54	8.2	57.51	1.57	34.08
	3.06	0	61.23	8.21	61.2	1.57	38
	3.06	1	63.26	8.22	63.23	1.57	45.25
	3.57	0	67.38	8.24	67.35	1.57	47.3
	3.57	1	70.59	8.25	70.56	1.57	56.4
	4.09	0	74.85	8.27	74.81	1.57	56.59
	4.09	1	79.5	8.29	79.47	1.57	67.56
	4.61	0	83.35	8.3	83.32	1.57	60.52
	4.61	1	89.69	8.33	89.65	1.57	72.26
	5.12	0	92.58	8.33	92.54	1.57	68.37
	5.12	1	100.37	8.38	100.33	1.57	74.56

Figure 5.3: mylog.csv example

Delving into the specific components of this structure:

- Time: This column denotes the precise timestamp at which the data entry was logged, facilitating temporal analysis and synchronization with other measurements.
- ObjId: The ObjId (0 or 1) column records the identification number of the object or entity under observation. It enables the differentiation of multiple entities within the recorded data and facilitates targeted analysis on individual entities.
- s, x, y: These columns capture the spatial attributes of the observed objects, specifically their position coordinates. The s value represents the object's relative position, while x and y indicate the corresponding Cartesian coordinates.
- Heading: The heading column corresponds to the orientation or heading of the object, providing valuable insights into its directional alignment or orientation.
- Speed: The speed column signifies the object's velocity at a given timestamp. It enables velocity-based analysis and facilitates the evaluation of dynamic aspects associated with the observed entities.

By incorporating these columns and their associated data, the mylog.csv file establishes a comprehensive structure to represent and archive the recorded data in an organized manner.

diff_df Structure

In addition to the mylog.csv file, the diff_df file offers further insights into the recorded data by presenting the differential values between consecutive data points of objects, in this case the ego vehicle (ObjId0) and target vehicle (ObjId1) given time. This structure allows for the analysis of variations and changes over time, aiding in the identification of patterns and trends. Figure 5.4 shows an example of how diff_df looks like.

Time	Diff_s	Diff_x	Diff_y	Diff_heading	Diff_speed
0.49	0	0	0	0	0
1	0	0	0	0	0.15
1.52	0.11	0	0.11	0	1.71
2.02	0.48	0	0.48	0	3.51
2.54	1.12	0	1.12	0	5.39
3.06	2.03	0.01	2.03	0	7.25
3.57	3.21	0.02	3.21	0	9.09
4.09	4.65	0.02	4.65	0	10.97
4.61	6.33	0.04	6.33	0	11.74
5.12	7.79	0.05	7.79	0	6.19
5.64	7.99	0.06	7.99	0	3.1
6.16	6.86	0.08	6.86	0	11.64
6.66	5.41	0.09	5.41	0	10.63
7.17	3.74	0.11	3.74	0	12.99
7.69	1.78	0.13	1.78	0	13.37
8.2	0.07	0.01	0.07	0.04	10.68
8.7	1.22	0.55	1.22	0.03	8.16
9.22	2.11	0.76	2.1	0.01	3.03
9.74	1.75	0.88	1.74	0.01	7.63
10.24	0.1	0.93	0.09	0.01	13.27

Figure 5.4: diff_df.csv example

Exploring the components of this structure:

- Time: similar to the mylog.csv file, the diff_df file includes the Time column to establish temporal coherence and facilitate cross-referencing with both vehicles as this denotes the exact time both vehicle data was logged.
- diff_s, diff_x, diff_y: these columns capture the differential values in the spatial attributes (s, x, and y) between successive data points of both ego and target vehicle given the same time. They provide essential information regarding spatial changes and movements of the observed entities.
- diff_heading: the diff_heading column represents the differential change in the heading attribute, allowing for a more detailed understanding of the rotational dynamics of both entities.

- diff_speed: finally, the diff_speed column quantifies the variation in speed between the ego and target vehicle given the same timestamps, enabling the analysis of acceleration or deceleration patterns.

By encapsulating these differential values, the diff_df file augments the data structure and provides a valuable resource for investigating changes and trends within the recorded data.

This comprehensive analysis of the logged data structure demonstrates the importance of organizing and structuring data in a systematic manner. The mylog.csv file captures essential information about vehicle attributes at specific timestamps, while the diff_df file focuses on the differential changes between successive data points of vehicles considering time. Understanding these structures is crucial for conducting thorough data analysis and drawing meaningful insights from the recorded data.

5.2.4 Data Preprocessing Between Logged Files

Data preprocessing is a crucial step in extracting meaningful insights and patterns from raw data. In the context of logged files, efficient data preprocessing techniques are essential for transforming unrefined data into a format suitable for analysis. This subtopic delves into the Python Pandas module and its role in facilitating the preprocessing of data between logged files.

Effective data preprocessing lays the foundation for accurate and reliable analysis. By cleansing and transforming data, we can eliminate inconsistencies, handle missing values, and prepare the data for further exploration. Preprocessing also involves organizing data in a structured manner, ensuring it aligns with the desired analytical objectives.Figure 5.5 depicts a typical process involved in data preprocessing.

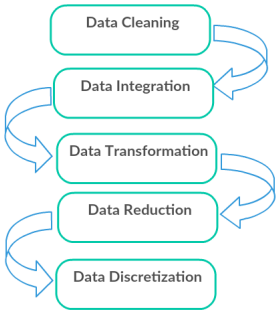


Figure 5.5: Data Preprocessing

For our purpose, that is, to preprocess the logged data, we took the following process:

Reading the Data
The first step in the preprocessing workflow involves reading the logged data into a Pandas DataFrame. This data structure provides a tabular representation of the data, making it convenient for subsequent operations. The 'read_csv()' function allows seamless ingestion of CSV files, which are commonly used for storing logged data.

Splitting Data Based on Object IDs
To facilitate comparative analysis between different objects in the logged files, the data is split into separate DataFrames based on their respective object IDs. This separation allows for focused analysis and targeted preprocessing on individual objects.

Preparing the Time Column
The time column plays a crucial role in temporal analysis. To ensure proper alignment and consistency, the time column is extracted from the DataFrame and treated as a separate entity. This step enables easy manipulation and indexing based on time, facilitating subsequent calculations and comparisons.

Removing Unnecessary Columns
Certain attributes may not contribute significantly to the analysis or may introduce unwanted noise. By removing these unnecessary columns, we streamline the dataset, ensuring that subsequent calculations focus on the relevant attributes for comparison.

Calculating Differences
One of the primary goals of preprocessing logged files is to compute differences between objects based on specific attributes. By iterating through the DataFrames and comparing attribute values, we obtain the absolute differences for attributes such as position, heading, and speed. These differences are then recorded in a new DataFrame for further analysis.

Generating Summary Statistics
To gain a comprehensive understanding of the differences, summary statistics provide valuable insights. The Pandas module offers the 'describe()' function, which produces statistical measures such as mean, standard deviation, minimum, and maximum values.

These statistics offer a concise summary of the differences and aid in identifying trends and patterns within the data.

Saving the Preprocessed Data
Once the preprocessing steps are complete, the preprocessed data is saved to a new CSV file using the 'to_csv()' function. This ensures that the processed data is accessible for subsequent analysis or for sharing with collaborators, facilitating re-producibility and collaboration.
The process above is shown in the Figure 5.6 below.

