



Technische Hochschule
Ingolstadt

International Automotive Engineering
Master Thesis

Simulating the Virtual Environment for
Autoware based Automated Driving Vehicle
in IPG CarMaker

Author Name:

Manoj Kambampati

First Examiner

Prof. Ondrej Vaculin, Ph.D.

Second Examiner

Prof. Dr. rer. nat. Armin Arnold

Registered on:

30.07.2023

Submitted on:

12.10.2023

Declaration

I hereby certify that this thesis is my independent research and understanding. I have not published for academic evaluation elsewhere, and I have given complete credit to all the citations and sources that I considered.

Date: 12.10.2023

Place: Ingolstadt

k. manoj
Signature

Acknowledgement

I would like to express my eternal gratitude to Prof. Dr. Ing. Ondrej Vaculin for his extremely valuable guidance, encouragement, and dedicated support throughout the accomplishment of my thesis. I am privileged to have Prof. Dr. rer. nat. Armin Arnold as my second examiner. I am grateful to Mr. Thiago de Borba for his valuable technical support.

I would also like to express my heartfelt gratitude to my family and friends, whose firm faith in my abilities has provided constant encouragement and promoted my personal progress. I am also grateful to my country, India for providing me with a solid foundation and constant encouragement when I pursued my Master's degree in Germany.

Abstract

In the rapidly changing Automotive industry, particularly in the areas of Automated Driving Systems (ADS) and Advanced Driver Assistance Systems (ADAS), the demand for effective virtual testing methods has increased significantly. Manual testing of automated vehicles, such as ANTON, on proving grounds needs remarkable costs and triggers substantial safety risks. To tackle such challenges, the study commenced by conducting a comprehensive analysis of different co-simulation platforms. Consequently, Autoware.AI-Carla has been identified as the leading co-simulator, while CarMaker was distinguished for its exceptional vehicle dynamics simulation capabilities. The extensive evaluation of the Aslan-Carmaker project identified complexities and challenges encountered during the reconstruction, which provided a solid foundation for the Autoware.AI-Carmaker project. With this detailed understanding, The primary objective of this research was to create a reliable ROS bridge between Autoware.ai and IPG CarMaker, enabling efficient interaction. Even in its early stages of development, the bridge has shown its potential by successfully controlling the Car in the CarMaker simulation through Autoware's 'control' tab. After rigorous research, this thesis has successfully achieved all its objectives, providing an authoritative resource for future research endeavours. This research not only demonstrates the potential of virtual testing in automated driving but also establishes a connection between testing platforms for seamless communication, paving the way for future advancements in the field.

Contents

| | |
|--|-----------|
| List of Abbreviations | vi |
| 1 Introduction | 1 |
| 1.1 Introduction to ANTON vehicle | 2 |
| 1.2 Importance of Virtual Methods for Autonomous Driving Development | 3 |
| 1.3 IPG CarMaker and Autoware.AI, Need for ROS Bridge | 3 |
| 1.4 Selection of Autoware.AI over Aslan | 4 |
| 1.5 Objectives of the Thesis | 5 |
| 1.5.1 Analysing the Aslan-CarMaker project | 5 |
| 1.5.2 Analyse the existing methods for coupling simulation tools with Autoware.AI | 5 |
| 1.5.3 Propose the structure of Autoware.AI-IPG CarMaker interface | 5 |
| 1.5.4 Implementation of the interface | 6 |
| 2 State of Art | 7 |
| 2.1 Introduction to IPG CarMaker | 7 |
| 2.2 Integration of ROS with IPG CarMaker for Automotive Co-Simulation | 8 |
| 2.3 Aslan-CarMaker GitHub repository by the UK IPG Automotive | 9 |
| 2.4 Autoware User Manual | 10 |
| 2.5 LGSVL Simulator Integration with Autoware | 12 |
| 2.6 ROS-Autoware-CarMaker Simulation Architecture for Autonomous Vehicles . . . | 13 |
| 2.7 Comparison of different simulators and AD stack software | 14 |
| 2.8 Development and Validation of ADAS Applications with the CARLA Simulator in ROS | 16 |
| 2.9 Overview of the Planning Support Tool for Autonomous Driving | 16 |
| 3 Research Methodology | 18 |
| 3.1 Required and Supported software versions | 19 |
| 3.1.1 Installation | 19 |
| 3.2 Reconstruction and Evaluation of the Current Aslan-CarMaker Project | 20 |
| 3.3 Gazebo co-simulation with Aslan AD stack | 21 |
| 3.4 Creating ROS Bridge (main.cpp bridge) | 24 |
| 3.4.1 About the code in main.cpp | 26 |
| 3.5 Implementation of Autoware-CarMaker Communication | 28 |

| | | |
|----------|--|-----------|
| 3.5.1 | Initialise the ROS Environment and roscore | 28 |
| 3.5.2 | Initializing CarMaker and Selecting the Project | 28 |
| 3.5.3 | Configuring CarMaker Executable | 30 |
| 3.5.4 | Start CMRosIF | 30 |
| 3.5.5 | Initializing CarMaker Keyboard Control | 31 |
| 3.5.6 | Configuring and loading the open world test run in CarMaker GUI | 32 |
| 3.5.7 | Testing the working of Keyboard control script with CarMaker | 33 |
| 3.6 | Information about ROS topics before starting Autoware and bridge | 34 |
| 3.7 | Launch Autoware User Interface | 35 |
| 3.8 | Launch ROS bridge | 35 |
| 3.9 | Information about ROS topics after starting Autoware.AI and bridge | 36 |
| 3.10 | Start Simulation with the bridge running | 37 |
| 4 | Results and Discussion | 45 |
| 4.1 | Evaluation of simulators and Co-Simulation Platforms | 45 |
| 4.2 | Discussion about Aslan-CarMaker project | 45 |
| 4.2.1 | Building ROS Workspace: | 46 |
| 4.2.2 | Compatibility with CarMaker 11.0: | 46 |
| 4.2.3 | CarMaker Keyboard Control Script: | 46 |
| 4.2.4 | Errors in Gazebo Simulation: | 46 |
| 4.3 | Discussion about ROS Bridge Algorithm and Autoware-CarMaker project Evaluation | 48 |
| 5 | Conclusion and Future Scope | 50 |
| 5.1 | Conclusion | 50 |
| 5.2 | Future Scope | 51 |
| | Appendix | 56 |

