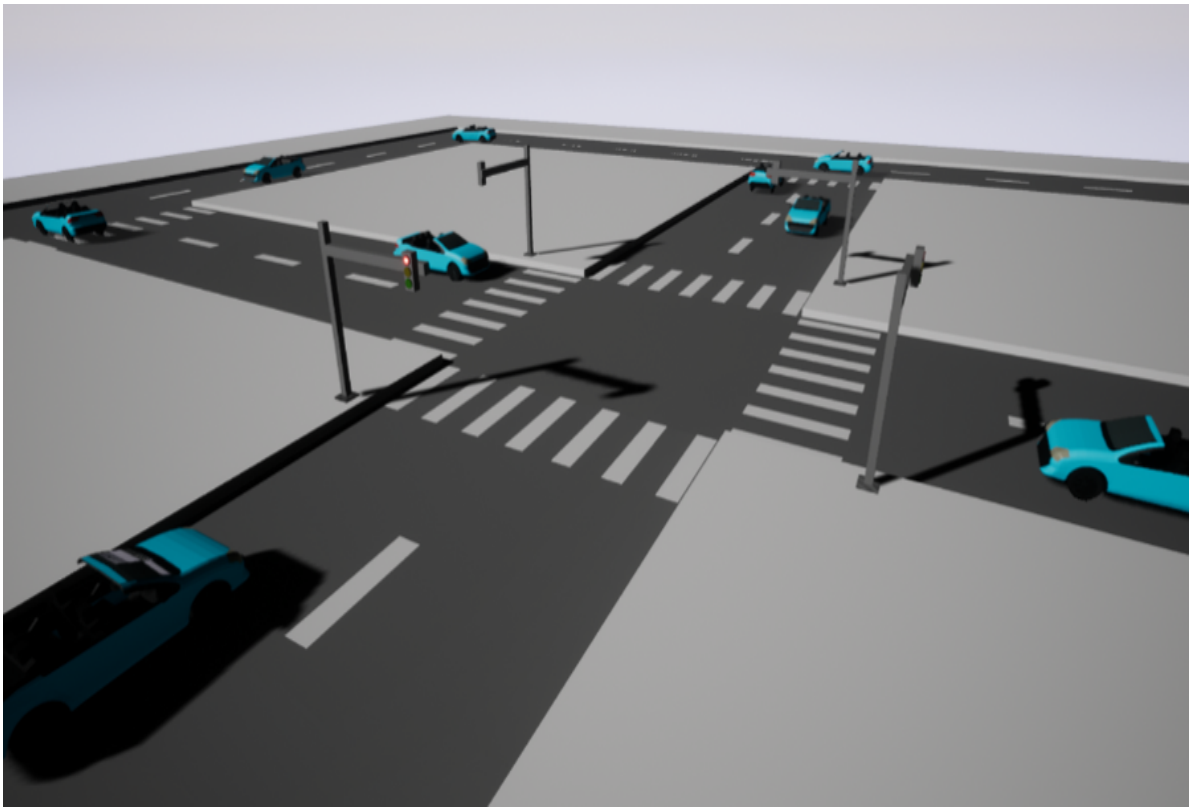

An Embedded Real-Time Traffic Simulation Environment on Vehicle-In-Loop Framework for Autonomous Driving

Master's thesis in Embedded Electronic System Design

VARSHA CHANDRASHEKAR
XINYUAN WANG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

An Embedded Real-Time Traffic Simulation Environment on Vehicle-In-Loop Framework for Autonomous Driving

VARSHA CHANDRASHEKAR
XINYUAN WANG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

An Embedded Real-Time Traffic Simulation Environment on Vehicle-In-Loop Framework for Autonomous Driving
Varsha Chandrashekar
Xinyuan Wang

Supervisor: Roger Johansson, Department of Computer Science and Engineering
Advisor: Siddhant Gupta, Volvo Car Corporation
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

An Embedded Real-Time Traffic Simulation Environment on Vehicle-In-Loop Framework for Autonomous Driving
Varsha Chandrashekar
Xinyuan Wang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Autonomous driving(AD) and Advanced Driver Assistance (ADAS) functions are rapidly evolving. Extensive testing of these systems is necessary at every stage of development. To ensure high performance and safety of ADAS and AD, the development of test technology has to happen at least at the same pace to support development. In order to cater to a more realistic testing, Vehicle-in-loop (VIL) methodology integrates a real vehicle into a virtual traffic environment to validate the vehicle's behaviour. This is both safe and resource efficient. Since it involves both physical/real actors and virtual actors from the simulator, for a timely execution of a test scenario we need a closed loop simulation, with each actor on the test track and simulator being aware of one-another. Closed loop simulation requires updating of trajectories dynamically to each actor based on their dynamic positions to be able to execute a defined traffic scenario. There needs to be synchronized flow of information to and from the actors in real time. In this thesis, we worked on the development of a server on an embedded real time hardware platform. The server is a centralized controller steering test targets along given real time updated trajectories. This server is capable of controlling all the test targets by communicating with them via User Datagram Protocol (UDP) over Wi-fi, in a synchronized manner to provide for a closed loop simulation in real time as a part of VIL. We also studied an existing traffic simulator developed in Simulink and analyzed the different sources of latencies in the system. These latencies lead to scenarios failing to execute in real time due to synchronization errors. As an approach to solving these timing issues, we deployed this simulator onto the same real time platform as the server. The test results showed that the server could steer a Radio Controlled (RC) car in synchronization with the virtual actor generated by the simulator both in open loop and closed loop. Synchronization was achieved by making all the actors refer to a global time stamp and by reducing latencies in the simulator to meet all real time execution constraints.

Keywords: VIL, Closed-loop, Server, Simulator, Real-Time systems, Hardware Platform, UDP.

Acknowledgements

We would like to thank our supervisor at Chalmers, Roger Johansson for his encouragement, constant support and guidance through entire course of the thesis.

We also extend our gratitude towards Siddhant Gupta and Francesco Costagliola for being great mentors at Volvo Car Corporation and giving us the opportunity to work on this thesis. They kept us motivated and provided us with all the necessary tools and valuable inputs. Their knowledge and insights about the work has been of great help to us.

Our special thanks to Goksan Isil, Junhua Chang and Angel Molina Acosta for their support and help with all the technical aspects.

Additionally, we would like to thank our examiner, Per Larsson Edefors for being a great advisor and providing us with constructive feedback throughout.

Varsha Chandrashekar & Xinyuan Wang, Gothenburg, January 2020

Contents

List of Figures

1

Introduction

Advancements in the field of Autonomous Driving (AD) are taking place at a rapid pace. Testing of these new developments is crucial in order to ensure high safety and performance. Therefore, testing technology is also expected to progress at the same pace to keep up with the developments. Using simulation as a part of verification plays a vital role as it allows to create every possible scenario for testing which may not otherwise be feasible or affordable to test physically. It also helps cover the increasing amount of required tests for every step towards completely autonomous vehicles. Integration of simulations with test track technology would help support complex tests for autonomous driving systems.

With the dynamic flow of data, it is necessary to have a continuous feedback, so as to take actions on real-time basis, and hence we require a closed-loop system. Closed-loop systems help simulate the dynamic behavior of the Vehicle Under Test (VUT) (changing of speed or trajectory by the AD system) and to adapt the behavior of the surrounding road users to the VUT. Connecting the closed-loop simulation environment to the real actors on test track, allows overcoming the limitations of the fixed (non-adaptable) trajectories[5]. Meanwhile, it requires synchronization in real-time between the simulator and controlling of robots or monitoring of physical objects[6].

To correctly test and validate the actions of autonomous driving on a simulator, it requires that the simulator also operates in real-time. Simulations should be able to represent the environment for the system while being fast enough to meet the real-time constraints. Simulators at Volvo Cars currently executed on general operating systems like Windows and Linux show large latencies[7] and hence the representation of actual real-time complicated scenarios might not be accurate. Therefore, there is a growing need to run realistic validation of time critical real-time systems on real-time platforms[8].

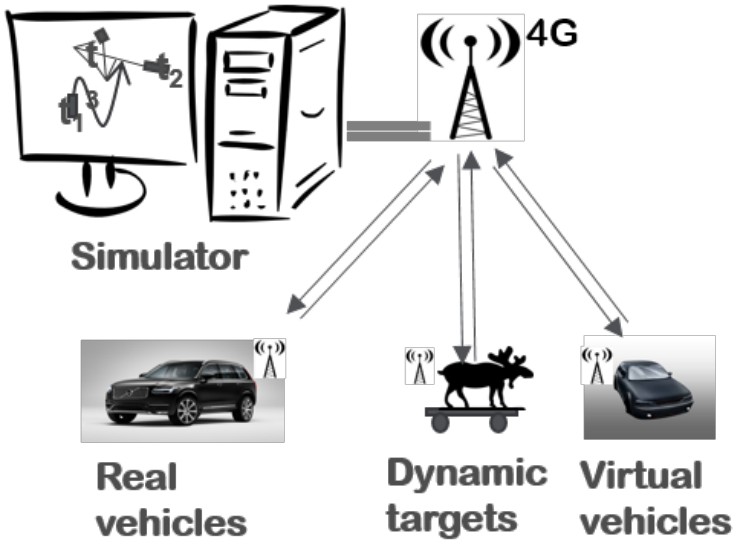

Figure 1.1: A closed-loop simulation environment including both virtual vehicles and real actors on test track.

1.1 Scope of the thesis

Increase in the complexity of AD and Advanced driver-assistance systems (ADAS) is placing increasing demands in testing of these functions. In addition to real-world testing on roads and test tracks, engineers rely strongly on virtual test driving with the help of simulators for development of such advanced systems. Some of the firmly established test methods are model-in-the-loop (MIL), software-in-the-loop (SIL) and hardware-in-the-loop (HIL) test systems which help validate a developed solution with simulations of real situations[9]. A MIL, SIL or HIL simulation describes the way in which a system is tested. In the instance of MIL, control algorithm and the mathematical model of the system are implemented as a simulation model which is then subjected to test. The model can consist of a physical or mechatronic system or model of the intended software[9]. SIL tests refer to testing a software, this software is optimized code for the embedded target but run it in a computer environment[9]. In a HIL simulation, a hardware like the ECU is involved. Any software deployed onto the hardware is compiled and built exactly like it's done for the intended application in the actual vehicle[9].