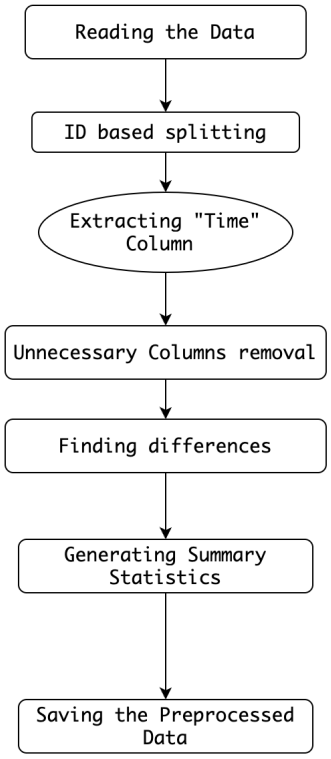


Figure 5.6: Process of Preprocessing

5.3 Some Application Features

5.3.1 Supported platforms

The application is designed to be cross-platform compatible, with support for Windows, macOS, and Linux operating systems. While the target system for the application is Rasp-

berry Pi, it can be run on other platforms with just a few extra configurations.

One of the key benefits of having a cross-platform compatible application is that it allows users to access the application regardless of their preferred operating system. This is particularly useful for organizations that have a mix of operating systems in their environment. By having a single application that can run on different platforms can be helpful in terms of finances.

Another benefit of having a cross-platform compatible application is that it allows users to take advantage of the strengths of different operating systems. For example, Windows is known for its compatibility with a wide range of hardware and software, while macOS is known for its user-friendly interface and security features. By having a cross-platform compatible application, users can choose the operating system that best suits their needs and preferences.

However, developing a cross-platform compatible application can be challenging. Different operating systems have different APIs, libraries, and dependencies, which can make it difficult to ensure that the application runs smoothly on all platforms. Additionally, optimizing the application for different architectures, such as ARM for Raspberry Pi, can require additional development effort.

To overcome these challenges, we used cross-platform development frameworks and tools, such as the Esmini library, Python and PyQt5, which are designed to be cross-platform compatible. These tools provide a consistent development environment across different platforms, making it easier to develop and maintain cross-platform applications.

In terms of system requirements, the application has different minimum requirements depending on the platform. For Windows, the minimum system requirements are Windows 7 or later, 2 GB of RAM, and 500 MB of free disk space. For macOS, the minimum system requirements are macOS 10.12 or later, 2 GB of RAM, and 500 MB of free disk space. For Linux, the minimum system requirements are a 64-bit processor, 2 GB of RAM, and 500 MB of free disk space.

While the target system for the application is Raspberry Pi, it can be run on other

platforms with just a few extra configurations. For example, to run the application on Windows or macOS, users may need to change targets of certain libraries and get the full the Esmini scenario player API. However, the application is designed to be cross-platform compatible, which means that it can be run on different platforms without requiring significant modifications.

Concluding, the application is designed to be cross-platform compatible, with support for Windows, macOS, and Linux operating systems. Cross-platform compatibility offers several benefits, including access to the application regardless of the user's preferred operating system and the ability to take advantage of the strengths of different operating systems. However, developing a cross-platform compatible application can be challenging, and developers may need to use cross-platform development frameworks and tools to ensure optimal performance. The application has different minimum system requirements depending on the platform, and while the target system is Raspberry Pi, it can be run on other platforms with just a few extra configurations.

5.3.2 Application Display

The application display is designed to be user-friendly and intuitive, with a clean and modern interface. The main display is used for selecting scenario files, playing scenarios, and displaying updates of vehicle speed of target and test vehicle, as well as the scenario time as shown Figure 5.2.

The main display have several buttons, each with its own purpose. The first is the scenario selection button, which allows users to select the scenario file they want to play. The scenario files are targeted by their extesnion, that ".xosc" , making it easy for users to find the scenario they need. Once a scenario file is selected, the user can click the Start button to start the scenario.

The second thing to talk about is the section, which displays the scenario time and the updates of vehicle speed of target and test vehicle. The scenario time is displayed in minutes and seconds, and it updates in real-time as the scenario plays. The vehicle speed updates are displayed in km per hour (kph) and are updated every second.

The application display is designed to be visually appealing and easy to navigate, with clear labels and tooltips. The buttons are organized logically, with related buttons grouped together. The button operation is consistent throughout the application, ensuring that users can easily navigate and use the different features. The buttons are designed to be

responsive and provide feedback to the user, ensuring that the user knows when an action has been completed.

Concluding, the application display is designed to be user-friendly and intuitive, with a clean and modern interface. The main display is used for selecting scenario files, playing scenarios, and displaying updates of vehicle speed of target and test vehicle, as well as the scenario time. The display is designed to be visually appealing and easy to navigate, with clear labels and tooltips. The button operation is consistent throughout the application, ensuring that users can easily navigate and use the different features.

Various Windows

Main Window

The main window serves as the central hub and primary interface through which users engage with the application, facilitating seamless interaction and providing access to a multitude of essential features and controls. As the cornerstone of the user experience, the main window is thoughtfully designed to offer a comprehensive and intuitive platform for users to navigate and leverage the application's functionalities.

At first glance, the main window captures the user's attention with its well-organized layout and visually appealing design. Its strategic placement of key features and controls ensures easy accessibility, enabling users to swiftly initiate their desired actions. Prominently positioned buttons catch the user's eye, inviting them to effortlessly load scenarios, initiate simulations, or access critical functions.

One of the central components of the main window is the real-time data display, which provides users with up-to-the-moment information regarding the scenario at hand. This dynamic feature showcases vital metrics, such as vehicle speed, position, heading, and other relevant parameters, in real time. By presenting this data in a clear and concise manner, users can readily monitor and assess the ongoing scenario, making informed decisions and adjustments as needed.

In addition to the real-time data display, the main window offers seamless navigation to other windows and essential functionalities. Intuitive menu options or tabbed interfaces allow users to effortlessly access additional windows, enabling them to delve deeper into specific aspects of the application's capabilities. Whether it be reviewing detailed performance analysis, configuring advanced settings, or conducting in-depth data exploration, the main window provides a user-friendly gateway to these critical features.

The design of the main window prioritizes simplicity and ease of use. By employing intuitive icons, clear labels, and logical grouping of controls, the interface fosters a seamless user experience. This approach ensures that both novice and experienced users can quickly grasp the application's functionality and effortlessly navigate through various features. This is shown in Figure 5.2

Scenario Window

The Scenario Window stands as an integral and indispensable component that plays a pivotal role in guiding the test driver to execute the scenario with utmost precision and accuracy. Its existence is owed to the innovative Esmini API, which empowers this window to deliver a highly immersive and interactive experience. By seamlessly integrating visual elements, the Scenario Window offers a comprehensive and real-time representation of the target vehicle, as well as the test vehicle (also known as the ego vehicle) that diligently follows it.

The primary purpose of the Scenario Window is to provide a visual guidance system that ensures the correct execution of the scenario. Through a meticulously designed user interface, this window furnishes the test driver with crucial information and cues, enabling them to navigate the scenario with confidence and adherence to predefined parameters. By displaying both the target vehicle and the ego vehicle in a synchronized manner, the Scenario Window establishes a visual connection that aids in accurately replicating real-world scenarios.

Through the Esmini API, the Scenario Window harnesses the power of real-time data to deliver an immersive experience. The continuous exchange of information between the virtual environment and the window facilitates seamless updates, enabling the test driver to make informed decisions and responses based on the dynamic nature of the scenario. The synchronized movement of the ego vehicle in relation to the target vehicle provides invaluable visual cues, ensuring that the test driver maintains the desired trajectory, speed, and relative position throughout the scenario.