List of Abbreviations

| | |
|---------|--|
| ADstack | Autonomous Driving Stack |
| ADAS | Advanced Driver-Assistance System |
| API | Application Programming Interface |
| AV | Autonomous Vehicle |
| BMBF | Germany Federal Ministry of Education and Research |
| CANbus | Controller Area Network bus |
| CARLA | Car Learning to Act |
| CarSIM | Car Simulator |
| GIS | Geographic Information Science |
| GNSS | Global Navigation Satellite System |
| HDRP | High Definition Render Pipeline |
| HIL | Hardware-In-the-Loop |
| HPC | High-Performance Computer |
| IMU | Inertia Measurements Unit |
| LGSVL | LG Silicon Valley Lab |
| LIDAR | LIght Detection And Ranging |
| MIL | Model-In-the-Loop |
| NDT | Normal Distribution Transform |
| OEMs | Original Equipment Manufacturers |
| PCD | Point Cloud Data |
| RADAR | Radio Detection And Ranging |
| ROS | Robotic Operating System |
| RViz | ROS Vizualisation |
| CMRosIF | CarMaker ROS Interface |
| SIL | Software-In-the-Loop |
| SUMO | Simulation of Urban Mobility |
| VIL | Vehicle-In-the-Loop |

Chapter 1

Introduction

The development of autonomous vehicles has accelerated significantly in recent years and extensive research is being done to enhance their overall performance, efficiency, and safety. However, testing and development of such vehicles as ANTON on proving grounds is expensive, time-consuming and poses potential safety risks. To mitigate these issues, several virtual testing simulation tools have been developed, among which IPG CarMaker, a Germany-based software, is one of the most prominent ones. The software provides a simulated environment where a virtual vehicle can be tested in different scenarios. Autoware.AI, on the other hand, is a powerful open-source software containing an autonomous driving stack with various modules for sensor integration, perception, localization, motion planning, and control, enabling the vehicle to execute different manoeuvres. However, to enable effective communication between IPG CarMaker and Autoware.AI as shown in figure 1.1, a ROS bridge needs to be created. Such a bridge would facilitate the exchange of commands and data between the two software packages, enabling the testing and development of autonomous vehicles in a simulated environment.

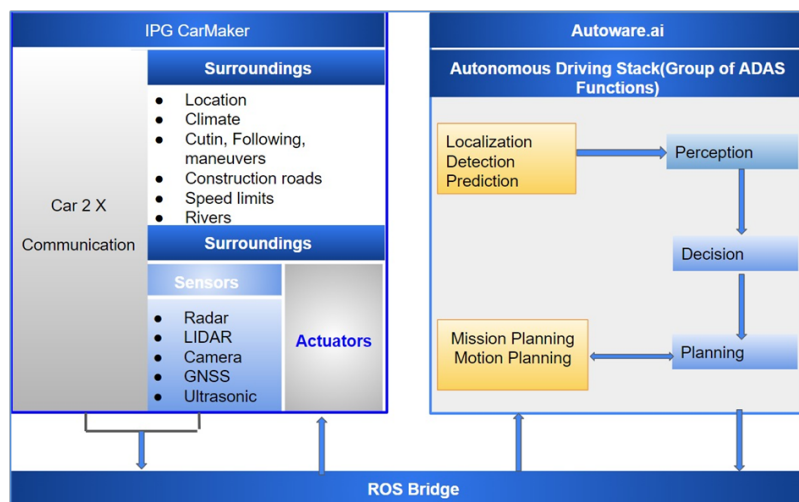


Figure 1.1: IPG CarMaker and Autoware.AI integration using ROS

Recently, a number of research studies have concentrated on the integration of simulation software with software stacks like Autoware.AI, and they have demonstrated that doing so gives important benefits for the development and testing of autonomous driving features. These systems integration enables the creation and testing of sophisticated independent driving capabilities such as automated lane changes, traffic light recognition and obstacle avoidance. These studies have also highlighted the importance of a robust and efficient communication system between IPG CarMaker and Autoware.AI [1].

Overall, the integration of IPG CarMaker with Autoware.AI offers a powerful platform for developing and testing autonomous driving functions [2]. The use of virtual simulations provides a safe and controlled environment for testing, while the integration of these software systems allows for the development of complex and sophisticated autonomous driving functions.

1.1 Introduction to ANTON vehicle

ANTON, originally a Renault Twizy, has been modified to serve as an experimental platform for the study and testing of networked and autonomous driving vehicles. The platform has been equipped with longitudinal and lateral drive-by-wire functions, an open software interface, Car2X communication functions and flexible sensor mounts to enable extensive experimentation and validation of autonomous driving systems. The developed driving functionalities can be tested in both simulated and actual traffic for usage on roads.

The platform has been equipped with longitudinal and lateral drive-by-wire functions, an open software interface, car-to-x communication functions flexible sensor mounts to enable extensive experimentation and validation of autonomous driving systems. The developed driving functionalities can be tested in both simulated and actual traffic thanks to ANTON's receipt of a federal exemption for usage on roads [3].

Using a joystick or computer signals, the drive-by-wire system gives full access to the vehicle's steering, braking and accelerator pedals over the CAN bus. With the ability for users to edit the source code and add new sensors or algorithms, the open software interface, implemented as an open-source autonomous driving stack, offers predefined modules for perception, decision-making, and planning. The CAN bus is used to send the autonomous driving stack's outputs to the drive-by-wire low-level controller, which in turn instructs the actuators. The platform also has adaptable sensor mounts that permit the addition of several sensor types, including cameras, RADAR and LIDAR, enabling the usage of a variety. Figure 1.2 demonstrates the Autonomous Vehicle ANTON with highlighted LIDAR, RADAR, and Camera Sensors [4].



Figure 1.2: Autonomous Vehicle ANTON with highlighted LIDAR, RADAR, and Camera Sensors [4]

1.2 Importance of Virtual Methods for Autonomous Driving Development

Virtual testing and validation methods have become crucial in the development of autonomous driving systems. These methods provide a safe, controlled, and cost-effective way to test and optimise the performance of autonomous vehicles in various scenarios without the need for real-world testing. Virtual methods allow for the creation of virtual environments where different scenarios, such as adverse weather conditions, challenging road layouts and complex traffic situations can be simulated to thoroughly evaluate the performance of autonomous driving systems. Moreover, virtual testing allows for the repetition of tests, precise control over testing conditions, and the ability to collect and analyse vast amounts of data, providing valuable insights into the performance and behaviour of autonomous vehicles [5].

1.3 IPG CarMaker and Autoware.AI, Need for ROS Bridge

IPG CarMaker is a widely used simulation tool that provides a realistic virtual environment for testing and validation of automotive systems, including autonomous driving functionalities. IPG Automotive Group is the registered owner of the trademark CarMaker. It is a high-fidelity vehicle dynamics simulation program that is utilized in the validation and verification prior to actual car experiments. It benefits from having a detailed vehicle model with models of its various systems and functions. Also, it makes it simple and thorough to specify track layout, manoeuvres, load scenarios, test specifications, etc. It also includes models for features that are more often used,

like stability control systems and anti-lock brakes [6]. It enables quick and accurate specification of traffic and other road users, making it particularly helpful for testing active safety systems. There are numerous models of various sensors that can detect things like traffic, car states, traffic signs and other aspects of the road. The IPG CarMaker is able to include a range of traffic aspects in the scenario, such as a stationary long obstruction (a truck), an active road user (an oncoming car), as well as other elements like road shoulders, lane markers, verge posts at the sides of the road, etc. On the other hand, Autoware.AI is a powerful open-source autonomous driving platform that offers a comprehensive stack of modules for developing autonomous driving systems. However, to effectively test and validate autonomous vehicles using IPG CarMaker and Autoware.AI, a seamless communication link needs to be established between the two software packages. This communication link would enable the exchange of commands and data, allowing for the testing and development of autonomous vehicles in a virtual environment using Autoware.AI's powerful capabilities in perception, localization, motion planning, and control. To bridge this communication gap, a ROS (Robot Operating System) bridge needs to be developed, which would establish a robust and efficient connection between IPG CarMaker and Autoware.AI, facilitating seamless communication and testing of autonomous driving functionalities as shown in figure 1.3.