The latest addition in the validation chain is vehicle-in-the-loop (VIL) testing. This method bridges the gap between HIL and real world test driving by embedding a real test vehicle into a virtual traffic environment[10]. In this environment, in order to execute a traffic scenario, the driver or physical robot maneuvers the car on the test track and other objects relevant to the test scenario such as other vehicles, pedestrians, cyclists, buildings or traffic signs are generated in the virtual world[10]. Conventional real world tests, for example in the case of testing emergency brake assist, use balloon targets as crash opponents to investigate the functionality. Whereas, in VIL, the crash opponents are generated virtually and displayed to the driver via

augmented reality glasses or on the simulation monitor. The virtual sensors detect these virtual traffic objects and transmit the information to the vehicle's ECU, which processes this data, executes the driver assist algorithms and actuates necessary action as per the situation. Therefore, it provides for a risk free, safe vehicle testing and resource efficiency[10].

Our thesis is a subset of the VIL environment. Here, we consider a mixed traffic scenario with physical actors(e.g. Radio control (RC) cars, driving robots, AD vehicles), virtual actors and VUT. We implement a simulator that generates virtual actors and traffic scenario to be executed in real-time and develop a server that controls all the test targets on track, brings about coordination and synchronized information flow to the actors to ensure timely, safe and realistic testing. The server communicates with the physical objects on track over Wi-fi, by transmitting and receiving packets via User Datagram Protocol (UDP). The virtual actors generated by the simulator are based on a Volvo vehicle model which has seven degrees of freedom, longitudinal, lateral, yaw for the chassis and the rotational axis of the four wheels, i.e. no pitch, roll or suspension movements.

1.2 Steer-By-Server Architecture

Steer-By-Server (SBS) is a traffic simulator built in Simulink running on Windows platform developed in-house by Volvo Car Corporation for test monitoring and control of both virtual and real objects. SBS aims to run traffic scenarios involving multiple actors such as physical actors(e.g. Radio control (RC) cars, driving robots, AD vehicles) and virtual actors connected to the server and control them on the test track to execute a defined traffic scenario.

The simulator is currently composed of State Server, Traffic Manager, Trajectory Planner and virtual actors generated by the virtual actors engine. The State Server takes the states and positions from different actors and fuses them into a common position bus for the whole test setup. Traffic Manager defines the scenario events that the actors should follow and overviews the test to make sure that everything works fine and can spot possible collisions. The Trajectory Planner module takes desired scenario events as the input and transforms them into a continuous and global trajectory which is then sent back to the actors as shown in figure 1.2.

1.2.1 State Server

The dynamic inputs to the server are the states of all the actors involved in the test. The state includes the positions, velocity at each direction, yaw and other additional data about the actor (for example ID and type of actor). The state is sent using a bus structure called actor bus which includes two sub buses, the pose bus for position and data bus for other information about the actor. In this way, individual vision from different actors could be obtained and collectively used for subsequent purpose of desired scenario generation in traffic management module[11].

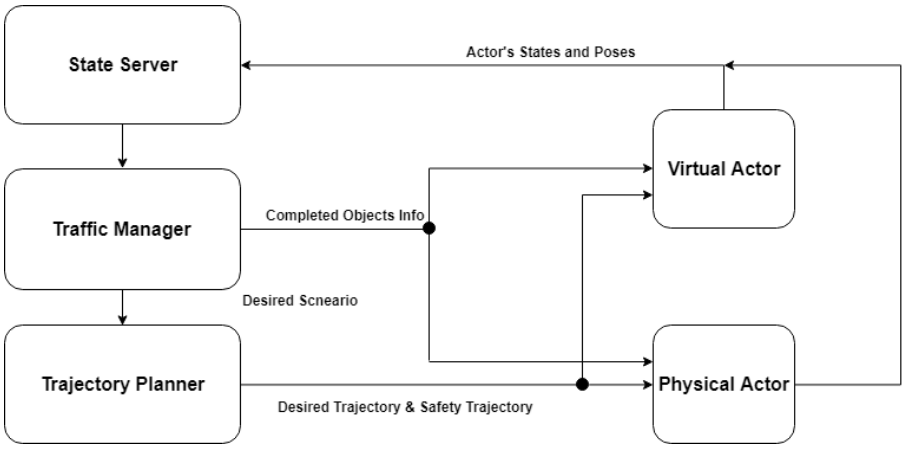

Figure 1.2: The architecture of SBS with both virtual and physical actors.

1.2.2 Traffic Manager

Traffic Manager accepts the actor bus from State Server as input, defines the scenario events that the actors should follow and oversees the test to make sure that everything works fine[12]. The initial static scenario including the parameters of object properties and corresponding road information is defined in an excel sheet, loaded to the workspace in Simulink after initialization. The scenario generator blocks are designed for each actor variant, i.e different vehicle models, new variants can be added when new kinds of vehicle models or scenarios are to be implemented.

1.2.3 Trajectory Planner

Trajectory Planner takes the desired scenario events from the Traffic Manager and transforms them into a continuous and global trajectory. As the server keeps receiving message from actors, desired trajectory of different actors gets updated at every time stamp, which is then sent back to the actors. Trajectory Planner also contains a Central Collision Avoidance System (CCAS) that can detect possible collisions based on the estimated future states of the actors and can generate safe trajectories for actors to avoid the collision[13], which meanwhile would be sent back so as to let the actor decide and select which trajectory to follow.

1.2.4 Actors

Actors in the SBS can contain both virtual actors and real actors on the test track. Actors' specific controllers, models and communication interfaces lie separately in each actor module.

1.3 Thesis Objectives

This thesis work aims to develop a simulator and a server to run on a real-time platform. It can help to simulate the dynamic interaction of traffic objects such as

virtual actors, physical vehicles and pedestrians on the test track with the VUT. At the same time it will allow a more efficient testing of the VUT's functions, resulting in higher efficiency and lowering of risk. Synchronization problems with traffic objects shall be eliminated which is one of the main goals of this work.

In particular the thesis work will focus on the following engineering tasks:

- Develop a light-weight traffic simulator on a real-time platform that can simulate a pre-defined traffic scenario with 2 traffic objects, including two cases: two virtual objects as well as one virtual object and one physical object.
- Migrate an existing closed-loop system for dynamic behavior of VUT on the developed real-time simulation platform.
- Achieve time and data synchronization between the virtual actor in simulator and physical actor on a test track. Both of two actors can accelerate, brake and steer according to each other's action.

Some of the research challenges that have to be addressed are:

- Investigate and optimize the architecture of simulator to handle fast-speed and high-volume data processing tasks to generate the desired trajectories for objects.
- Find the cause of latency in the existing system and then select an appropriate hardware platform to run the simulator so as to meet specific real-time constraints.
- Optimize and quantify the real-time performance of developed traffic simulator with respect to processing time and communication latencies in the system.

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 presents the selection of real-time hardware platform and the implementation of SBS on the hardware platform. Chapter 3 explains how to achieve communication and synchronization between simulator and actors as well as between actors in detail. Chapter 4 gives the simulation results obtained. The conclusions followed by discussion about the future work are reported respectively in Chapter 5 and Chapter 6.

2
Simulator on Hardware Platform

Steer-By-Server (SBS) requires the use of real-time testing for a realistic performance evaluation. This can be put into effect with the Vehicle Under Test (VUT), equipped with Electronic Control Unit (ECU) on which the Active safety and Autonomous driving (AD) algorithms execute, on the test track along with the physical and virtual actors to realize a traffic scenario more efficiently[14]. Therefore, one of the objectives of this thesis work is to implement SBS on an embedded platform. SBS on the platform with a server that can control, communicate with and synchronize the physical actors with virtual actors on the test track in real time. It can also be able to adapt the trajectories of one actor in simulation with the dynamic movement of other actors in order to execute a desired traffic scenario.

2.1 Real-Time Constraints

According to the system specification, sample frequency of the data received from the physical test actors on track is 10 Hz. This is the highest frequency of data reception in the system, which implies that the execution of all the modules in the simulator should be completed before the arrival of the subsequent sample. The data being sent to and received from the physical actors on track will suffer from network delay and the reaction time caused by the physical actor, whereas, since the virtual actors are generated and controlled in the local machine there is no delay in the reception/sending of its data, resulting in positioning lag. Therefore, considering these communication properties, it requires a synchronization method in real-time between a simulation and control/monitoring of physical actors for the possibility to synchronize movement of physical actors with virtual actors.

2.2 Selection of Hardware Platform

In order to fulfill the requirements mentioned above and to solve the issues of latencies, it would first be necessary to reduce the execution time of the simulator, for which, it is necessary to determine the execution time of each task in the simulator and the potential to improve. First approach to do so would be to optimize the technology platform and run it on a dedicated processor[8]. When it comes to the selection of an appropriate hardware platform for deploying the SBS onto it to meet the real-time constraints, the Graphic Processing Unit (GPU), Field Programmable Gate Array (FPGA) and standalone microprocessor were our initial considerations. GPU is well known for its massively parallel architecture and high data-parallel

execution performance[15], while the FPGA owns the advantages of reconfigurability, computing power and high development efficiency. Nevertheless, for our system there is not large enough data processing parallelism in the applications that could exploit the computing power of the GPU, meanwhile the lack of operating system as well as VHDL/Verilog development environment on traditional FPGA also makes it time-consuming and hard to test the system in real-time[16].

Based on the above analysis and limitations, we turn our concentration to a standalone microprocessor. According to the specifications in hand, the first hardware platform for the prototype we focus on is the Raspberry Pi 3B+[1]. It is equipped with 32-bit ARM Cortex processor based on Broadcom BCM2837B0 System on Chip (SoC) with ARMv7-A architecture, also with four cores running on 1.4 GHz frequency. A large memory, 1GB LPDDR2 SDRAM, also contributes to executing complex system efficiently. Its pre-installed operating system is named as Raspbian, a 32-bit operating system based on Debian and optimized for the Raspberry Pi hardware. Besides, for an initial prototype, the cost of Raspberry Pi is low and reasonable. However, in seeking to make a comparison between the performance of different hardware platforms, then we selected the second platform with more powerful hardware support, the NVIDIA Jetson TX2[2]. It's an embedded supercomputer equipped with 64-bit ARM Quad-Core A57 processor, one NVIDIA Pascal architecture GPU and 8GB L128-bit DDR4 memory. Moreover, 64-bit UBUNTU 16.04 operating system run on it. Our developed simulator will be implemented on both of the selected platforms and their performance test results will be shown and discussed in Chapter 4.