One of the key strengths of the Scenario Window lies in its ability to convey complex information in a visually intuitive manner. By leveraging cutting-edge graphics and visualization techniques, this window offers a highly realistic representation of the scenario, fostering a sense of immersion and enhancing the driver's situational awareness. From the meticulously modeled vehicles to the accurately simulated surroundings, every element within the Scenario Window is designed to facilitate a seamless and engaging experience.

This shown in Figure 5.7

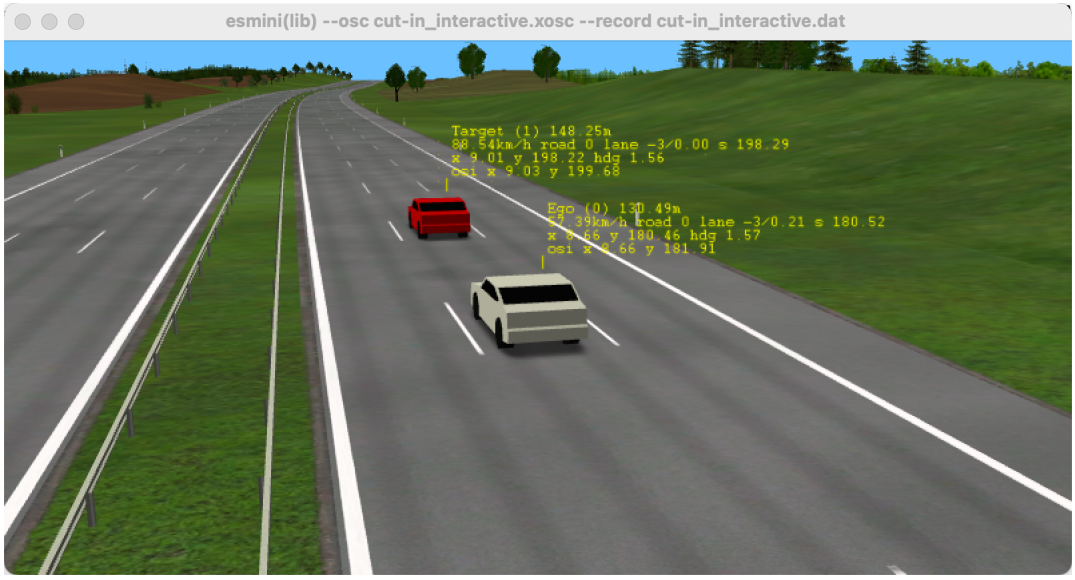


Figure 5.7: Navigation Window

Stats Window

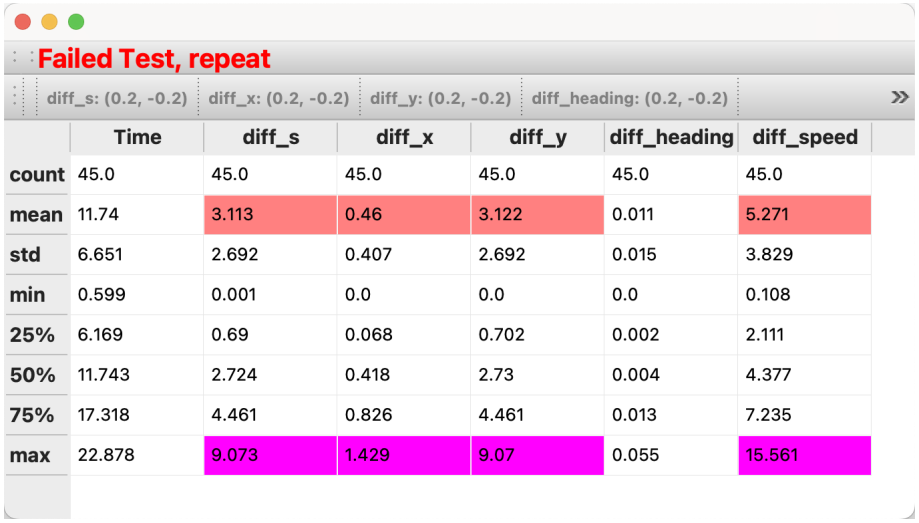
The stats window serves as a valuable resource for obtaining detailed insights into the performance of the vehicle throughout a specific scenario. Its purpose is to present comprehensive and informative statistics derived from the logged data, enabling a thorough analysis of the vehicle's behavior and performance over time.

This meticulously designed window provides users with a wealth of information that encompasses various aspects of the scenario. By leveraging advanced statistical techniques, the stats window generates summary statistics that encapsulate key performance metrics and offer a concise overview of the vehicle's behavior.

One of the primary elements showcased within the stats window is the summary statistics derived from the logged data. These statistics are carefully computed based on the temporal aspect, taking into account the progression of the scenario over time. By focusing on this temporal dimension, the stats window captures essential patterns and trends, providing valuable insights into the vehicle's performance evolution.

Within the summary statistics, a range of crucial indicators is presented, offering a comprehensive understanding of the vehicle's behavior. To ensure the clarity and readability of the presented information, the stats window organizes the summary statistics in a structured manner. Importantly, the design of the stats window prioritizes user experience, ensuring that the displayed information is presented in a manner that is comprehensible

to a wide range of users. The use of clear and concise language, accompanied by intuitive visualizations, fosters effective communication and promotes collaboration among team members with varying levels of technical expertise. This window also gives access to the logged data as shown in Figure 5



	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
count	45.0	45.0	45.0	45.0	45.0	45.0
mean	11.74	3.113	0.46	3.122	0.011	5.271
std	6.651	2.692	0.407	2.692	0.015	3.829
min	0.599	0.001	0.0	0.0	0.0	0.108
25%	6.169	0.69	0.068	0.702	0.002	2.111
50%	11.743	2.724	0.418	2.73	0.004	4.377
75%	17.318	4.461	0.826	4.461	0.013	7.235
max	22.878	9.073	1.429	9.07	0.055	15.561

Figure 5.8: Stat Window

Log Window

In the graphical user interface (GUI) of the logging system, the logged data is presented in a dedicated window, providing a comprehensive overview of the recorded information. The design of this window takes into consideration the importance of parameter thresholds, such as speed, distance, position, and other relevant metrics. By incorporating threshold-based visualization techniques, the displayed data allows for intuitive interpretation and immediate identification of data points that fall within or exceed the predefined thresholds.

Within this carefully designed display, various parameters are represented using different visual cues and color coding to enhance clarity and highlight important information. Specifically, the values associated with each parameter are visually distinguished using a color scheme that assists in the distinction between values that are below or equal to the set threshold and those that exceed it. This approach ensures that users can quickly and accurately identify instances where certain parameters cross the defined limits, warranting further attention and analysis.

To facilitate this visual representation, the GUI employs a color scheme that assigns green and red colors to specific data points based on their relationship to the threshold. Values that are below or equal to the set threshold are displayed in a calming shade of

green, indicating that they fall within the acceptable range. On the other hand, values that surpass the established threshold are represented in a vibrant shade of red, instantly drawing attention to instances where the parameter exceeds the desired limit.

By adopting this color-coded system, the logged data window offers a user-friendly interface that enhances data comprehension and expedites decision-making processes. Researchers and operators can effortlessly identify potential anomalies or critical situations by visually scanning the displayed values. This intuitive approach eliminates the need for manual calculations or extensive data processing, allowing users to focus on interpreting and responding to the identified thresholds in a timely and efficient manner.