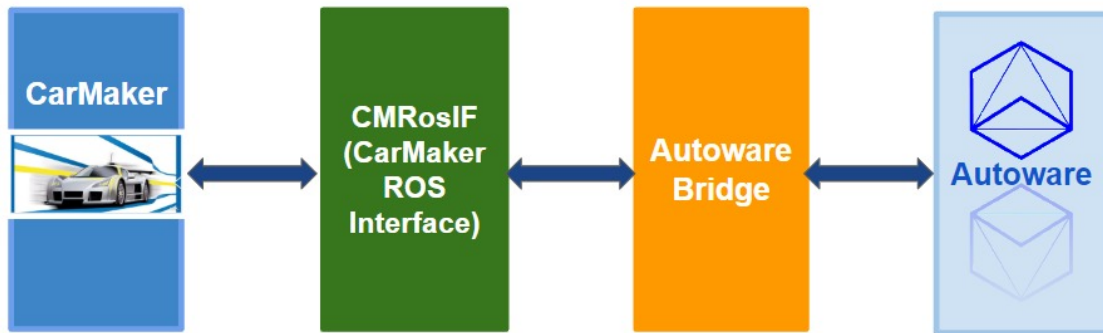


Figure 1.3: Complete CarMaker-Autoware Architecture Overview

1.4 Selection of Autoware.AI over Aslan

In the context of this thesis, the choice to use Autoware.AI as the primary platform for research and development arises from several fundamental benefits it provides. Autoware.AI offers a more robust AD stack, with a full array of tools and algorithms for different aspects of autonomous driving such as perception, localization, motion planning, and control. Its more diverse industrial use, combined with its adaptability to various hardware and sensor combinations, adds to its value. The active supportive community associated with Autoware.AI guarantees that consumers have easy access to information and assistance, making it a viable option. Furthermore, it interfaces well with realistic simulation environments such as Carla, LGSVL etc., simulators allowing for full testing and validation in virtual scenarios. Its ongoing development cycle maintains consistency with the most recent breakthroughs in autonomous driving technology, and its expanded feature set enables a thorough examination of autonomous driving capabilities. Importantly, Autoware.AI's

adaptability expands beyond low-speed urban applications; it is equally good at handling high-speed scenarios, making it a great choice for researchers and developers engaged in a wide variety of autonomous driving scenarios. This versatility to both low and high-speed situations assures that Autoware.AI can be used effectively for a variety of research purposes, including those requiring high-speed capabilities. It is worth mentioning that, Aslan benefited from Autoware.AI's technology and uses Autoware messages. These benefits together make Autoware.AI a strong and versatile choice for research in the realm of autonomous driving, encouraging its selection over Aslan AD stack in the current research.

1.5 Objectives of the Thesis

The objectives outlined in this thesis serve as a roadmap for gaining a comprehensive understanding of the Aslan-CarMaker project and its integration with Autoware.AI. These objectives can be further elaborated as follows:

1.5.1 Analysing the Aslan-CarMaker project

This objective involves a thorough analysis of the Aslan-CarMaker interface and its documentation, which is the communication interface provided by IPG Automotive UK in a GitHub repository. It includes a detailed examination of the documentation covering requirements, installations, project setup, data formats, communication protocols and functionalities of the Aslan-CarMaker interface, with the aim of understanding how it can be effectively utilised to establish a connection with Autoware.AI.

1.5.2 Analyse the existing methods for coupling simulation tools with Autoware.AI

This objective includes thoroughly examining the current methodologies and approaches for integrating additional simulation tools such as CARLA, LGSVL Simulator and CarSIM with Autoware.AI. It entails conducting a thorough examination of relevant literature and research articles in order to understand the existing techniques precisely, as well as discovering ways to adapt and use these techniques for creating a better communication interface between IPG CarMaker and Autoware.AI.

1.5.3 Propose the structure of Autoware.AI-IPG CarMaker interface

Based on the analysis of the existing methods and the requirements of ANTON, this objective involves proposing a structure for the IPG CarMaker with the Autoware.AI interface. This includes defining the necessary data exchange protocols, data formats and software components that will enable seamless communication between IPG CarMaker and Autoware.AI.

1.5.4 Implementation of the interface

This objective includes the practical implementation of the proposed interface between IPG CarMaker and Autoware.AI. It involves several tasks, including selecting a computer with supported hardware and software, installing necessary software packages, setting up directories, cloning project repositories, configuring project settings, building the project, and establishing communication protocols and data formats. The goal is to establish a robust and efficient bridge between the two software packages, ensuring collaboration and uninterrupted interaction.

Overall, the objectives of this thesis aim to develop a robust and efficient ROS bridge between IPG CarMaker and Autoware.AI, enabling seamless communication and testing of the autonomous driving functionalities for ANTON in a virtual environment.

Chapter 2

State of Art

Before diving directly into the development of a ROS bridge between IPG and Autoware.AI, it is crucial to conduct a comprehensive literature review to determine if any prior research has been conducted in this field. This section presents a summary of research efforts associated with the development of ROS bridges in simulation software, with a particular focus on identifying any currently available ROS bridges. The literature review involves an in-depth examination of research articles, theses, and GitHub repositories [7]. These sources are carefully analysed and evaluated in order to gain insights into previous researchers' approaches, methodologies and discoveries in the field of ROS bridges [8]. Detailed explanations and visual aids are provided to ensure a clear knowledge of the principles and approaches used in previous studies.

Furthermore, the literature review critically assesses the strengths and limitations of the evaluated papers and concepts, with the goal of identifying potential areas for further improvement. This analysis gives a thorough understanding of the current state of the art in the field and serves as the foundation for the creation of the proposed ROS bridge in the current research project.