Figure 2.1: The Raspberry Pi 3B+ module[1].

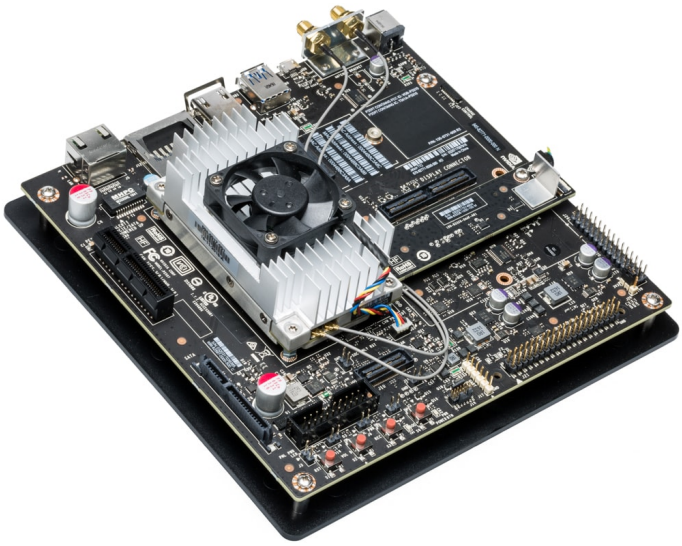

Figure 2.2: The NVIDIA Jetson TX2 Development Platform[2].

2.3 System Implementation on Hardware Platform

Before running the SBS on Raspberry Pi, a modification needs to be done in advance. One of the ideas behind the architecture of SBS on the hardware platform, which is shown in figure 2.3, is to have a modular approach, where the actors and scenario generators have a number of different variables that are changeable. This means that the design is open to any type of actor and scenario that can be defined with the existing interface. In this thesis, in order to achieve synchronization between the virtual and physical actors for a mixed traffic scenario, the model of Radio Control (RC) Car, which represents the physical actor is included as a new type of actor into the simulation environment. It could then be treated the same way as the virtual actors by the server algorithms, but with real vehicle dynamics unlike in the case of virtual actors where the dynamics are simulated using a vehicle model. The interaction between the physical actor and the SBS will be discussed more in details in Chapter 3.

2.3.1 Code generation

Since SBS is currently built based on the Simulink models, in order to deploy SBS on the selected hardware platform, we need to convert it to Embedded C code. After initializing the SBS in Simulink and loading the intersection scenario with two actors included, we utilize the Embedded Coder, an installed application in MATLAB, to generate the equivalent C code for SBS, which includes the necessary source and header files for the whole system.

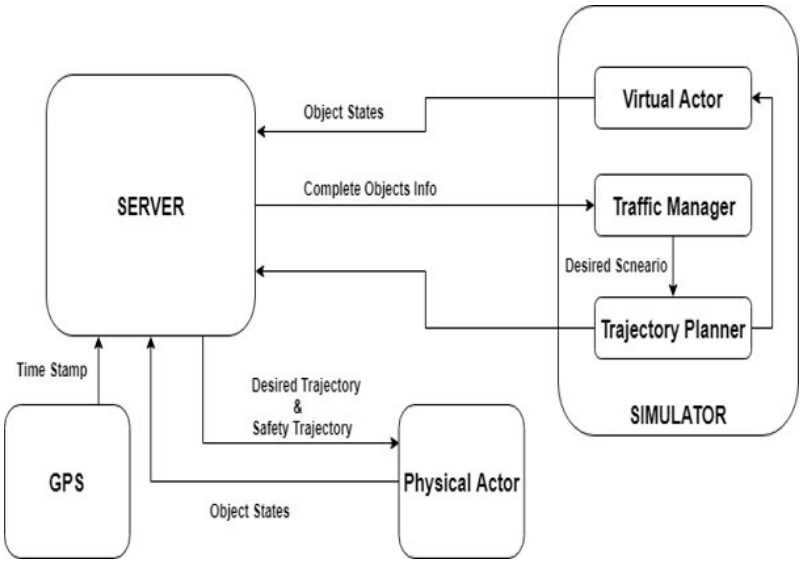

Figure 2.3: The architecture of SBS on the selected real-time platform.

2.3.2 SBS Scheduling

When it comes to the scheduling of SBS so as to be executed more efficiently, there are two main methods using multi-threading that have been taken into consideration [8].

As shown in figure 2.3, the inputs to the three main functional blocks of SBS (State Server, Traffic Manager and Trajectory Planner) depend on former block's output and hence require sequential execution one after the other. Therefore, we build a monolithic architecture for these three blocks and schedule them as one thread run on one single core with a periodic call. We then explore the potential to assign sub-blocks of SBS to independent threads and let them run in parallel. A thread is the smallest sequence of programmed instructions that can be managed independently by a scheduler. In order to obtain a faster overall execution time to meet the real-time requirement, multi-threading is considered to be used on the actor modules in SBS. Each actor module will be arranged as one new thread since they are independent of each other and other three functional modules. With each core of the processor executing a separate thread and multiple threads would be executed in parallel, better utilization of computing resources of the quad-core processor of Raspberry-Pi could therefore be fulfilled through the thread-level parallelism[17]. However, since multiple threads share the same address space and can therefore interfere with each other, we implement mutual exclusion using mutex flags in scheduled code. A mutex flag can protect common data in caches from being overwritten, for example, by the thread of virtual actor while being modified and stored by the thread of three functional blocks.

The other method considered is to parallelize specific functions inside a certain block. In this case, an Application Programming Interface (API) named as OpenMP[18], which supports multi-platform shared memory multiprocessing programming in C,

is introduced into the generated code. OpenMP is also an implementation of multi-threading, in which certain number of threads will be forked from a master thread and concurrently run the parallelized section of code[8]. In our generated code, we found that a few functions are looped for a specific times to generate desired output for each actor, which is completely independent at each time and could be scheduled by calling OpenMP to run in parallel. In the Traffic Manager module, for example, OpenMP is called before one for-loop, in which a specific function is called at each time to find the current starting coordinate of road for each actor at certain time stamp. Because in the intersection traffic scenario, each actor run on an independent road with its own distinct desired scenario, in this way, we could replace the for-loop with concurrently running threads and thus reduce the time complexity of function's algorithm, which leads to minimizing its execution time.

2.3.3 Cross-compilation

Considering the fact that Raspberry Pi and Jetson TX2 are both powered by ARM Cortex A processor, but the generated C code is to fit the Windows x86 operating system, means that a cross-compiler is compulsorily required if we need to execute the code on the ARM based platform. In our case, for remote debugging we use Eclipse[19] as the Integrated development environment and select two types of cross-compiling toolchain named respectively as *arm-linux-gnueabihf* [20] for Raspberry Pi's 32-bit ARM architecture processor and *arm-linux-aarch64* [20] for Jetson TX2's 64-bit AArch64 architecture processor. The generated C code can then be cross-compiled and linked to an executable file that could run on two selected platforms. The executable file could be transferred onto Raspberry Pi through the Secure Shell (SSH) connection between the target hardware and Eclipse.

3

Communication and Synchronization

In this chapter, we discuss the implementation of communication between the simulator on selected hardware platform and physical actor in order to update the states and trajectory dynamically. The behavior synchronization mechanism between physical actor and virtual actor is also discussed in the following section.

3.1 Communication

User Datagram Protocol (UDP) is selected to take the role of communication protocol for data transfer between the server on real-time platform and the physical actor on test track , which is Radio Control (RC) Car in this case, to enable a fast packet transmission in real-time. The corresponding UDP communication interfaces have been implemented on the sever and the actor.

3.1.1 UDP

UDP communication is used to exchange information over WiFi about states (position and velocity etc.), trajectory and time stamp periodically between the state server and real actors, which is RC Car in this case. The reason why we select UDP rather than Transmission Control Protocol (TCP) is that TCP/IP is an acknowledgment based protocol, lost data packets will be resent and received by receiver in incorrect order, which will not only cause congestion in communication network but also introduces unexpected latency. In contrast, what UDP offers is unreliable but fast data transmission[21]. The UDP transmitter attempts to send data every 100ms and the UDP receiver is continuously waiting until new data packet is available and accepted, while the data packet from UDP transmitter unsuccessful to be received will be let dropped off since it would be out-of-time and invalid in real-time scenario.

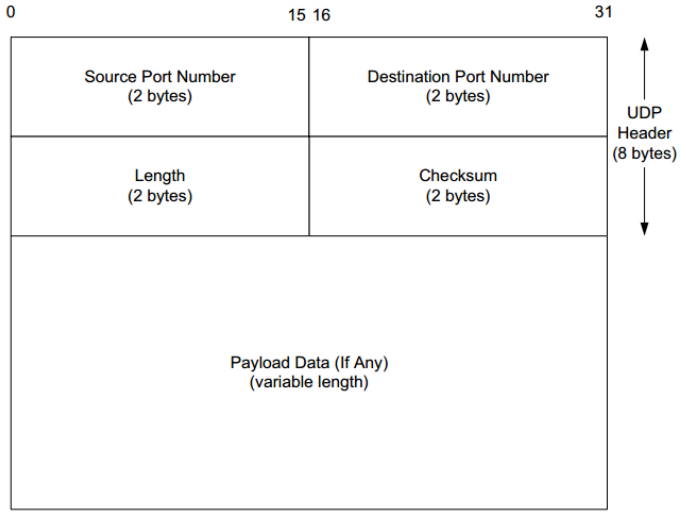

Figure 3.1: The format of UDP datagram, including the UDP header and payload[3].

Distinct network IP address is assigned to the UDP server running on the selected hardware platform and UDP client on physical actor. IP address of server on the platform is set to be static so that we do not need to update the UDP code every time the network changes. The server will keep listening for the address bound socket and queue the received UDP packets if multiple clients are sending messages at the same time. The size of buffer for sending and receiving packets at bound socket should be updated with the number of clients and the average size of packet. Format of UDP packets transmitted between the UDP server and client is shown in figure 3.2. Once the UDP server receives the message from client, contents in this message are parsed, mapped to actor bus and concatenated with other state information updated by virtual actors in the server at the same time.