In addition, the utilization of this color-coded visualization technique ensures that the display remains user-centric and easily understandable, even for individuals who may not possess an extensive technical background. The color distinctions offer a clear and intuitive representation of the data, fostering effective communication and facilitating collaboration among team members. Figure 5.9 gives an illustration.

Log Data

diff_s: 2, -2 | diff_x: 0.2, -0.2 | diff_y: 0.2, -0.2 | diff_heading: 2, -2 | »

	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
1	0.599	0.001	0.000	0.000	0.000	0.108
2	1.102	0.154	0.001	0.154	0.000	1.897
3	1.616	0.566	0.002	0.566	0.000	3.812
4	2.116	1.223	0.005	1.223	0.000	5.569
5	2.619	2.101	0.009	2.101	0.000	7.056
6	3.128	3.161	0.015	3.161	0.000	8.438
7	3.636	4.461	0.023	4.461	0.000	10.288
8	4.146	6.029	0.034	6.029	0.001	12.103
9	4.644	7.808	0.050	7.807	0.001	13.636
10	5.153	9.073	0.068	9.070	0.001	4.474
11	5.653	9.055	0.085	9.053	0.002	4.526
12	6.169	7.841	0.103	7.839	0.002	9.161
13	6.685	6.829	0.126	6.827	0.003	6.584
14	7.187	5.806	0.155	5.803	0.003	8.722
15	7.699	4.644	0.189	4.642	0.004	7.078
16	8.185	3.814	0.229	3.812	0.004	6.291
17	8.703	3.233	0.276	3.230	0.002	1.816

Figure 5.9: Log Data Window

Graph Window

The speed profile graph displayed in this window offers a comprehensive visualization of the velocity patterns exhibited by both the target vehicle and the test vehicle. This graph serves as a valuable tool for analyzing and comparing the speed profiles of these two entities, providing essential insights into their behavior and performance.

By plotting the speed data over time, the graph allows for a detailed examination of how the velocities of the target and test vehicles evolve throughout a specific scenario. This visualization facilitates a comparative analysis, enabling researchers and operators to identify similarities, differences, and any noteworthy trends in their speed profiles.

The graph is precisely designed to ensure clarity and readability, employing visually appealing elements that aid in understanding the data. The time axis is clearly labeled, allowing for precise temporal analysis, while the speed axis accurately represents the velocity values. These design choices contribute to a user-friendly interface that promotes efficient interpretation of the displayed information.

Distinctive lines or curves representing the speed profiles of the target and test vehicles are plotted on the graph. Each line is thoughtfully color-coded and accompanied by clear legends, making it easy to distinguish between the two vehicles. This visual distinction facilitates a direct and intuitive comparison, enabling users to identify potential discrepancies or similarities in their speed behaviors. Figure 5.10 shows an example of the plot.

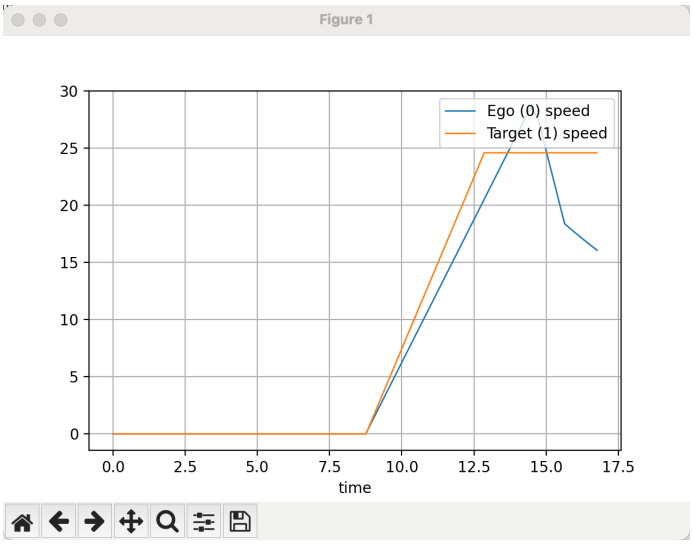


Figure 5.10: Speed Profile Window

5.3.3 Button Operation

The user interface of this system incorporates several intuitive buttons that enable seamless interaction and control. These buttons serve specific functions and play a vital role in facilitating various operations and tasks.

Load Button

The Load button is a fundamental element of the user interface, seamlessly integrated to facilitate the effortless retrieval and integration of the scenario file, specifically the xosc file, within the system. Its primary purpose is to provide users with a convenient means of accessing and loading the designated file from their computer's storage.

Upon clicking the Load button, a user-friendly interface is presented, allowing users to navigate through their computer's file directory system. With ease, users can locate and select the desired scenario file, the xosc file, which encapsulates the essential information and parameters defining the simulated environment or scenario.

The Load button serves as an interactive gateway, connecting the user's intention to the system's functionality. Its unobtrusive design and intuitive placement within the user interface ensure its visibility and accessibility, guiding users towards a seamless experience. The button's clear labeling, accompanied by a familiar and recognizable icon, contributes to its usability, eliminating any ambiguity and enabling users to swiftly identify its purpose.

Start Button

The Start button serves as a fundamental component within the user interface, allowing users to initiate and commence the simulation process seamlessly. Its primary purpose is to trigger the execution of a simulated scenario, enabling users to observe and analyze the behavior and performance of the system under specific conditions.

When users interact with the Start button, they activate a series of intricate mechanisms that set the simulation in motion. This button embodies the essence of control, granting users the ability to initialize the system's functionality and observe the dynamic outcomes that unfold as the simulation progresses.

Replay Button

The Replay Button serves as a valuable feature within the system, offering users the ability to revisit and analyze simulations that have already been performed. This functionality enhances the user experience by allowing them to delve deeper into the intricacies of a simulated scenario and gain further insights from the recorded data.

Graph Button

The Graph button plays a pivotal role in granting users access to a powerful visualization tool called the Graph window, as depicted in Figure 5.10

Stats Button

The stats button give access to the Stats window. This window shown in Figure 5.8

6 Evaluation

When evaluating an application, it is important to conduct a comprehensive assessment to ensure that it performs as intended and meets the necessary requirements. In this case, the evaluation involved four distinct scenarios: two aimed at testing the application's success and two designed to assess its failure based on some threshold that was chosen.

These scenarios were carefully crafted to challenge the application's capabilities in various testing situations that falls within the scope of the application, ensuring a comprehensive evaluation of its performance. During the testing process, certain limitations of the application were identified, highlighting areas where improvements could be made to enhance its performance.

The success scenarios tested the application's ability to play different scenarios. The application demonstrated a high level of proficiency in these situations. Also, the ability to assess a scenario's correctness was quite good.

The failure scenarios tested the application's response to unexpected parameters that are in the test scenario. The application struggled to adapt to unexpected changes in traffic patterns and struggled to recognize other objects other than a target and ego vehicle.

To address these limitations, proposed improvements in scenario generation to meet requirements. The other limitations came from the Esmini API and also the ability to set a proper threshold for a particular test.

The evaluation process was critical in identifying the application's strengths and areas for improvement. By addressing the identified limitations, the application's effectiveness and reliability can be enhanced, ultimately leading to a more efficient test experience. The evaluation process highlights the importance of thorough testing and continuous improvements in the development of the application.