2.1 Introduction to IPG CarMaker

CarMaker is an Integration and Testing Platform specifically designed for the Automotive Industry. Its primary objective is to enhance the efficiency of the developmental and testing processes for cars and light-duty vehicles at several phases, including Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), Hardware-in-the-Loop (HIL) and Vehicle-in-the-Loop (VIL) [9]. It is a versatile solution that can be applied to various domains such as autonomous vehicles, ADAS [10], power train, and vehicle dynamics. It enables the implementation of virtual test scenarios [11]. Simulation facilitates for the accurate representation of real-world testing scenarios, including all environmental factors, in a virtual simulation environment. CarMaker supports the development, calibration, testing, and validation of various vehicle systems, all within realistic scenarios. It offers

a user-friendly interface that simplifies the parameterized process, facilitating the configuration of different settings for enhanced convenience. It has the ability to perform Hardware-in-the-Loop (HIL) tests, which include component-level evaluations and integration tests. The versatility of CarMaker enables its smooth integration throughout the development process, customised to specific testing requirements. It ensures comprehensive documentation of virtual dashboard instruments, IPGmovie (a 3D visualisation tool), IPGControl (a data analysis tool), specific parameters, data and models relevant to the test scenario. The interface offers great flexibility by employing manoeuvres and event-based testing methods. This approach integrates essential sensor technologies, including LIDAR, Ultrasound, RADAR, and Camera, which are crucial for measuring the capabilities of automated and autonomous driving. Despite functioning within time limitations, it produces fast results, facilitating prompt analysis and efficient visualisation. CarMaker improves efficiency by conducting extensive test catalogue runs on High-Performance Computing (HPC) clusters, which reduces testing cycles [12]. CarMaker supports various interfaces and standards, facilitating seamless integration into existing development environments [13].

2.2 Integration of ROS with IPG CarMaker for Automotive Co-Simulation

The Robot Operating System (ROS)(ROS1 & ROS2) is an open-source robotics meta-operating system. Some of its most compelling features include the ability to communicate via Topics with adjacent or distant systems within the same network and control Nodes via Services and Parameters. Systems that transfer data for complex control tasks are called modular systems. It incorporates inter-process message passing, hardware abstraction, implementation of frequently used functionality, low-level device control and package management. Automated driving and advanced driver assistance system (ADAS) technologies are being used more and more in the automotive sector. The open interfaces in IPG CarMaker and ROS facilitate the ease of co-simulation [14]. The interface and workflow that IPG Automotive is developing will enable deep integration of ROS with CarMaker/TruckMaker. IPG Automotive developed the CarMaker ROS Interface, a C++ programming interface that enables the integration of ROS capabilities into the CarMaker executable. Thus, a ROS Node is already present in CarMaker's executable. A shared library that depends on the core ROS libraries, user-defined libraries, and maybe CarMaker libraries is part of the "CarMaker ROS Node". The user-editable C++ code for this shared library can be dynamically loaded into a pre-made CarMaker executable with the use of a CarMaker ROS Interface extension. The ROS-independent CarMaker ROS Interface extension provides an API for the user-accessible CarMaker C code modules and Info file method. The add-on manages the previously mentioned ROS-dependent shared library and enables simple CarMaker ROS Node configuration via a CarMaker Info file (such as Node name or argument remapping). It provides equipment for common CarMaker Hook Points [15], [16]. This integration permits the transfer of data between modular systems for the execution of complex control tasks. Linux must be the user's operating system

of choice in order to use the ROS and CarMaker ROS Interface instead of Windows or Xenomai due to compatibility limitations. The User Guide describes the CarMaker ROS Interface package installation procedure as well as how to use the CarMaker ROS Interface package. The CarMaker ROS Interface enables the use of ROS messages and services within the simulation environment of CarMaker. This makes it possible to create complex simulations for testing and developing autonomous driving systems. The user guide also describes how to test communication with ROS tools and how to construct and execute an example simulation using the CarMaker ROS Interface. All things considered, the CarMaker ROS Interface is a potent tool that enables the integration of ROS features into the CarMaker simulation environment, enabling the construction of complex simulations for the testing and development of autonomous driving systems.

2.3 Aslan-CarMaker GitHub repository by the UK IPG Automotive

The project Aslan-CarMaker focuses on integrating IPG CarMaker and Aslan through ROS melodic. This integration enables virtual testing of low-speed automated driving functions. The project on the GitHub repository is hosted and maintained by IPG Automotive UK. The repository contains code, documentation, and a git modules file that specifies project dependencies for the integration process. The documentation offers guidance on the installation, configuration, and utilisation of the Aslan-CarMaker integration. The repository includes a sample project that shows a fundamental ROS node inside of CarMaker synced with an outside ROS node as shown in figure 2.1.

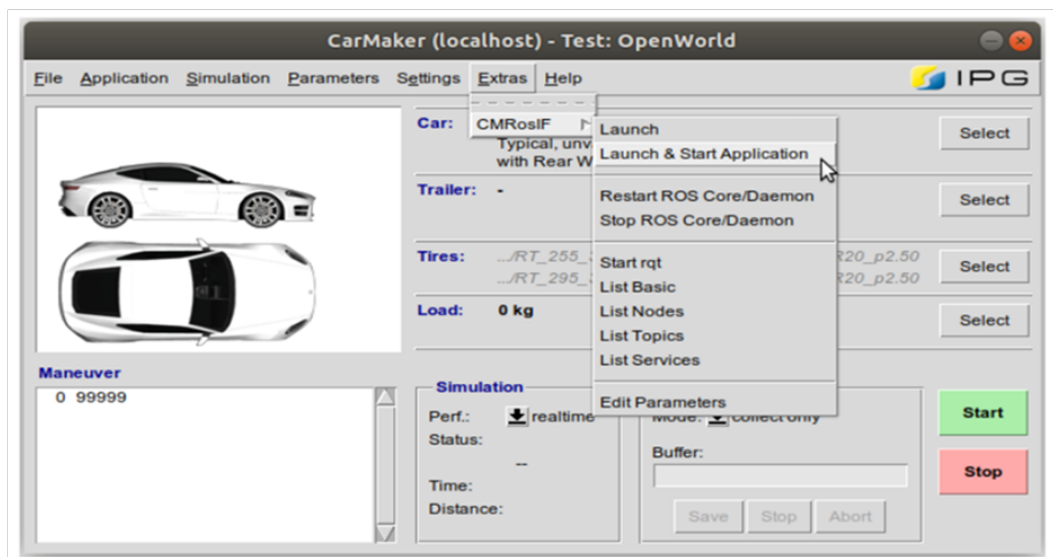


Figure 2.1: ‘CMRosIF’ option within Extras

The Aslan-CarMaker project’s flow can be summarised as follows: The CarMaker simulation environment is integrated with ASLAN through the use of ROS Melodic. A ROS node

is implemented within CarMaker and synchronised with an external ROS node. This allows for the testing of autonomous driving algorithms in a simulated environment before deployment in real-world scenarios. The advantage of integrating this system offers a virtual testing environment for autonomous driving functions. Additionally, it facilitates the development and testing of algorithms in a safe and efficient manner.

Thirdly, it enables an examination of ASLAN’s functionalities and implementation. The Aslan-CarMaker GitHub repository is a valuable resource for code and documentation, facilitating the development of my research. The repository contains documentation and code that explain the integration of CarMaker with Aslan for virtual testing of autonomous driving functions in low-speed applications [7].