3.1.2 RC Car

The communication between the server and RC Car is not direct but through a RC control station, which is developed based on QT[22] in C++ and run on a laptop. The communication protocol between the server and the control station is still UDP but the format of transmitted UDP data has been pre-defined. Currently, the message format is based on the structure of Extensible Markup Language (XML), which is shown in figure 3.2. The message is composed of different types of commands such as replace route, add route point and get state; the ID of the RC Car and certain number of way points including the coordinates, speed and time stamp. Both the XML parser and deparser have been implemented on the server. When the RC control station receives the UDP packet from the server, it decodes the packet and then parse the contained XML message. The processed message is sent to the RC control station which then forwards it to the microcontroller on RC Car through a radio wave with 115200 Baud rate,the RC car then behaves as per the received commands and follow the desired trajectory.

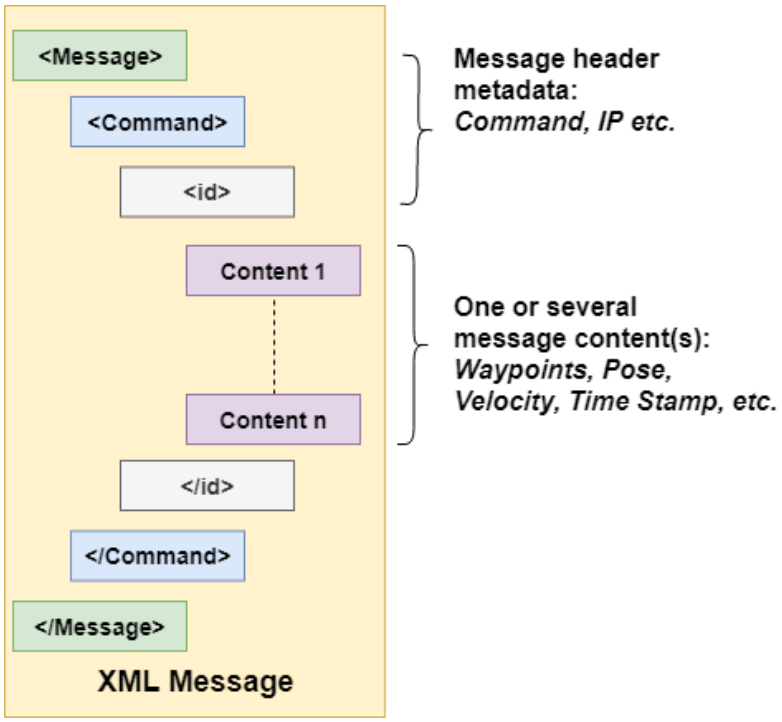

Figure 3.2: Structure of the XML message transmitted between UDP server and client.

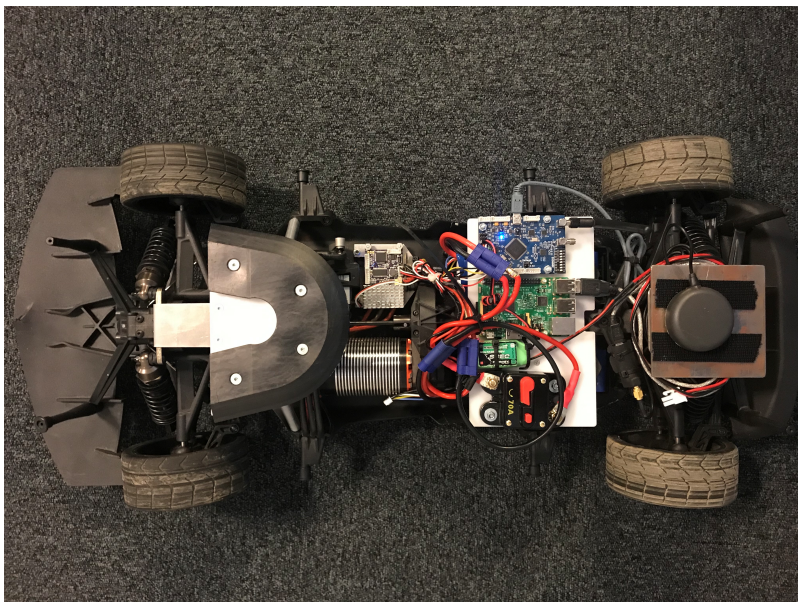

Figure 3.3: The RC Car used as a physical actor in the project.

3.2 Synchronization

Synchronization in the system should be achieved with regards to the control and movement of physical actors and virtual actors that are connected to the state

server over WiFi. The position updates from the physical actors are sent to the sever periodically and the server sends back the updated trajectories to all the actors. The position update for the virtual actors is done on the local machine and therefore will not suffer from network delay, whereas, in case of data transfer to and from the physical actors to the server, there will be receive and transmission network delays which makes the synchronization between all the actors hard to achieve due to the network delay being random.

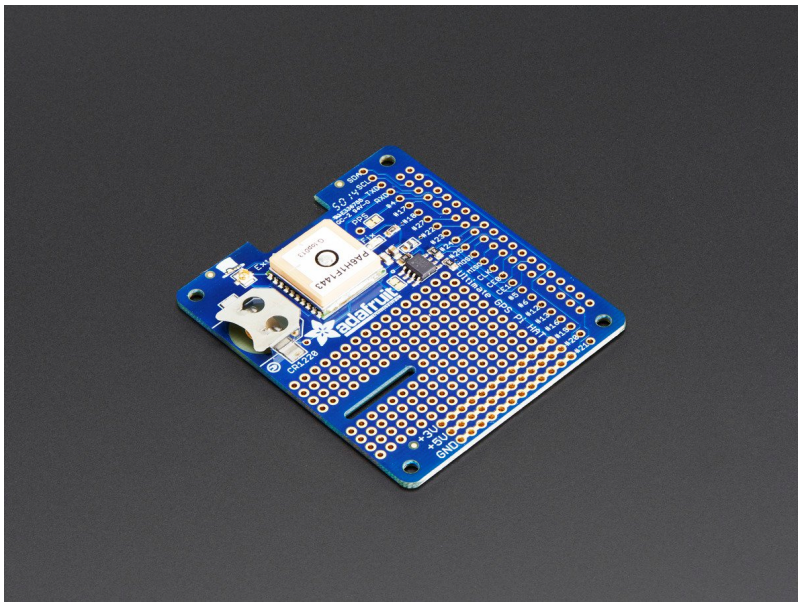

Figure 3.4: Adafruit Ultimate GPS HAT mounted on the selected platforms[4].

3.2.1 Time Synchronization

Time synchronization is required to be able to synchronize trajectories and measurements in the distributed actors-mixed traffic test system. We aim to achieve this by time stamping each of the messages sent with reference to a global time stamp[6], indicating the corresponding position information from the GPS module at that particular instance of time. Each physical actor connected to a local GPS will receive a very high accuracy time stamp from its own GPS time module. The server and the virtual actors do not make use of a GPS module, therefore, a very high accuracy clock stamp is obtained by using an external time module, which is named as Adafruit Ultimate GPS HAT[4] and shown in figure 3.4. It has a built in Real Time Clock (RTC), -165 dBm sensitivity, 10 Hz updates and 66 channels. When the server receives the message from the physical actor, the actor's time stamp contained inside the message will be extracted and compared with the time stamp from the server. Due to the network latency occurring in the delivery of message, a timing offset need to be taken into consideration on the server side and added to its time stamp when it needs to send the message back to physical actor. Thus, after each 100 messages being sent and received, the mean of message delivery time will be calculated, updated and used as the offset to synchronize the time between the server and physical actors.

3.2.2 Data Synchronization

There are mixed type of traffic actors in the distributed system, it's therefore important to know whether all actors have received the data at the right time step. Here, there exist two different directions of data flow which have to be taken into account and processed differently. Data received from the actors is treated non-critical and the test can still continue even if this data is not received for a few time steps. Therefore, having a timeout on the receiving data is sufficient and the test can be shut down if there is no data that is received from one of the actors for some stipulated amount of time. When it comes to the data synchronization between the virtual actors and physical actors, a buffer will be utilized to hold the updated state of the virtual actor for a certain time until the latest state of physical actor being received. Then, both of them will be concatenated to an actor bus and sent to the state server.

There are two main messages going from the server to the actors. One is safe trajectory to be taken in case of a collision detection and the other is the desired trajectory to execute the test traffic scenario. It gets more difficult when the information is sent from the server because the information is more critical and the synchronization task has to be handled by both, the actor and the server. Hence we require an acknowledgement from the actors to the server. If the server doesn't receive any acknowledgement, it will then try and resend the data and if a few tries fail, shut down the test. If the actors don't receive any data from the server for some interval of time, they will have to execute their shutdown process. A system for acknowledgement or time out should be implemented in the server. The actors have to shut down if there is loss of contact with the server.

4

Testing and Results

In this chapter, based on the developed simulator and server as well as the hardware platform, a set of function validation and performance tests have been carried out. The two different testing methods and the corresponding test results will be demonstrated in the following sections.

4.1 Testing Methods

We start with the open-loop simulation, which consists of two virtual actors generated by the simulator running on the local machine, on the selected real time platform, in order to test whether the consumed execution time can meet the preset simulation deadline or not. Next, a more complex closed-loop simulation, with one physical actor introduced to replace a virtual actor in the simulation, has been carried out to test if the execution time of the system can meet the real-time constraints at each sampling step and if synchronization between two actors can be achieved in the close-loop simulation. Both of the simulation tests utilize the same defined scenario, in this case an intersection scenario is considered with simulation time of 18s to be fulfilled.

The intersection traffic scenario defined is as shown in figure 4.1 below, there are two actors with a given initial velocity of 30 Km/h. Actor 1 will set off from the coordinate (102, 0) while actor 2 will set off from the coordinate (9, 98) in the map. The trajectory is generated by the trajectory planner depending on the scenario which is defined by the traffic manager, such that the actors meet each other at the defined intersection area which considers the width of the vehicle at an expected time stamp.

4.1.1 Open-Loop Simulation Testing

In open-loop simulation testing, the actors are not aware of each other's positions and there is no dynamic adaptation of trajectories. The simulator defines a scenario, trajectories for the actors are generated every 100ms. Firstly, the test was carried out with two virtual actors that are generated by the simulator, this test was to investigate the execution times of the processes without network delay as virtual actors run on the local machine. We execute the system on both of the 32-bit and 64-bit architecture hardware platforms mentioned earlier.