Requirement for Test

These requirements were chosen arbitrarily for general purpose of this test. For one's specific needs and requirements, the threshold values can be changed by developer to

serve the purpose they want. With that said, this are the values for the threshold:

- diff_s: the difference in longitudinal position between the ego vehicle and the target vehicle should be within the range of (2, -2) km/h. The mean difference should be within (0.2, -0.2) km/h.
- diff_x: the difference in lateral position (x-axis) between the ego vehicle and the target vehicle should be within the range of (0.2, -0.2) km/h. The mean difference should be within (0.2, -0.2) km/h.
- diff_y: the difference in lateral position (y-axis) between the ego vehicle and the target vehicle should be within the range of (0.2, -0.2) km/h. The mean difference should be within (0.2, -0.2) km/h.
- diff_heading: the difference in heading angle between the ego vehicle and the target vehicle should be within the range of (2, -2) degrees. The mean difference should be within (0.2, -0.2) degrees.
- diff_speed: the difference in speed between the ego vehicle and the target vehicle should be within the range of (1, -1) km/h. The mean difference should be within (0.2, -0.2) km/h.

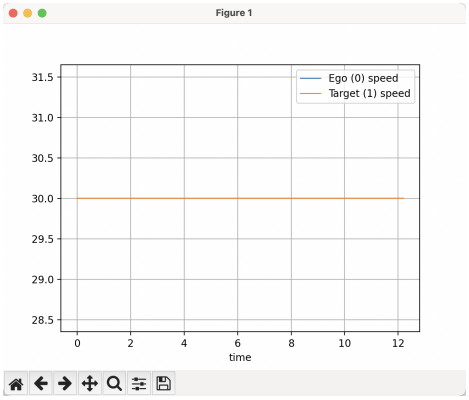
These threshold requirements represents the acceptable ranges of differences and mean differences between the ego vehicle and the target vehicle for a test case to pass. If the actual values fall within these ranges, the test case will be considered successful. However, if any of the values exceed the specified thresholds, the test case will fail.

6.1 Successful Test Case

6.1.1 Constant Speed From Start (automatic control)

In this test case, both the ego vehicle and the target vehicle were subjected to an evaluation of their performance in maintaining a constant speed throughout the scenario. The objective was to assess the application's ability to regulate and control the vehicles' speeds accurately and consistently.

The test began with both vehicles starting at a speed of 30 km/h, as prescribed by the scenario. The application, operating in automatic control mode, utilized its advanced algorithms and sensor data processing capabilities to monitor and adjust the vehicles' speeds in real-time.



(a) Speed Profile

PASSED: Perform next test

diff_s: (0.2, -0.2) diff_x: (0.2, -0.2) diff_y: (0.2, -0.2) diff_heading: (0.2, -0.2)

	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
count	32.0	32.0	32.0	32.0	32.0	32.0
mean	8.41	0.0	0.0	0.0	0.0	0.0
std	4.741	0.0	0.0	0.0	0.0	0.0
min	0.587	0.0	0.0	0.0	0.0	0.0
25%	4.489	0.0	0.0	0.0	0.0	0.0
50%	8.407	0.0	0.0	0.0	0.0	0.0
75%	12.327	0.0	0.0	0.0	0.0	0.0
max	16.25	0.0	0.0	0.0	0.0	0.0

(b) Summary Result

Figure 6.1: Pass Test Case 1 Statistics

Throughout the scenario, the application provided continuous feedback on the vehicles' speed profiles through its graphical and user-friendly interface. This allowed the test driver to visualize the speed variations and evaluate the application's effectiveness in maintaining the desired constant speed.

The analysis of the test results revealed that both the ego vehicle and the target vehicle successfully maintained a consistent speed of 30 km/h throughout the entire scenario. The application demonstrated its capability to accurately control the vehicles' speeds, ensuring compliance with the prescribed criteria. Figure 6.1a shows the speed profile and Figure 6.1b shows the summary statistics of the performed simulation.

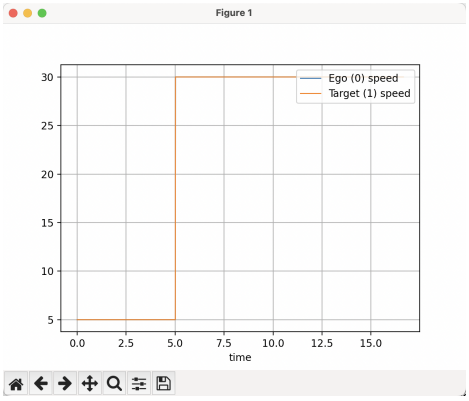
6.1.2 Swift Speed Change (automatic control)

The scenario involved both vehicles starting at a speed of 5 km/h and gradually increasing their speeds to a target value of 30 km/h.

The test scenario was executed using a predefined storyboard, which controlled the actions and behaviors of the vehicles. The application utilized its advanced control algorithms and sensor data processing capabilities to ensure the ego vehicle closely followed the speed profile of the target vehicle.

The test began with both the ego vehicle and the target vehicle positioned on the same lane of the test track. The application initiated the control process by activating the controllers for both vehicles and setting their initial speeds to 5 km/h. This allowed the vehicles to start moving at the same initial speed.

As the test progressed, the application continuously adjusted the speed of the ego vehicle to match the target vehicle's speed. The application's control actions, guided by



(a) Speed Profile

PASSED: Perform next test

diff_s: (0.2, -0.2) diff_x: (0.2, -0.2) diff_y: (0.2, -0.2) diff_heading: (0.2, -0.2)

	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
count	23.0	23.0	23.0	23.0	23.0	23.0
mean	6.171	0.0	0.0	0.0	0.0	0.0
std	3.434	0.0	0.0	0.0	0.0	0.0
min	0.597	0.0	0.0	0.0	0.0	0.0
25%	3.394	0.0	0.0	0.0	0.0	0.0
50%	6.164	0.0	0.0	0.0	0.0	0.0
75%	8.96	0.0	0.0	0.0	0.0	0.0
max	11.744	0.0	0.0	0.0	0.0	0.0

(b) Summary Result

Figure 6.2: Pass Test Case 2 Statistics

the predefined storyboard, followed a stepwise dynamics pattern, gradually increasing the speed of both vehicles.

Through the evaluation of the test results, it was observed that the application successfully facilitated the ego vehicle in accurately mimicking the speed of the target vehicle. The ego vehicle steadily increased its speed in synchronization with the target vehicle, ultimately reaching the target speed of 30 km/h.

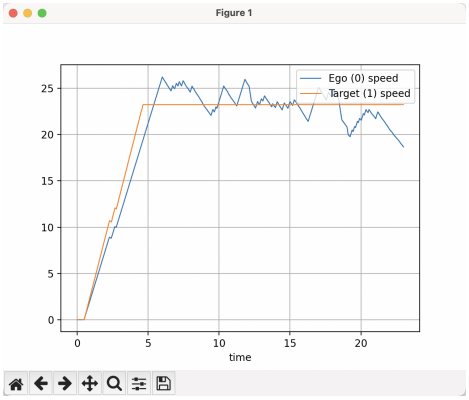
The successful execution of this test case demonstrates the application's effectiveness in enabling the ego vehicle to mimic the speed profile of the target vehicle. This capability is crucial in various scenarios, such as maintaining a safe distance or replicating specific driving patterns. Figure 6.2a shows the speed profile and Figure 6.2b shows the summary statistics of the performed simulation.

6.2 Failure Test Case

6.2.1 Cut-In and Braking (manual control)

In this failure test case, the objective was to assess the application's performance when the ego vehicle was under manual control instead of automatic control. The scenario involved the ego vehicle being manually driven with a target speed set at 60 km/h.

To simulate manual control, the application deactivated the longitudinal control actions and relied on the driver's inputs to determine the speed of the ego vehicle. The target vehicle was positioned ahead of the ego vehicle on the same lane of the test track.