2.4 Autoware User Manual

Autoware’s user manual contains a comprehensive description of Autoware, a widely recognised open-source software project designed for the purpose of self-driving vehicles. Autoware is built upon the Robot Operating System (ROS1) and facilitates the incorporation of self-driving vehicles across a range of automotive domains and vehicle classifications. The platform is available on GitHub and serves as a comprehensive development environment for autonomous vehicles. It has a number of features, such as lane-keeping, steering, braking and acceleration. The core aspects of its autonomous driving systems are these parts, along with object detection and localization units. These components, in addition to object detection and localization modules, constitute the fundamental elements of its autonomous driving systems. The basic control and data flow overview in an automated driving vehicle is done by [17] represented in figure 2.2.

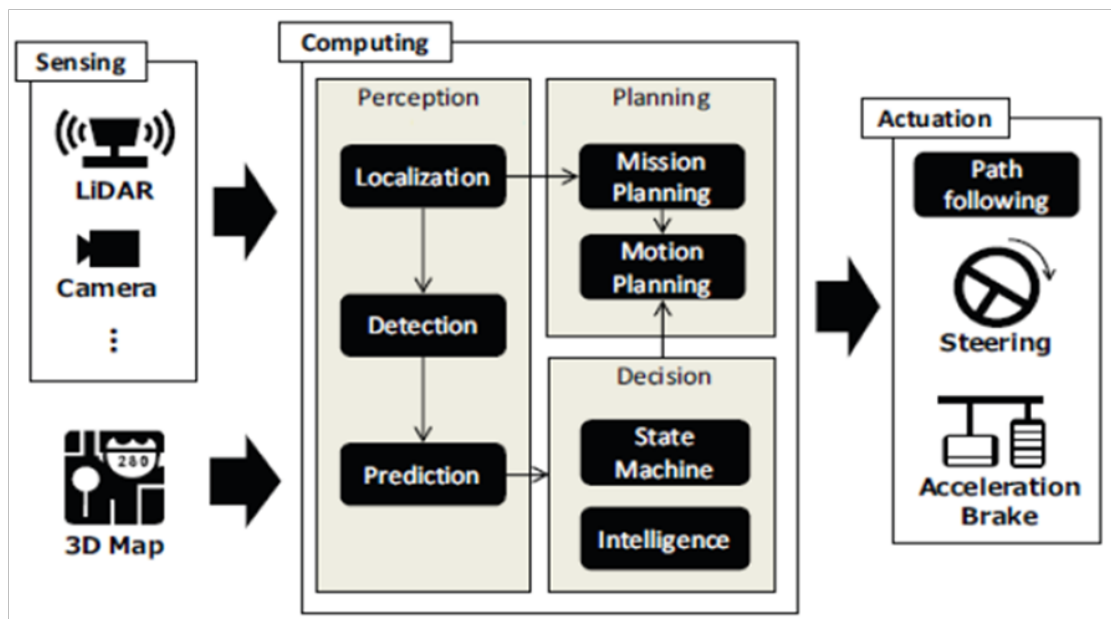


Figure 2.2: Basic Control and Data Flow Overview in an Automated Driving Vehicle [17]

Autoware offers essential functionalities, including 3D map generation, localization, recognition of objects and vehicle manipulation, that are crucial for the development of self-driving vehicles. Ego-vehicle localisation involves the utilisation of on-vehicle cameras and LIDAR sensor technology. Accurate ego-vehicle location is made possible by scan matching using the Normal Distribution Transform (NDT) technique in conjunction with 3D map recognition. The features of Autoware include the detection of surrounding objects using GNSS and LIDAR, assisting in secure navigation.

User Interface: Schematic illustration of the ROS-based Autoware system done by author Architecture [18], [19] represented in figure 2.3.

This guide highlights the use of a 3D map, emphasising how important it is for localising

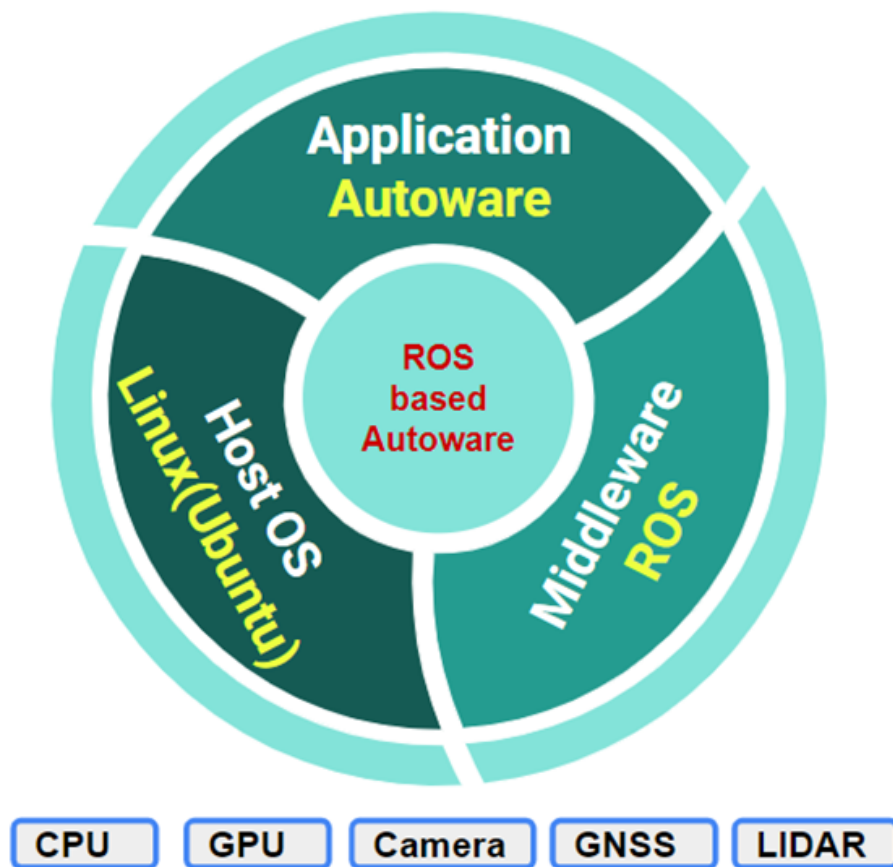


Figure 2.3: Schematic illustration of the ROS-based Autoware system Architecture[18,19].

the ego-vehicle in the deployment of autonomous driving in the actual world. The system uses landmarks to direct its path planning, which helps with stability and trajectory correction. The "Runtime Manager," which makes it simple to control important aspects like localization and path generation, is additionally discussed. RViz provides a wide variety of visualisation capabilities, like object detection, path planning, and path following. These visualisation elements are particularly useful for localisation purposes, specifically in the context of 3D maps. The "AutowareRider" is a cellphone-operated interface designed to enhance motion planning, mission planning, navigation,

and the activation of autonomous driving mode in conjunction with ROS PCs. Finally, Autoware’s adaptability in autonomous driving systems is further enhanced by its capability to project 3D maps onto mobile devices and in-vehicle dashboards.