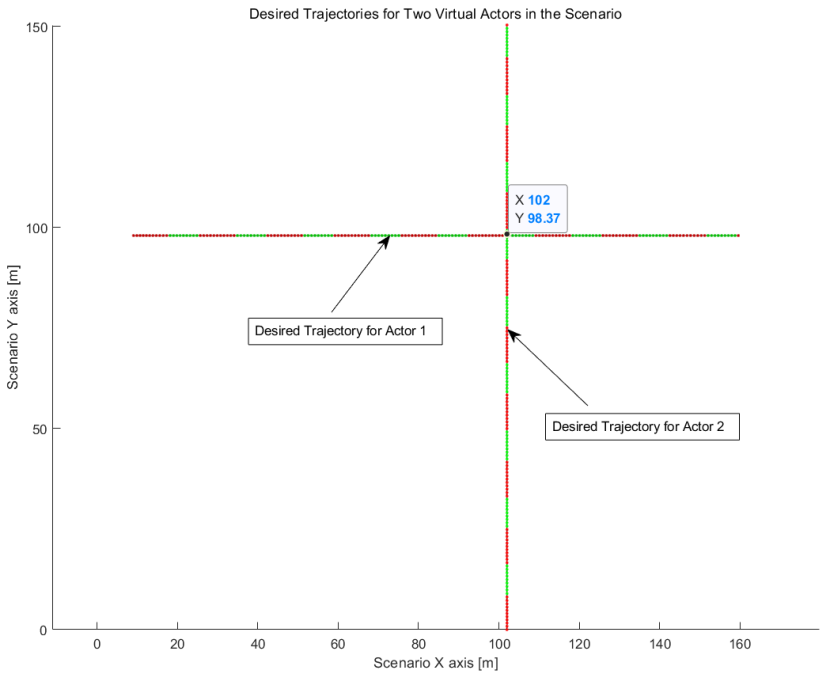

Figure 4.1: Illustration of intersection scenario defined for the simulation test.

4.1.2 Closed-Loop Simulation Testing

In closed-loop simulation, all the actors are aware of each other's positions. One actor adapts itself with other actors' changing dynamically trajectories. Adaptation is based on the position and velocity of the actors. We first perform this test with two virtual actors as mentioned in the previous test method to benchmark the execution time. We then perform the same procedure with one virtual actor and one physical actor, the Radio Control (RC) car, since the physical actors are connected to the server over the WiFi in Volvo Car Corporation, network delay is introduced in this case. This test also helps us identify if the synchronization between the actors is achieved as the virtual actors run on the local machine with no network involvement, the server is expected to send data to both physical and virtual actors only when the processing of data from the physical actors is complete, which maybe delayed due to communication over the network. This system is also executed on both of the 32-bit and 64-bit architecture platforms.

4.2 Test Results

In this section, we demonstrate the results from two types of simulation tests. Desired trajectories, dynamic velocity and positions for both of actors during simulation are plotted according to the generated simulation log files. The related real-time clock data is also added into the figures to present the realistic simulated process at each time stamp.

4.2.1 Open-Loop Simulation Test Results

The open-loop simulation results are shown in the figures below from the tests performed by following the set up mentioned in section 4.1.1.

Figures 4.2 and 4.4 illustrate the desired trajectories for two virtual actors generated by the simulators running on Raspberry Pi and NVIDIA TX2 respectively, figures 4.3 and 4.5 show the corresponding positions of the two actors at different time stamps following the desired trajectories. The positions of the actors in case of open loop refer to a local co-ordinate system defined in the scenario. The time stamp for each of the actors positions and trajectory points is obtained from the internal system clock which can be seen in the plots as the Z axis. This provides for real time sampling and execution of different tasks.The significance of these time stamps in the plots is that, they indicate the time of the coordinates of actors at instances of intersection and end of scenario which represent if the real time execution time constraints were fulfilled.

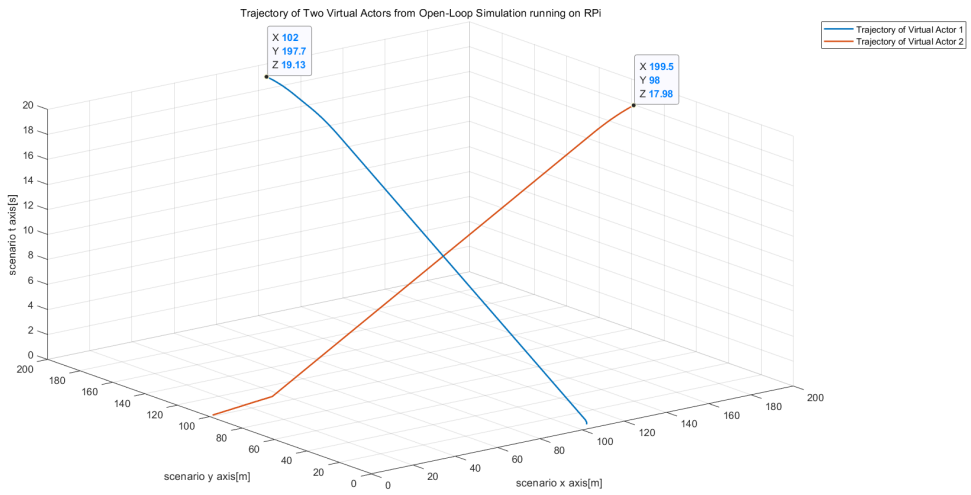

Figure 4.2: Illustration of two trajectories with related time information, which is plotted as Z axis, generated from the open-loop simulation running on Raspberry Pi. Two virtual actors receive and follow their respective trajectories to move forward at each time stamp. Two marked coordinates in the figure demonstrates the endpoints of the two generated trajectories.

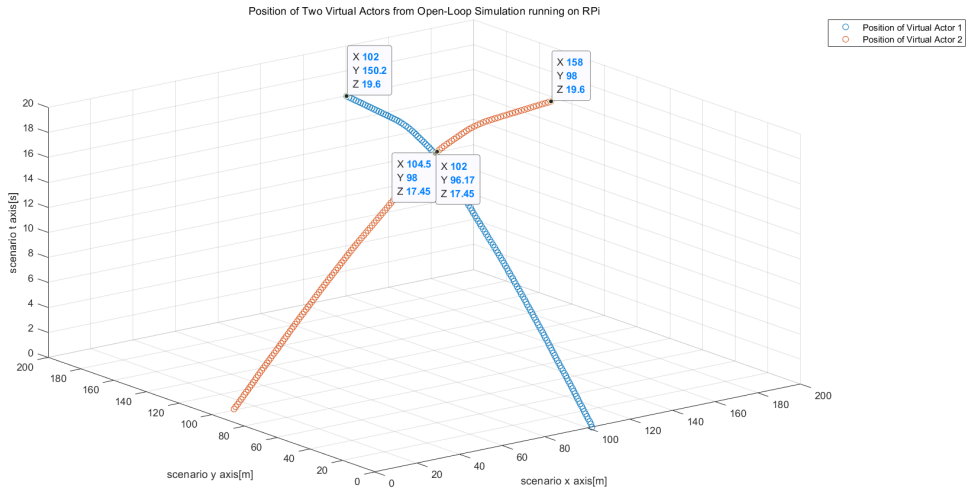

Figure 4.3: Illustration of the positions of two virtual actors while following the trajectories shown in figure 4.2 at different time stamps in the open-loop simulation running on Raspberry Pi. Two nearby marked coordinates in the center of the figure demonstrates that the two virtual actors meet or collide with each other at 17.45s around the intersection point (102, 98) as shown in figure 4.1. Besides, the other two marked coordinates deliver the information that two virtual actors respectively stop in front of their corresponding endpoint of trajectories at 19.6 s.

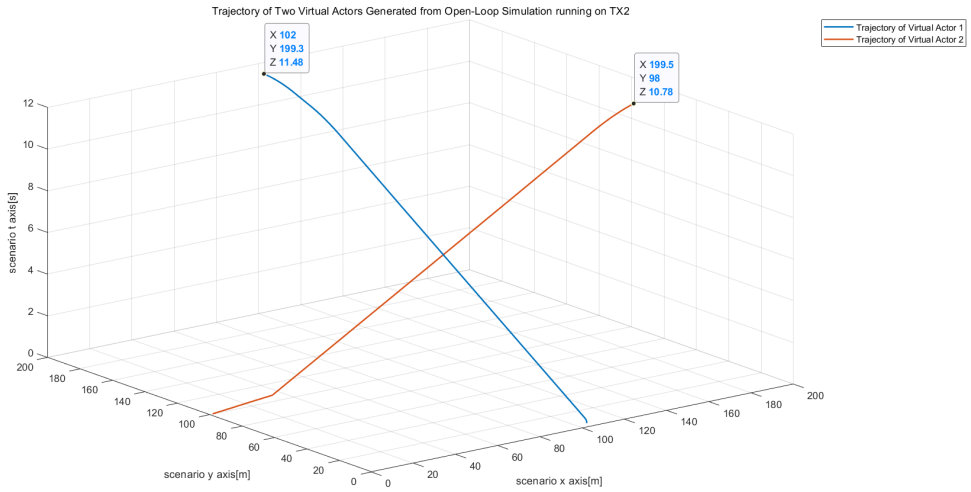

Figure 4.4: Illustration of two trajectories with related time information, which is plotted as Z axis, generated from the open-loop simulation running on NVIDIA TX2. Compared with two endpoints of generated trajectories shown in figure 4.2, we can notice that the coordinate value of Z axis is decremented by averaged 7.43 s, which proves that the open-loop simulation running on NVIDIA TX2 to generate trajectories is faster than open-loop simulation running on Raspberry Pi.