(a) Speed Profile

Failed Test, repeat

	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
count	45.0	45.0	45.0	45.0	45.0	45.0
mean	11.74	3.113	0.46	3.122	0.011	5.271
std	6.651	2.692	0.407	2.692	0.015	3.829
min	0.599	0.001	0.0	0.0	0.0	0.108
25%	6.169	0.69	0.068	0.702	0.002	2.111
50%	11.743	2.724	0.418	2.73	0.004	4.377
75%	17.318	4.461	0.826	4.461	0.013	7.235
max	22.878	9.073	1.429	9.07	0.055	15.561

(b) Summary Result

Figure 6.3: Fail Test Case 2 Statistics

At the beginning of the test, both vehicles were teleported to their initial positions on the test track. The ego vehicle's controller was activated, and its initial speed was set to 60 km/h. Unlike in the successful test cases, where the application controlled the ego vehicle to mimic the target vehicle, in this test case, the ego vehicle's speed was solely dependent on manual input.

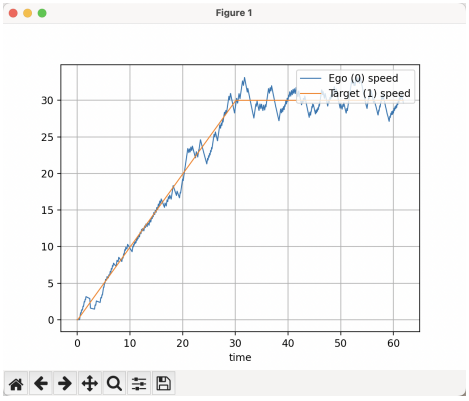
Throughout the test scenario, the ego vehicle's driver manually controlled the acceleration and braking, trying to maintain the target speed of 60 km/h. The application did not intervene or provide any guidance or assistance to the driver, as it would in automatic control mode.

The evaluation of the test results revealed several challenges and limitations associated with manual control. The ego vehicle's speed fluctuations, deviations from the target speed, and inconsistent driving patterns were observed. These shortcomings indicate the difficulty of maintaining precise speed control manually and the inherent limitations of human driving behavior. Figure 6.3a shows the speed profile and Figure 6.3b shows the summary statistics of the performed simulation.

6.2.2 Matching Acceleration and Constant Speed (manual control)

The scenario involved the ego vehicle being manually driven with a target speed set at 50 km/h.

To simulate manual control, the application deactivated the longitudinal control actions and relied on the driver's inputs to determine the speed of the ego vehicle. The target



(a) Speed Profile

Failed Test, repeat

diff_s: (0.2, -0.2) diff_x: (0.2, -0.2) diff_y: (0.2, -0.2) diff_heading: (0.2, -0.2)

	Time	diff_s	diff_x	diff_y	diff_heading	diff_speed
count	45.0	45.0	45.0	45.0	45.0	45.0
mean	11.74	3.113	0.46	3.122	0.011	5.271
std	6.651	2.692	0.407	2.692	0.015	3.829
min	0.599	0.001	0.0	0.0	0.0	0.108
25%	6.169	0.69	0.068	0.702	0.002	2.111
50%	11.743	2.724	0.418	2.73	0.004	4.377
75%	17.318	4.461	0.826	4.461	0.013	7.235
max	22.878	9.073	1.429	9.07	0.055	15.561

(b) Summary Result

Figure 6.4: Fail Test Case 2 Statistics

vehicle was positioned ahead of the ego vehicle on the same lane of the test track.

At the beginning of the test, both vehicles were teleported to their initial positions on the test track. The ego vehicle's controller was activated, and its initial speed was set to 50 km/h. Unlike in the successful test cases, where the application controlled the ego vehicle to mimic the target vehicle, in this test case, the ego vehicle's speed was solely dependent on manual input.

Throughout the test scenario, the ego vehicle's driver manually controlled the acceleration and braking, attempting to maintain the target speed of 50 km/h. The application did not intervene or provide any guidance or assistance to the driver, as it would in automatic control mode. The evaluation of the test results revealed several challenges and limitations associated with manual control.Figure 6.4a shows the speed profile and Figure 6.4b shows the summary statistics of the performed simulation.

6.3 Limitations

Despite its capabilities, the application has certain limitations that should be taken into consideration. These limitations include the following.

Manual Control Interface

The application currently utilizes a keyboard interface for manual control inputs. However, controlling the vehicle solely through keyboard inputs can be challenging and may not provide the most intuitive user experience. Users may find it difficult to precisely manipulate the vehicle's movements and perform complex maneuvers. Future improvements could focus on introducing alternative control interfaces, such as a game pad or a more user-friendly graphical user interface (GUI) for navigation, that Esmini Api do not provide, to enhance the ease and accuracy of manual control.

Multiple Windows

The application operates within multiple windows, which can be a limitation in terms of user experience and visual integration. This limitation arises from the fact that the Emini API used by the application does not support seamless display integration. As a result, the user may need to manage multiple windows simultaneously, which can be cumbersome and potentially affect the overall user experience. Consideration should be given to optimizing the display integration to provide a more streamlined and immersive user interface.

7 Conclusion

The project aimed to develop an application that can be utilized on a test track to verify compliance with prescribed vehicle dynamics, reliability, and safety criteria. The application's specific objectives included the creation of a graphical and user-friendly interface, guiding the test driver during tests, and evaluating test results numerically using GNSS positioning data and IMU sensor data.

The developed application successfully achieved these objectives and proved to be a valuable tool for test track operations. It provided a comprehensive solution for verifying test compliance, ensuring that the performed tests align with the prescribed criteria. The graphical and intuitive interface facilitated ease of use, allowing test drivers to navigate the application effortlessly.

Through the utilization of GNSS positioning data and IMU sensor data, the application guided the test driver during test sessions. This guidance ensured accurate navigation on the test track, enabling the execution of tests according to the prescribed procedures. By leveraging real-time data, the application enhanced the efficiency and accuracy of the testing process.

Moreover, the application offered a numerical evaluation of test results. It allowed test drivers to assess the performance of the vehicle based on collected data, identifying areas where the vehicle did not meet the prescribed criteria. This evaluation process proved invaluable in highlighting potential areas for improvement and modifications, contributing to enhanced vehicle performance and safety.

7.1 Future Directions

The successful development of the application opens up various exciting possibilities for future directions and enhancements. Several avenues can be explored to further optimize its functionality and expand its capabilities.

One potential future direction involves integrating additional sensor technologies into the application. By incorporating lidar or radar sensors, the application's perception abilities can be enhanced. This integration would enable the application to detect and mitigate blind spots during testing, providing a more comprehensive assessment of the vehicle's performance and safety.

Another promising area for future development is the incorporation of machine learning algorithms. By leveraging machine learning techniques, the application can dynamically adapt to varying driving conditions and refine its analysis techniques. This would allow for more accurate and personalized insights, enabling the application to provide tailored recommendations and identify performance improvements specific to individual vehicles.

Additionally, the application can be further refined to support real-time monitoring and feedback during test sessions. This would enable test drivers to make immediate adjustments based on the application's analysis, ensuring optimal performance throughout the testing process.

Collaboration with industry partners and researchers could also be pursued to expand the application's capabilities and validate its effectiveness in real-world scenarios. By working closely with experts in the field, the application can be refined based on their valuable insights and feedback, leading to continuous improvements and advancements in automotive testing practices.