2.5 LGSVL Simulator Integration with Autoware

The LGSVL Simulator is a realistic autonomous driving simulator. The simulator engine provides comprehensive simulation capabilities that can be seamlessly integrated with Autoware and Apollo, enabling end-to-end, full-stack simulation. In addition, the simulation engine includes simulator utilities that allow users to easily alter sensors, create new programmable objects, replace certain simulator modules, and generate digital twins of specific configurations. The code is open-source and can be found on GitHub. It utilises the Unity game engine for the development of the core simulation. The simulator includes a bridge for exchanging messages between the simulator and an AD stack. The bridge is designed for Autoware, which is a ROS-based platform that inherently supports ROS1, ROS2, and Cyber RT communications. The figure 2.4 below illustrates the workflow of autonomous driving simulation facilitated by LGSVL Simulator [20].

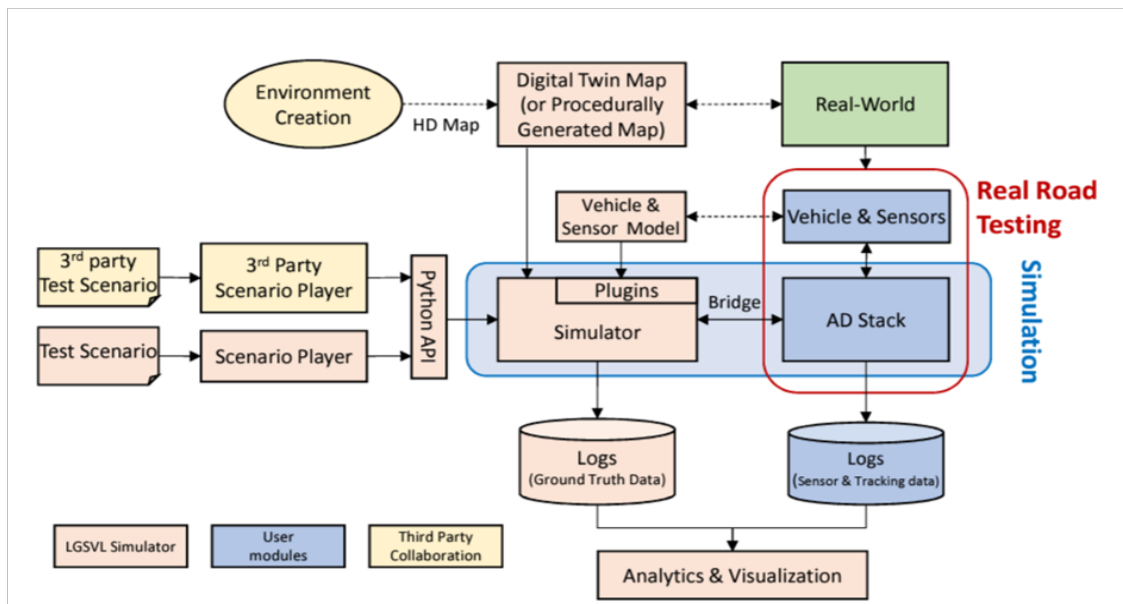


Figure 2.4: LGSVL Simulator’s overall[20]

The user’s autonomous driving (AD) software stack, Autoware, establishes a connection with the LGSVL Simulator by utilising a communication bridge interface. The choice of the bridge is contingent upon the runtime framework employed by the Autoware AD stack. Automation and AI. With open-source ROS and ROS2 bridges, Auto, a ROS and ROS2 application, may communicate with LGSVL Simulator [20].

The LGSVL Simulator utilises the Unity game engine and incorporates the latest Unity technologies, such as the High-Definition Render Pipeline (HDRP), to accurately replicate photo-

realistic virtual environments that closely resemble the real world. The LGSVL Simulator provides extensive options for customising the sensor configuration of the ego vehicle. The simulator currently includes default sensors, such as a camera, LIDAR, Radar, GPS, IMU and various virtual ground truth sensors. In addition, users have the option to develop custom sensors and integrate them into the simulator as sensor plugins. The LGSVL Simulator enables the creation, editing and exportation of HD Maps for existing 3D environments. This function enables users to create and modify distinct high-definition map annotations within a three-dimensional environment. The Python API provided by LGSVL Simulator enables users to interact with and manipulate simulated environments. The LGSVL Simulator facilitates SIL (Software-In-the-Loop) and HIL (Hardware-In-the-Loop) testing for AD (Autonomous Driving) stacks. The system has been integrated with Autoware and Apollo AD stacks for comprehensive testing. It can also be easily expanded to incorporate other comparable AD systems. Follow the instructions on our Autoware.auto after downloading the simulator binary to utilise it with Autoware.auto. The tutorial for using Autoware with the LGSVL Simulator is accessible on GitHub [21]. They discussed how Autoware enables running it with LG Silicon Valley Lab’s autonomous driving simulator. It is simplest to create and run a custom Docker image in order to run Autoware with the LGSVL simulator. Additionally, it is essential to clone LGSVL’s Autoware data repository. This repository contains the necessary point cloud maps and launch scripts that are required for running Autoware in the default San Francisco environment of the simulator. This process will need a Linux operating system. The ROSbridge suite, which offers JSON interfacing with ROS publishers/subscribers, is used by Autoware to communicate with the simulator.

2.6 ROS-Autoware-CarMaker Simulation Architecture for Autonomous Vehicles

This paper [22] provides an architectural framework of a different approach that combines Autoware.AI and IPG CarMaker to facilitate the testing of autonomous vehicles in a virtual environment shown in figure 2.5. This study focuses on the development of middleware that connects IPG-CarMaker and Autoware.AI. using libraries and features from IPG CarMaker’s Application Online (APO) interface and the Robot Operating System (ROS) in both Linux and Windows operating systems.