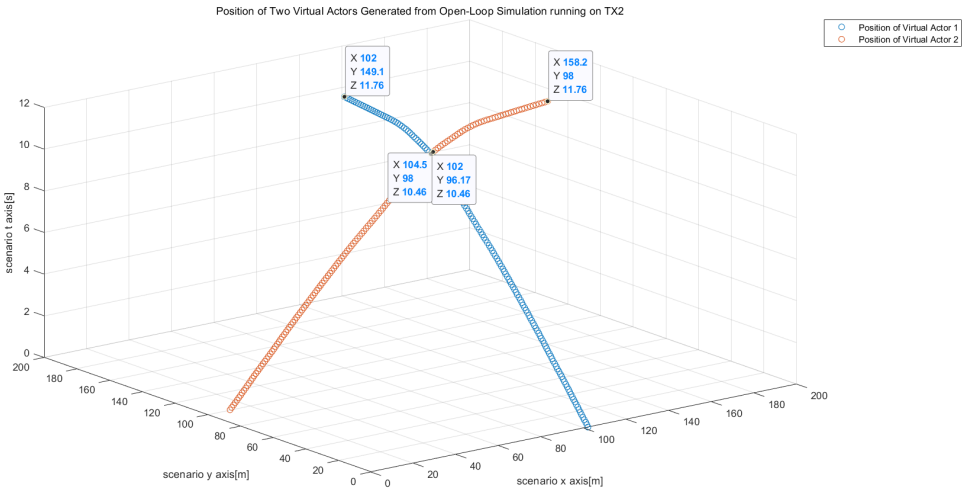

Figure 4.5: Illustration of the positions of two virtual actors while following the trajectories shown in figure 4.4 at different time stamps in the open-loop simulation running on NVIDIA TX2. Compared with marked coordinates of colliding and ending positions of two actors as shown in figure 4.3, we can also observe a drop of consumed simulation time for two actors to collide with and stop moving in this case.

4.2.2 Closed-Loop Simulation Test Results

The closed-loop simulation results are shown below after completing the set of tests as mentioned in section 4.1.2.

Figure 4.6 and figure 4.8 illustrate the desired trajectory for two virtual actors generated from the closed-loop simulation running on Raspberry Pi and NVIDIA TX2 respectively, while figure 4.7 and figure 4.9 show the simulated positions of two virtual actors in the scenario. The real time stamp for each position point is calculated from the system internal clock and also added as the Z axis to all of the figures, which assists to restore the dynamic simulation process. Furthermore, the specific coordinate and time values of critical trajectory or position points, such as the end points and the intersection points, are also marked in the figures, which could be regarded as the proof of the success of the closed-loop test. Figure 4.10 shows the forward velocities of two virtual actors along their roads at different simulation time stamps. Due to the virtual actor 1 owning the functionality of velocity adaption in its model, we can therefore view an obvious deceleration from its initial velocity 8.33m/s to 5.57m/s in the end but with a significant speed fluctuation in the beginning of the process. In contrast, the velocity of the second actor, which does not contain the functionality to adapt, is a constant value for most of the simulation time but also with a few jitters in the beginning.

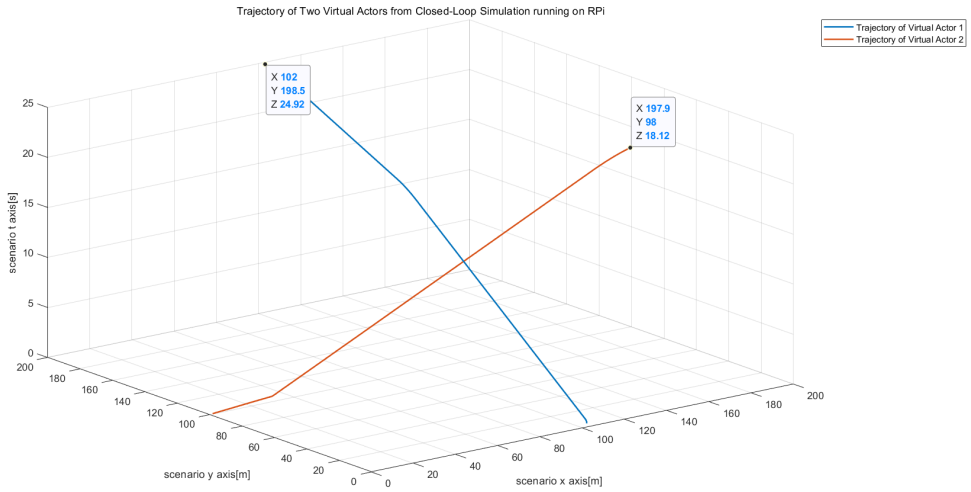

Figure 4.6: Illustration of two trajectories with related time information, which is plotted as Z axis, generated from the closed-loop simulation running on Raspberry Pi. Compared with two endpoints of generated trajectories shown in figure 4.2, we can find that the coordinate value of Z axis is incremented by averaged 5.79 s, which reflects that the closed-loop simulation takes a longer time than open-loop simulation running on Raspberry Pi.

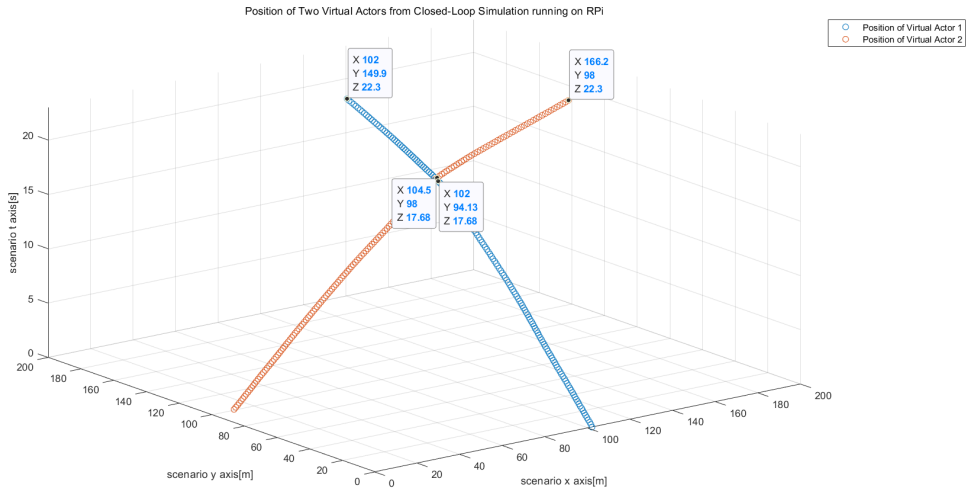

Figure 4.7: Illustration of the positions of two virtual actors while following the trajectories shown in figure 4.6 at different time stamps in the closed-loop simulation running on Raspberry Pi. When we concentrate on the two marked coordinates in the center of figure in this case, in contrast to what is presented in figure 4.3, we can find that at two virtual actors do not meet each other around intersection point at 17.68s, which proves that collision is successfully avoided in closed-loop simulation.

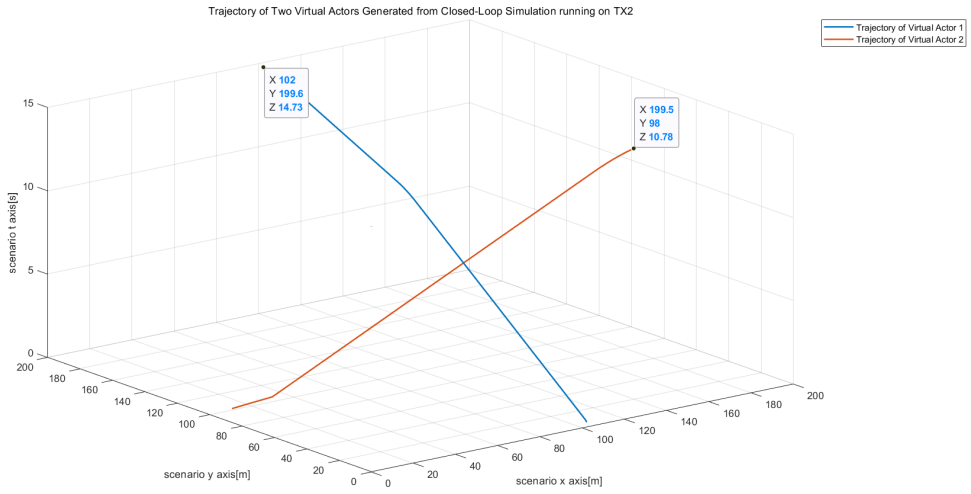

Figure 4.8: Illustration of two trajectories with related time information, which is plotted as Z axis, generated from the closed-loop simulation running on NVIDIA TX2. Compared with two endpoints of generated trajectories shown in figure 4.4, we can find that the coordinate value of Z axis is incremented by 3.25 s, which reflects that the closed-loop simulation takes a longer time than open-loop simulation running on NVIDIA TX2 but still less than running on Raspberry Pi.

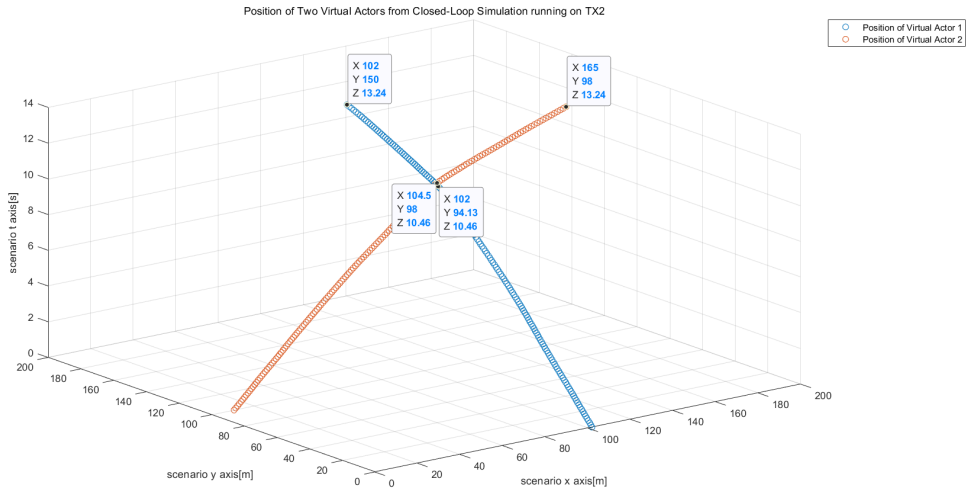

Figure 4.9: Illustration of the positions of two virtual actors while following the trajectories shown in figure 4.8 at different time stamps in the closed-loop simulation running on NVIDIA TX2. In this case, we can also observe the avoidance of collision around intersection point between two virtual actors at 10.46 s as well as the drop of consumed simulation time compared with closed-loop simulation running on Raspberry Pi.