In all, the developed application has demonstrated its value and potential in the field of automotive testing. With future advancements such as integrating additional sensors, leveraging machine learning algorithms, and real-time monitoring, the application has the potential to revolutionize the testing process and contribute to the continuous improvement of vehicle dynamics, reliability, and safety. By embracing these future directions, the application can further solidify its position as a reliable and indispensable tool for enhancing vehicle performance and safety standards.

8 References

[1] *Everything You Need to Know about V2X Technology* (May 2021). [Online; accessed 14. Nov. 2022]. URL: https://www.autoweek.com/news/technology/a36190311/v2x-technology.

[2] Rohde & Schwarz GmbH & Co KG (Nov. 2022). *Scenario based testing of C-V2X applications in lab and field environments*. [Online; accessed 14. Nov. 2022]. URL: https://www.rohde-schwarz.com/us/applications/scenario-based-testing-of-c-v2x-applications-in-lab-and-field-environments-application-card_56279-1236097.html.

[3] Glaab, Markus, Alois Mauthofer, and Udi Naamani (2009). "New test and evaluation methods for future car2x communication based driver assistance". In: *21st Int. Technical Conference on the Enhanced Safety of Vehicles (ESV)*.

[4] Gelder, Erwin de and Jan-Pieter Paardekooper (2017). "Assessment of automated driving systems using real-life scenarios". In: *2017 ieee intelligent vehicles symposium (iv)*. IEEE, pp. 589–594.

[5] *List of Automotive Proving Grounds [Updated 2021] | Dewesoft* (Nov. 2022). [Online; accessed 14. Nov. 2022]. URL: https://dewesoft.com/daq/list-of-automotive-proving-grounds.

[6] ovyshnyk (Sept. 2022). "Human Machine Interface (HMI) Design for Autonomous Vehicles | Intellias Blog". In: *Intellias*. URL: https://intellias.com/what-s-really-important-about-designing-human-machine-interfaces-for-autonomous-vehicles.

[7] Alam, Mohammad Saquib (2020). *Automatic generation of critical driving scenarios*.

[8] Tamilarasan, Santhosh, Daniel Jung, and Levent Guvenc (2018). *Drive scenario generation based on metrics for evaluating an autonomous vehicle controller*. Tech. rep. SAE Technical Paper.

[9] Wen, Mingyun, Jisun Park, and Kyungeun Cho (2020). "A scenario generation pipeline for autonomous vehicle simulators". In: *Human-centric Computing and Information Sciences* 10.1, pp. 1–15.

[10] Menzel, Till, Gerrit Bagschik, and Markus Maurer (2018). "Scenarios for development, test and validation of automated vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1821–1827.

[11] Karunakaran, Dhanoop et al. (2022). "Automatic lane change scenario extraction and generation of scenarios in OpenX format from real-world data". In: *arXiv preprint arXiv:2203.07521*.

[12] Bengler, Klaus et al. (2020). "From HMI to HMIs: Towards an HMI framework for automated driving". In: *Information* 11.2, p. 61.

[13] Naujoks, Frederik et al. (2017). "Improving usefulness of automated driving by lowering primary task interference through HMI design". In: *Journal of Advanced Transportation* 2017.

[14] Fremont, Daniel J et al. (2020). "Formal scenario-based testing of autonomous vehicles: From simulation to the real world". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 1–8.

[15] Zhang, Zhengang, Xueyao Chen, Jieming Huang, et al. (2018). "Research on innovation posture of automated driving technology based on patentometrics". In: *Technology and Investment* 9.03, p. 137.

[16] Xinxin, Zhang, Li Fei, and Wu Xiangbin (2020). "CSG: Critical scenario generation from real traffic accidents". In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1330–1336.

[17] Elrofai, Hala, Daniël Worm, and Olaf Op den Camp (2016). "Scenario identification for validation of automated driving functions". In: *Advanced Microsystems for Automotive Applications 2016*. Springer, pp. 153–163.

[18] Ponn, Thomas et al. (2020). "Identification of challenging highway-scenarios for the safety validation of automated vehicles based on real driving data". In: *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, pp. 1–10.

[19] Miura, Keita and Takuya Azumi (2020). "Converting Driving Scenario Framework for Testing Self-Driving Systems". In: *2020 IEEE 18th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, pp. 56–63.

[20] Kalra, Nidhi and Susan M Paddock (2016). "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" In: *Transportation Research Part A: Policy and Practice* 94, pp. 182–193.

[21] Pütz, Andreas et al. (2017). "System validation of highly automated vehicles with a database of relevant traffic scenarios". In: *situations* 1, E5.

[22] Sippl, Christoph et al. (2019). "Scenario-based systems engineering: An approach towards automated driving function development". In: *2019 IEEE International Systems Conference (SysCon)*. IEEE, pp. 1–8.

[23] Rodarius, C et al. (2015). "Deliverable D7. 1: test and evaluation plan". In.

[24] *Setting the Standards for Scenario-Based ADAS/AD Testing, Design and Editing* (Nov. 2022). [Online; accessed 14. Nov. 2022]. URL: https://www.avl.com/-/setting-the-standards-for-scenario-based-adas-testing-design-and-editing.

[25] Elrofai, H, Jan-Pieter Paardekooper, et al. (2018). "StreetWise: scenario-based safety validation of connected automated driving". In.

[26] Bagschik, Gerrit, Till Menzel, and Markus Maurer (2018). "Ontology based scene creation for the development of automated vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1813–1820.

[27] Sauerbier, Jan et al. (2019). "Definition of scenarios for safety validation of automated driving functions". In: *ATZ worldwide* 121.1, pp. 42–45.

[28] Hasan, Monowar et al. (2020). "Securing vehicle-to-everything (V2X) communication platforms". In: *IEEE Transactions on Intelligent Vehicles* 5.4, pp. 693–713.

[29] Checkoway, Stephen et al. (2011). "Comprehensive experimental analyses of automotive attack surfaces". In: *20th USENIX security symposium (USENIX Security 11)*.

[30] Nast, Condé (July 2015). "Hackers Remotely Kill a Jeep on the Highway—With Me in It". In: *WIRED*. URL: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway.

[31] Humayed, Abdulmalik et al. (2017). "Cyber-physical systems security—A survey". In: *IEEE Internet of Things Journal* 4.6, pp. 1802–1831.

[32] Alnasser, Aljawharah, Hongjian Sun, and Jing Jiang (2019). "Cyber security challenges and solutions for V2X communications: A survey". In: *Computer Networks* 151, pp. 52–67.

[33] Shrestha, Rakesh et al. (2020). "Evolution of V2X communication and integration of blockchain for security enhancements". In: *Electronics* 9.9, p. 1338.

[34] Costandoiu, A and M Leba (2019). "Convergence of V2X communication systems and next generation networks". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 477. 1. IOP Publishing, p. 012052.

[35] Matuszak, Justyna (May 2022). "How to Select the Ideal HMI for your Application? - KnowHow". In: *KnowHow*. URL: https://knowhow.distrelec.com/manufacturing/how-to-select-the-ideal-hmi-for-your-application.

[36] Vaidya, Omkar S et al. (2020). "The Implementation of HMI based Smart Mirror using ARM Processor". In: *International Journal of Interdisciplinary Innovative Research & Development (IJIIRD)*.

[37] Prabha, D, MS Karthika, and P Manivanan (2019). "Cloud Based Health Monitoring and Abnormality Detection using Smart Mirror". In: *Indian Journal of Science and Technology:* 12.8, pp. 1–7.

[38] Forster, Yannick et al. (2017). "Driver compliance to take-over requests with different auditory outputs in conditional automation". In: *Accident Analysis & Prevention* 109, pp. 18–28.