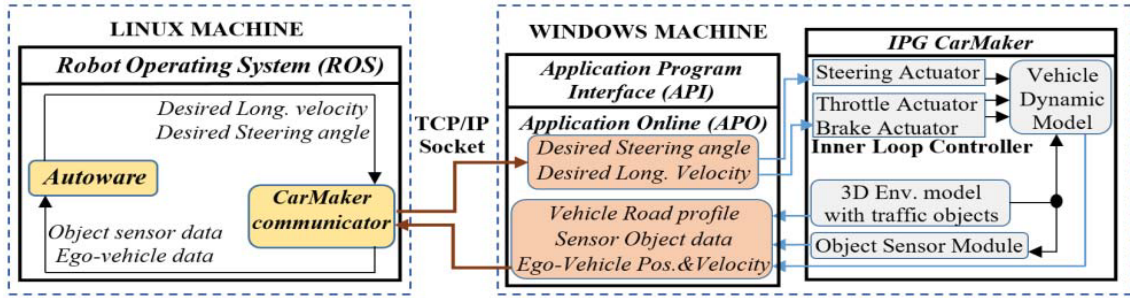


Figure 2.5: Simulation architecture using ROS, Autoware, and CarMaker [22]

The paper [22] also discusses the data exchange between Autoware.AI and IPG, specifically focusing on time synchronisation to enhance communication. This study presents a case study that evaluates the performance of Autoware.AI in two different manoeuvres using the IPG CarMaker virtual simulation platform. The integration platform offers several advantages. The platform integrates ROS, Autoware and CarMaker to offer a comprehensive simulation architecture for testing diverse aspects of autonomous vehicles. However, it is crucial to recognise specific constraints. Using two different operating systems, such as Linux and Windows, can lead to increased complexity in configuration and usage, particularly for individuals lacking familiarity with both systems. In summary, this research paper has provided valuable insights and a clear framework for creating an efficient interface between Autoware.AI and IPG CarMaker (Please note that this solution is not available for public use) [22].

2.7 Comparison of different simulators and AD stack software

Several simulation tools used in the creation and testing of autonomous cars are covered in the literature study, such as IPG, PreScan and CARLA, and it specifically emphasises CARLA as an open-source software that fulfils certain criteria. This thesis also discusses various open-source platforms for autonomous driving stacks and recommends Apollo and Autoware as the best options. It is emphasised that both Autoware and Apollo’s architectures are built on ROS and provide autonomous driving software stacks, which has contributed to the rising popularity of respective HD map formats shown in figure 2.6 [23].

The review highlights the significance of converting data to the OpenDRIVE format for utilisation in CARLA. The SUMO net convert tool can be used for this purpose. The paper also addresses the essential components needed for setting up the simulation environment, such as the necessary software and dependencies.

The paper [24], [25] outlines the methodology for generating a 3D scenario for simulating autonomous driving by utilising freely available GIS data. The steps required for CARLA setup



Figure 2.6: Overview of the Apollo architecture[23]

and establishing a bridge to Autware with the necessary commands are discussed. Once the HD map, the CARLA Autware bridge can be initiated. This allows for control over the simulation either through interaction with Autware or by activating the manual control functionality made available by the CARLA PythonAPI module via the ROS bridge figure 2.7.

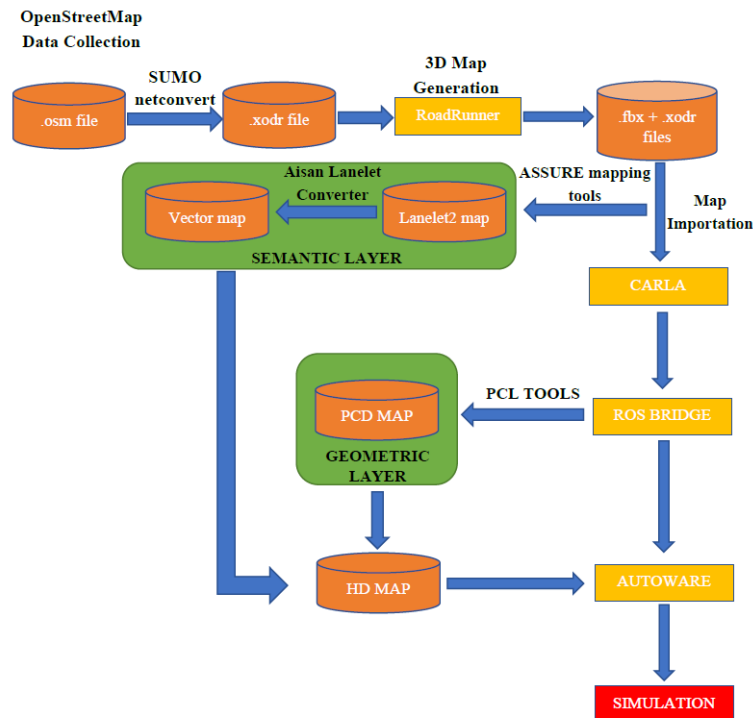


Figure 2.7: Workflow Flowchart for OpenStreetMap Data Collection [24,25]

This research provides a detailed overview of simulation software, open-source platforms, HD map creation, and the necessary simulation environment setup for autonomous driving frameworks. It emphasises the significance of precise HD maps, as well as the roles of Autoware and Apollo in HD map development. It also provides a practical methodology for generating a 3D scenario for autonomous driving simulation using freely available GIS data, it outlines the necessary steps for setting up CARLA and Bridge to Autoware. [24].

2.8 Development and Validation of ADAS Applications with the CARLA Simulator in ROS

This paper [26] highlights the fact that unforeseen obstacles and incidents are expected as the level of automated driving increases. To successfully handle these issues, modern vehicles growing complexity requires powerful computing hardware and software. Testing methods that have been used in practice are expensive, time-consuming and filled with risk. Virtual simulation technologies are therefore highly recommended for their application in car testing. These tools enable Original Equipment Manufacturers (OEMs) to establish closed sensor-actuator loops and perform computations. Simulated sensory input is employed for testing production software, and simulated action outcomes are generated in accordance with the program's instructions. Before the production of physical automobile prototypes Original Equipment Manufacturers (OEMs) have the opportunity to enhance the design of the vehicles [26]. Optimising early leads to reduced costs and reduced waiting periods. This study demonstrates the creation of basic C++ perceptual applications using ROS as a platform for validation and prototyping. The applications were then evaluated using SIL processes.

The CARLA simulator plays a crucial role in creating simulations by providing data, translating commands from the autonomous platform, and executing simulated manoeuvres. The integration of the CARLA simulator and Autoware enables the validation of application-specific scenarios. [27].

2.9 Overview of the Planning Support Tool for Autonomous Driving

The execution of automated driving depends heavily on planning. For its development, validation, and execution, planning requires a variety of software and tools. This paper [27] highlights a survey of these tools, which include middleware, simulators, data visualizations, benchmark datasets, open-source planning, communication, traffic rules and map representation. They have given an in-depth investigation of the relationships between state-of-the-art development and analysis. Numerous studies have evaluated planning algorithms, but there is a lack of comprehensive research on planning tools. The project aims to simplify the development of a software platform

that satisfies academic needs while facilitating the use of open-source methods. The author endeavours to develop a creative motion planner as an illustrative instance. Integrating the open-source Autoware and ROS Middleware as a software stack is a prudent decision due to their applicability to real-world vehicles. CARLA can be employed as a simulator to effectively utilise its graphics and sensor simulation capabilities in the development process. To enhance the realism of the simulation, researchers can employ the open-source SUMO programme for simulating large-scale traffic flows and the for-profit CarMaker tool for simulating advanced car dynamics [28], [29]. The open drive maps can be used as a source to create the Autoware, SUMO and CARLA map formats. The constructed method can be examined and compared to different strategies using standard road conditions. Co-simulation software and interfaces are useful for tool coupling. Consequently, the complex process of tool and interface creation can be eliminated, allowing more time to be dedicated to algorithm development [30].