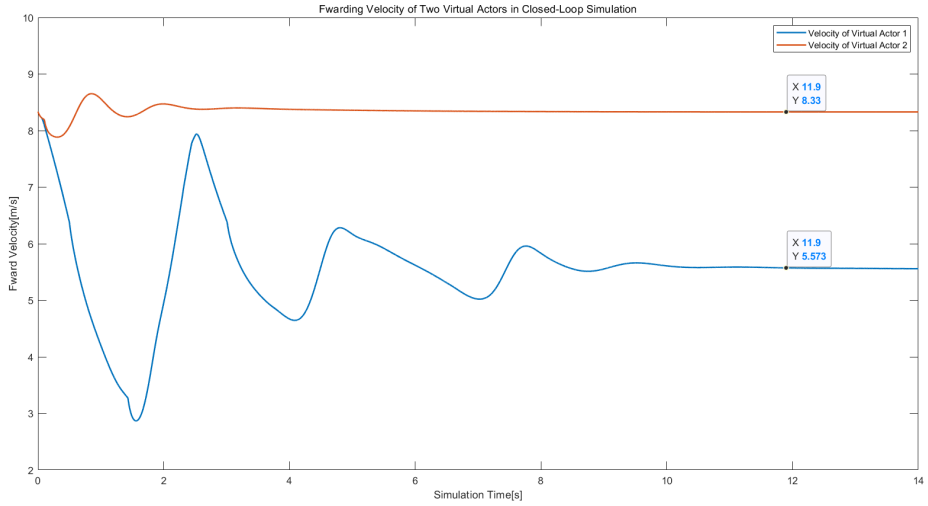

Figure 4.10: The forward velocities of two virtual actors at different time stamps in the closed-loop simulation running on NVIDIA TX2.

When it comes to the closed-loop test with one RC Car involved to replace the virtual actor 2 in the previous test, the focus is then put on whether the system can meet the real-time constraint at each sampling step and the behaviour synchronization of two types of actors. Figure 4.11 shows the execution time of one step of simulator running on two platforms. The worst case execution time for two cases on corresponding platform is also highlighted in the figure. The delay of UDP packets send and receive on the server side during simulation is shown in figure 4.12. The average of delay is calculated by 100 delay samples and shown as the red line in the figure.

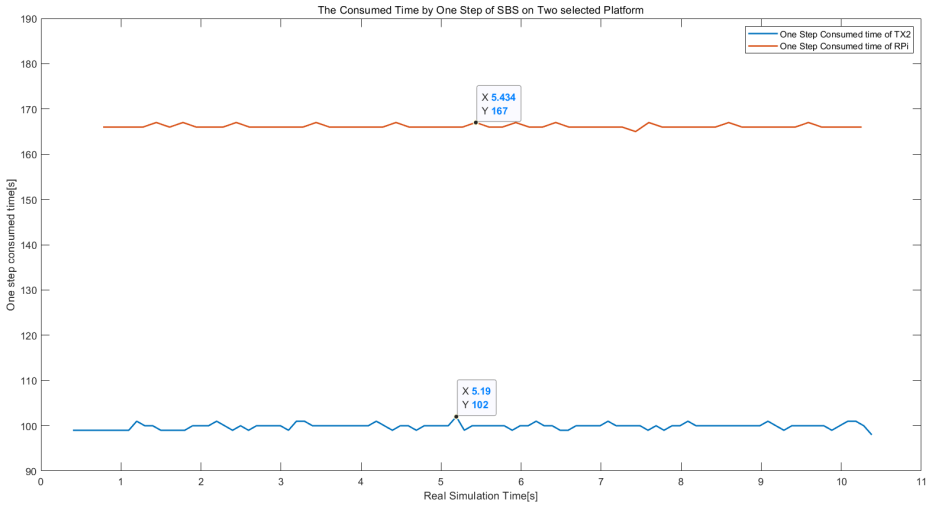

Figure 4.11: Execution time consumed by one step of SBS on the two selected platforms.

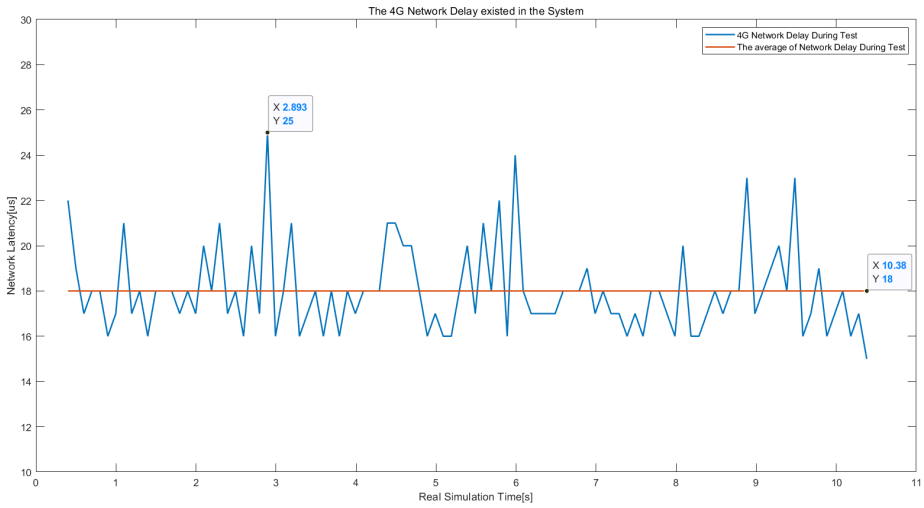

Figure 4.12: Delay between send and receive of UDP packets on server side in the closed-loop simulation test.

Figure 4.13 demonstrates the desired trajectory of RC Car and virtual actor in the simulation running on Raspberry Pi. The generated desired trajectory for RC Car is sent via UDP from the server to the RC control station. As figure 4.14 presents, the RC Car can search and follow the received way-points of trajectory through a virtual red cycle with a certain radius, which can be viewed on the map of RC control station in real time.

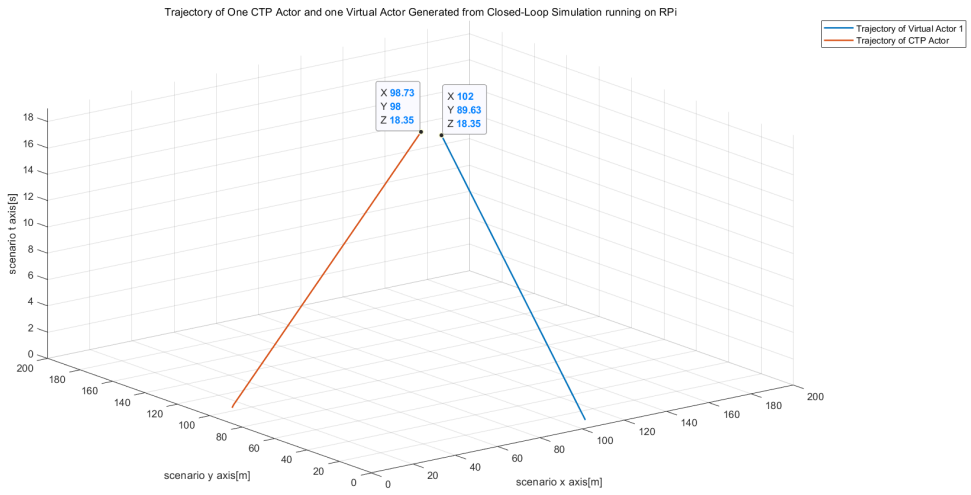

Figure 4.13: The generated trajectories for one virtual actor and RC Car from the closed-loop simulation running on NVIDIA TX2, which demonstrates that trajectories for two actors would be terminated to generate if the simulator find out two actors would potentially collide with each other soon.

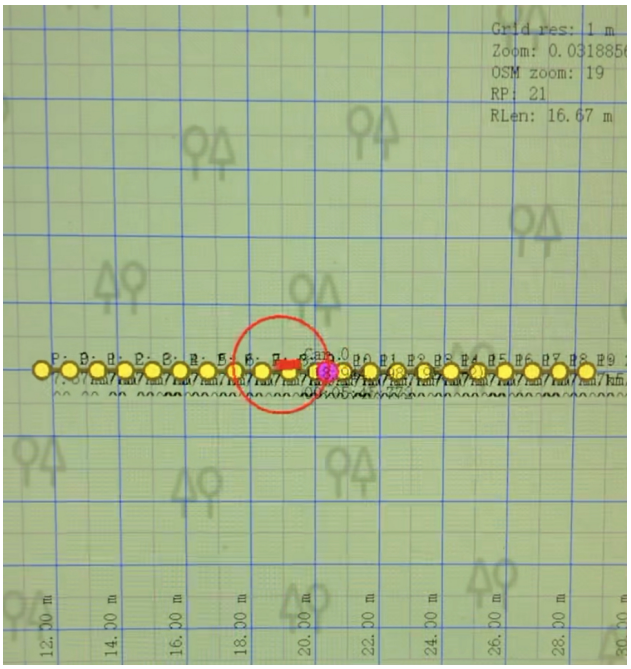

Figure 4.14: Figure depicting the RC Car following the received trajectory in the map of control station.

5
Conclusion

Based on the testing results presented in Chapter 4, we can have a discussion about the real-time performance of the server based simulator running on two different platforms and also the synchronization we achieved between two types of actors in simulation. Finally, the conclusions about the whole simulation system and the thesis will be drawn.

5.1 Discussion

Our first main focus is to reduce the execution time of the traffic simulator to fulfill real time constraints. This is necessary because the simulator is to execute a scenario with a physical actor involved and controlled by the server in real time. It was noted to be failing in simulations carried out using Simulink models on MATLAB. Besides, in case of the simulator running on windows, it was not easy to determine the exact execution time of each task because of its inconsistent time consumption.

To investigate the worst case execution time of each task, as a prototype we executed the simulator on a real time platform providing it with a dedicated processor. At first we used Raspberry Pi, with a 32-bit ARMv7 architecture. After which we noticed that the execution time was increasing linearly for each step due to the function logic implemented in the traffic manager task. We optimized the functionality here and it can be seen in the results in Chapter 4, one simulation step on Raspberry Pi takes around 167ms on an average. But the sampling rate of the tasks of the simulator is 100ms, which means each cycle is expected to be completed within 100ms. We used a monolithic architecture, in which the tasks were scheduled with a period of 100ms. Internal to the tasks to divide the complexity of computations among different cores of hardware platform and also to improve execution time, we used multi-threading implemented by OpenMP as discussed in Chapter 3. Even though it was observed that the execution time of the simulator on Raspberry Pi was lesser than that on Simulink, which consumed 45s for a 18s scenario, it was not sufficient to meet the deadlines. Therefore, we then used a faster 64-bit ARMv8 platform from NVIDIA to investigate the time consumption. As seen from the results in chapter 4, the timing constraints were successfully met on this platform. The tests we have performed are all with two actors, we were unable to test for three actors as the current functionality of trajectory planner does not support three actors. With addition of more actors they can be run in parallel so as to keep the execution time within required limits.