[39] Kiesel, Andrea and Jeff Miller (2007). "Impact of contingency manipulations on accessory stimulus effects". In: *Perception & Psychophysics* 69.7, pp. 1117–1125.

[40] Daziano, Ricardo A, Mauricio Sarrias, and Benjamin Leard (2017). "Are consumers willing to pay to let cars drive for them? Analyzing response to autonomous vehicles". In: *Transportation Research Part C: Emerging Technologies* 78, pp. 150–164.

[41] Cruz-Benito, Juan, Francisco J Garcia-Penalvo, and Roberto Theron (2019). "Analyzing the software architectures supporting HCI/HMI processes through a systematic review of the literature". In: *Telematics and Informatics* 38, pp. 118–132.

[42] Kaur, Ekamjot (Apr. 2022). "A Female School Student Investigates the Python Programming Languag". In: *International Journal of Innovative Science and Research Technology* 7.4, pp. 152–155. DOI: 10.5281/zenodo.6476748. URL: https://doi.org/10.5281/zenodo.6476748.

[43] Kalita, Limi (2014). "Socket programming". In: *International Journal of Computer Science and Information Technologies* 5.3, pp. 4802–4807.

[44] Kurose, James F (2005). *Computer networking: A top-down approach featuring the internet, 3/E.* Pearson Education India.

[45] Olsson, Daniel and Eric Rylander (2022). "A Framework for Verification and Validation of Simulation Models in esmini". In.

[46] Tenbrock, Alexander et al. (2021). "The conscend dataset: Concrete scenarios from the highd dataset according to alks regulation unece r157 in openx". In: *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*. IEEE, pp. 174–181.

[47] Blanchette, Jasmin and Mark Summerfield (2006). *C++ GUI programming with Qt 4.* Prentice Hall Professional.

[48] Willman, Joshua and Joshua Willman (2021). "Overview of pyqt5". In: *Modern PyQt: Create GUI Applications for Project Management, Computer Vision, and Data Analysis*, pp. 1–42.

[49] Gibbs, Lisa et al. (2007). "What have sampling and data collection got to do with good qualitative research?" In: *Australian and New Zealand journal of public health* 31.6, pp. 540–544.

9 List of Figures

10 List of Abbreviations and Symbols

AV	Autonomous Vehicle
SUT	System Under Test
IMU	Inertial Measurement Unit
HRU	Human Road Users
CAD	Conditional Automated Driving
NDRT	National Disaster Response Team
LED	Light emitting diode
FTP	File Transfer Protocol
HTTPS	Hypertext Transfer Protocol
SSL	Secure Sockets Layer
JSON	JavaScript Object Notation
I/O	Input/Output
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
VUT	Vehicle Under Test
CSV	Comma-Separated Values
XML	Extensible Markup Language
SDK	Software Development Kit
V2X	Vehicle to Everything
GNSS	Global Navigation Satellite System
NDRT	National Disaster Response Team
NDRTs	Non-Driving-Related Tasks

11 Appendix

Listing 11.1: Scenario file

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSCENARIO>
   <FileHeader revMajor="1"
               revMinor="0"
               date="2020-10-09T10:00:00"
               description="Basic␣follow␣with␣interactive␣controller"
               author="esmini-team"/>
   <ParameterDeclarations>
      <ParameterDeclaration name="HostVehicle" parameterType="string" value="car_white"/>
      <ParameterDeclaration name="TargetVehicle" parameterType="string" value="car_red"/>
      <ParameterDeclaration name="EgoStartS" parameterType="double" value="50"/>
   </ParameterDeclarations>
   <CatalogLocations>
      <VehicleCatalog>
         <Directory path="../xosc/Catalogs/Vehicles"/>
      </VehicleCatalog>
      <ControllerCatalog>
         <Directory path="../xosc/Catalogs/Controllers" />
      </ControllerCatalog>
   </CatalogLocations>
   <RoadNetwork>
      <LogicFile filepath="../xodr/e6mini.xodr"/>
      <SceneGraphFile filepath="../models/e6mini.osgb"/>
   </RoadNetwork>
   <Entities>
      <ScenarioObject name="Ego">
        <CatalogReference catalogName="VehicleCatalog" entryName="$HostVehicle"/>
        <ObjectController>
            <CatalogReference catalogName="ControllerCatalog" entryName="interactiveDriver
                " />
        </ObjectController>
      </ScenarioObject>
      <ScenarioObject name="Target">
         <CatalogReference catalogName="VehicleCatalog" entryName="$TargetVehicle"/>
      </ScenarioObject>
   </Entities>
   <Storyboard>
      <Init>
```

11 Appendix

```
      <Actions>
         <Private entityRef="Ego">
            <PrivateAction>
               <TeleportAction>
                  <Position>
                     <LanePosition roadId="0" laneId="-3" offset="0" s="$EgoStartS"/>
                  </Position>
               </TeleportAction>
            </PrivateAction>
            <PrivateAction>
                 <ActivateControllerAction longitudinal="true" lateral="true" />
            </PrivateAction>
            <PrivateAction>
               <LongitudinalAction>
                  <SpeedAction>
                     <SpeedActionDynamics dynamicsShape="step" dynamicsDimension="time"
                          value="0.0"/>
                     <SpeedActionTarget>
                        <AbsoluteTargetSpeed value="50"/>
                     </SpeedActionTarget>
                  </SpeedAction>
               </LongitudinalAction>
            </PrivateAction>
         </Private>
         <Private entityRef="Target">
            <PrivateAction>
               <TeleportAction>
                  <Position>
                     <LanePosition roadId="0" laneId="-3" offset="0" s="50"/>
                  </Position>
               </TeleportAction>
            </PrivateAction>
         </Private>
      </Actions>
   </Init>
   <Story name="FollowStory">
      <Act name="FollowAct">
         <ManeuverGroup maximumExecutionCount="1" name="FollowSequence">
            <Actors selectTriggeringEntities="false">
               <EntityRef entityRef="Target"/>
            </Actors>
            <Maneuver name="IncreaseSpeedManeuver">
               <Event name="TargetSpeedIncreaseEvent" priority="overwrite">
                  <Action name="TargetSpeedIncreaseAction">
                     <PrivateAction>
                        <LongitudinalAction>
                           <SpeedAction>
                              <SpeedActionDynamics dynamicsShape="linear" value="1"
                                  dynamicsDimension="rate"/>
                              <SpeedActionTarget>
                                 <AbsoluteTargetSpeed value="30"/>
                              </SpeedActionTarget>
```

11 Appendix

```
                        </SpeedAction>
                     </LongitudinalAction>
                  </PrivateAction>
               </Action>
               <StartTrigger>
                  <ConditionGroup>
                     <Condition name="TargetSpeedIncreaseStartCondition"
                                delay="0"
                                conditionEdge="none">
                        <ByValueCondition>
                           <StoryboardElementStateCondition storyboardElementType="
                               act"
                                                            storyboardElementRef="
                                                                FollowAct"
                                                            state="startTransition"/
                                                                >
                        </ByValueCondition>
                     </Condition>
                  </ConditionGroup>
               </StartTrigger>
            </Event>
         </Maneuver>
      </ManeuverGroup>
      <StartTrigger>
         <ConditionGroup>
            <Condition name="FollowActStart" delay="0">
               <ByValueCondition>
                  <SimulationTimeCondition value="0" rule="greaterThan"/>
               </ByValueCondition>
            </Condition>
         </ConditionGroup>
      </StartTrigger>
   </Act>
  </Story>
 </Storyboard>
</OpenSCENARIO>
```