Besides, from figures 4.5, 4.9, 4.3 and 4.7 showing positions of two virtual actors in both open-loop simulation and closed-loop simulation, we can view the coordinates of intersection points where the two actors are located around the intersection area at the same time stamp. Considering the length and width of virtual actor model is 5.0m and 2.0m respectively, which would be added as the offset to validate whether the two virtual actors will encounter a collision or not. Comparing figures 4.5 and 4.9, or figures 4.3 and 4.7, we can see that at the same time and with same X coordinate parameter, the Y coordinate values are different in the two figures. When we compare the Y coordinate value of virtual actor 1 with the result of Y coordinate value of virtual actor 2 minus the sum of half length and half width of vehicle model in two cases, we can prove that in open-loop simulation, the two virtual actor will collide in the intersection area while in closed-loop simulation, the collision can be avoided since the velocity of virtual actor 2 has been adapted. In the closed-loop test with RC Car and one virtual actor, we modify the velocity adaption function in the simulator. In this case, when the distance between two actors are within 15 meters, the desired velocity for two actors will be set to half of its previous velocity so as to let them slow down. However, if the distance is less than 10 meters, the desired velocity will be set to 0 and the desired trajectory will also stop at that position to let both of actors hard break, as shown in figure 4.13, the RC Car and virtual actor will both stop moving when their distance is below the potential collision threshold.

Moreover, the synchronization between the actors is achieved by setting them all to refer to a global reference time using GPS on the server, additionally, adding an offset obtained by considering an average of the time consumed to send and receive 100 UDP packets over the WiFi in corporation, which is 18 microseconds during the test. The send and receive time is inclusive of not only the network delay but also the time lapse in the communication between the RC control station running on a desktop and the RC Car.

Furthermore, after comparing the timing performance on two selected hardware platforms, it can be observed that the Raspberry Pi 3B+ module is under-performing for the computational complexity required for SBS, on the other hand, NVIDIA TX2 is over-performing for the execution of tasks that are required to be run in sequence. In this case, the high level of parallelism offered by the GPU on this platform is not used. Besides, the high cost of this development platform makes it inefficient for our application. As a middle ground between these two options, a more suitable platform would possess a processing unit with 64-bit ARM quad-core architecture, operating frequency greater than 2.0 GHz, 4 GB RAM memory as well as 2 MB cache would also be a plus. According to the above specifications and our investigation, some recommended economical alternatives could be Raspberry Pi 4 Model B or Libre Computer AML-S905X-CC (Le Potato) and similar platforms.

5.2 Conclusion

With the increasing complexity of AD and ADAS functions required by the highly automated vehicles in current market, safety and real-time performance testing and

validating in simulations turns out to be more and more significant and mandatory. The simulation with VIL and virtual actor introduced could provide us with a more realistic yet a safer environment to view the vehicle's behaviour and validate the functionality. In this thesis, we managed to develop a light-weight server based traffic simulator, which can run on a real-time hardware platform to execute a predefined intersection scenario with both virtual and physical actor involved.

The simulator was previously build by Simulink models and run on Windows system, which unfortunately failed to satisfy real-time requirements when it tries to interact with physical actors on the test track. Thus, to resolve this problem, we converted the Simulink model to Embedded C code, scheduled, optimized, cross complied and deployed it onto a standalone real-time platform. In this case, Raspberry Pi 3B+ module and NVIDIA jetson TX2 were selected as the hardware platform to run the simulator which are also compared in the performance tests. A server which could communicate with physical actors through UDP packets over the WiFi is developed and also deployed onto the platform along with the simulator. A global reference time from a GPS HAT is added to the server for synchronization between actors as well.

According to the test results from open-loop and closed-loop simulation, simulator running on NVIDIA TX2 could not only meet the simulation deadline but also satisfy the real-time constraints determined by the sampling frequency of simulator in closed-loop simulation, while the test with simulator run on the Raspberry Pi failed due to the longer execution time per step compared with NVIDIA TX2, which proves that the simulator requires a more powerful hardware support than Raspberry Pi can provide so as to finish execution within 100 ms per step. The delay between UDP packets send and receive on the server side was measured in simulation and the average of delay has been taken into consideration so as to achieve the time synchronization between the simulator and physical actor. According to the measured results, the transmission delay of UDP packets between server and actor client does not play a major role in the existing system's latency. The execution time consumed by the module Trajectory Planner of Steer-By-Server, which generates safe trajectories for actors to avoid the collision, currently contributes to most part of the system latency. Optimizing the specific algorithm inside this module to reduce its time complexity as well as utilizing multi-threading method to accelerate helped improve time performance of the simulation system. Therefore, in closed-loop simulation test, which requires more strict real-time constraint than open-loop simulation test, RC Car can follow the received trajectory generated from the simulator and send back its own state to the server as requested in real time, meanwhile the virtual actor in simulator can adapt its velocity to avoid collision at the intersection point after detecting the position of RC Car in simulation.

6
Future Work

There exist a few limitations of current thesis work. There is scope for further improvement, some of the areas we identified are as follows.

- The code running on the embedded platform is compiled for a single scenario, as a future work it would be feasible to have a provision to configure different scenarios without having to compile the code each time.
- The code converted from Simulink blocks in MATLAB using the embedded coder is not fully optimized and we noticed several instances where the memory usage and execution effort can be reduced.
- We built a prototype with the main aim to study and reduce latencies in the systems, improve execution time to meet the sample time constraints and have a predictable and consistent behaviour, therefore couldn't focus on testing the system on the test track with an actual vehicle's ECU. The system can be further tested by injecting virtual objects generated by the simulator to the vehicles ECU along with other physical objects on the track in closed loop.
- The RC car that we used could only be operated remotely through a control station GUI, bypassing this control station GUI and communicating directly with the RC car would help in better synchronization with the RC car.
- All the tests we performed are with two actors as the trajectory planner module posed limitation on supporting more than two actors, investigating the time consumption and if the system will miss deadlines would be a good area to focus on in future work.
- We visualized the executed scenario by plotting the log data, for viewing the scenario being executed in real time, it would be useful to have a visualization tool integrated to the system.

Bibliography

[1] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+*. https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[2] NVIDIA Corporation. *NVIDIA Jetson, The AI Platform for Autonomous Machines*. https://developer.nvidia.com/embedded/develop/hardware.

[3] Y. Sahraoui, A. Ghanam, S. Zaidi, S. Bitam, and A. Mellouk. Performance evaluation of tcp and udp based video streaming in vehicular ad-hoc networks. In *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pages 67–72, Oct 2018.

[4] Adafruit Industries. *Adafruit Ultimate GPS HAT for Raspberry Pi*. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-hat-for-raspberry-pi.pdf.

[5] Isaac D. T. de Souza, Sergio N. Silva, Rafael M. Teles, and Marcelo A. C. Fernandes. Platform for real-time simulation of dynamic systems and hardware-in-the-loop for control algorithms. *Sensors*, 14(10):19176–19199, 2014.

[6] Khondokar Fida Hasan, Yanming Feng, and Yu-Chu Tian. GNSS time synchronization in vehicular ad-hoc networks: Benefits and feasibility. *CoRR*, abs/1811.02741, 2018.

[7] Per Ohlsson and Vanessa Olsson. Server based closed loop trajectory control. Master's thesis, Department of Electrical Engineering, Mechatronics, Chalmers University of Technology, 2018.

[8] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll. Exploring execution schemes for agent-based traffic simulation on heterogeneous hardware. In *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications*, pages 1–10, Oct 2018.

[9] M. Shedeed, G. Bahig, M. W. Elkharashi, and M. Chen. Functional design and verification of automotive embedded software: An integrated system verification flow. In *2013 Saudi International Electronics, Communications and Photonics Conference*, pages 1–5, April 2013.

[10] T. Bokc, M. Maurer, and G. Farber. Validation of the vehicle in the loop (vil); a milestone for the simulation of driver assistance systems. In *2007 IEEE Intelligent Vehicles Symposium*, pages 612–617, June 2007.

[11] Ronakorn Soponpunth. Fusing data from multiple vehicles into a common picture. Master's thesis, Department of Computer Science and Engineering, Chalmers University of Technology, 2016. 77.

[12] Alexander Berg and Andreas Käll. Track-to-track fusion for multi-target tracking using asynchronous and delayed data. Master's thesis, Department of Signals and Systems, Chalmers University of Technology, 2017.

[13] Andreas Hahlin and Anton Olsson. Centralized collision avoidance system for automated vehicles. Master's thesis, Department of Electrical Engineering, Chalmers University of Technology, 2017.

[14] Junhua Chang and Yichang Liu. Evaluation of hardware-in-the-loop framework for intelligent safety systems verification. Master's thesis, Department of Mechanics and Maritime Sciences, Vehicle Safety, Chalmers University of Technology, 2018.

[15] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. Exploring execution schemes for agent-based traffic simulation on heterogeneous hardware. pages 1–10, 10 2018.

[16] Madhu Monga, Daniel Roggow, Manoj Karkee, Song Sun, Lakshmi Kiran Tondehal, Brian Steward, Atul Kelkar, and Joseph Zambreno. Real-time simulation of dynamic vehicle models using a high-performance reconfigurable platform. *Microprocessors and Microsystems*, 39(8):720 – 740, 2015.

[17] Michel Dubois, Murali Annavaram, and Per Stenström. *Parallel computer organization and design.* Cambridge University Press, 2012.

[18] Blaise Barney, Lawrence Livermore National Laboratory. What is openmp? `https://computing.llnl.gov/tutorials/openMP/#Introduction`.

[19] Inc Eclipse Foundation. *Eclipse C/C++ Development User Guide.* `https://help.eclipse.org/2019-12/index.jsp`.

[20] Arm Limited (or its affiliates). *GNU Toolchain for Arm processors.* `https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain`.

[21] Douglas E. Comer. *Internetworking with TCP/IP.* Pearson, 2013.

[22] The Qt Company. Getting started with qt. https://doc.qt.io/qt-5/gettingstarted.html.