Chapter 3

Research Methodology

3. Research Methodology and Implementation

The thesis involves a multi-step process. First, it focuses on the reconstruction, simulation, and analysis of the Aslan-CarMaker project. This step is crucial to understanding the Aslan message datatype and its integration with CarMaker through CMRosIF. Following that, a bridge platform is programmed between Autoware and CarMaker by manipulating the topics. The primary objective of this step is to establish inter-process communication between these two software applications, enabling the exchange of data [31].

In the context of inter-process communication, the Robot Operating System (ROS) utilises message passing via publisher and subscriber topics to enable smooth connections and cooperation among various modules in a distributed system. This mechanism facilitates efficient data transmission from a sender to multiple receivers, supporting the development of complex distribution systems with diverse functionalities and components. It also allows for reading messages on various topics. In this case, the message-passing inter-process communication method allows a sender to transfer data to one or more subscribers.

In the ROS ecosystem, nodes are independent processes that perform specific tasks and communicate with each other using a publish/subscribe mechanism. This mechanism involves a node publishing messages to a topic, which are then received and processed by another node, allowing for efficient inter-node communication. When a node possesses data to distribute, it assumes the role of a publisher and transmits messages to a specified topic. Topics serve as channels through which data flows within the ROS network shown in figure 3.1 [32]. Alternatively, any node interested in receiving that particular type of data can become a subscriber and read the messages from the topic.

ROS enables the construction of robotic applications in a flexible and robust manner through

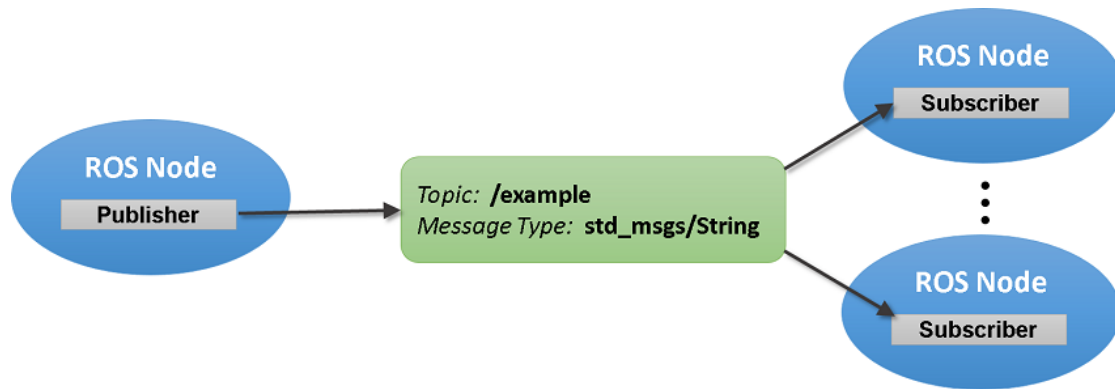


Figure 3.1: ROS with single publisher and multiple subscribers [32]

the implementation of the publish/subscribe pattern. Complex tasks can be decomposed into specialised nodes, each dedicated to a specific function, these nodes can communicate efficiently through topics, enabling information exchange and collaborative work towards the overarching goal.

Overall, this methodology entails a comprehensive approach to establishing inter-process communication between CarMaker and Autoware, with a focus on developing a bridge platform that allows data exchange between these two software.

3.1 Required and Supported software versions

The software versions needed and supported by this research are listed in this section. Functionality with these software versions is critical to ensuring the project’s seamless integration and optimal performance.

- Ubuntu 18.04 LTS (Linux operating system)
- ROS_1 Melodic desktop-full [33] ROS build farm.
- Catkin Command Line Tools (Original build system) [34]
- CarMaker 10.1
- Autoware 1.14.0

3.1.1 Installation

Several software installations are required to begin the project:

1. **ROS_1 Melodic:** It is essential to install ROS_1 Melodic because Autoware is only compatible with ROS_1. The installation procedure can be found at Link1 [35].
Verify the complete installation of ROS by running several commands in different terminals

for the ROScore, talker, and listener attributes, the display of hello world on the terminal screen confirms the successful installation of ROS.

2. **Aslan-CarMaker Project:** Clone the whole Aslan-CarMaker project from its GitHub repository: Link2 [36].
3. **Autoware 1.14.0:** Install Autoware 1.14.0 by referring to the instructions at Link3 [37].
4. **CarMaker 10.1:** The comprehensive CarMaker 10.1 package can be accessed and installed via the IPG Client Area, but this requires a licence. To access the client area, one must be logged in to the system with a valid account. Choose the CarMaker installation version that best meets your needs and download CarMaker for Linux from the "Office" page.

3.2 Reconstruction and Evaluation of the Current Aslan-CarMaker Project

The integration process of Aslan and CarMaker involves the combination of two software systems, namely ASLAN (an open-source autonomous driving Algorithm stack) and CarMaker (a simulation tool), to ensure their effective co-simulation. This integration allows researchers and developers to simulate and evaluate autonomous driving algorithms in a controlled environment prior to their implementation in real-world vehicles. Below is a concise overview of the essential stages involved in the integration process:

The integration process begins with the installation of CarMaker, a software platform used for creating and simulating vehicle models and scenarios. To proceed, users should download CarMaker version 10.1 from the IPG Client Area and follow the corresponding installation instructions. The environmental variables are set to indicate the location of the CarMaker installation directory.

Secondly, the procedure of setting up ROS (Robot Operating System) involves establishing a communication bridge between Aslan and CarMaker. This bridge facilitates the smooth exchange of data between the two software platforms, with ROS serving as the intermediary for communication. In order to ensure successful system functionality, it is essential to install ROS Melodic, along with the requisite components, including Catkin Command Line Tools. This step is crucial for establishing effective communication between Aslan and CarMaker.

Thirdly, the Aslan-CarMaker project repository is cloned to the local machine from GitHub, It serves as a framework for the integration, containing code and resources required for the collaboration between Aslan and CarMaker.

The integration project gets built using the provided scripts. This step entails establishing the essential components, workspaces and configurations required to facilitate seamless integration

between Aslan and CarMaker. In this phase, the custom executable for CarMaker is generated, including the necessary ROS communication libraries.

Fifthly, the configuration of the Aslan-CarMaker interaction: The Aslan GUI must be properly configured for optimal interaction between CarMaker and Aslan shown in figure 3.2. The Aslan-CarMaker integration is configured via a combination of scripts and GUI parameters. The "CMStart.sh" script is used to run CarMaker along with the essential ROS components, ensuring a synchronised environment. The Aslan GUI's Control tab allows for precise adjustment of Aslan's interaction with CarMaker, including settings for speed input and mode.

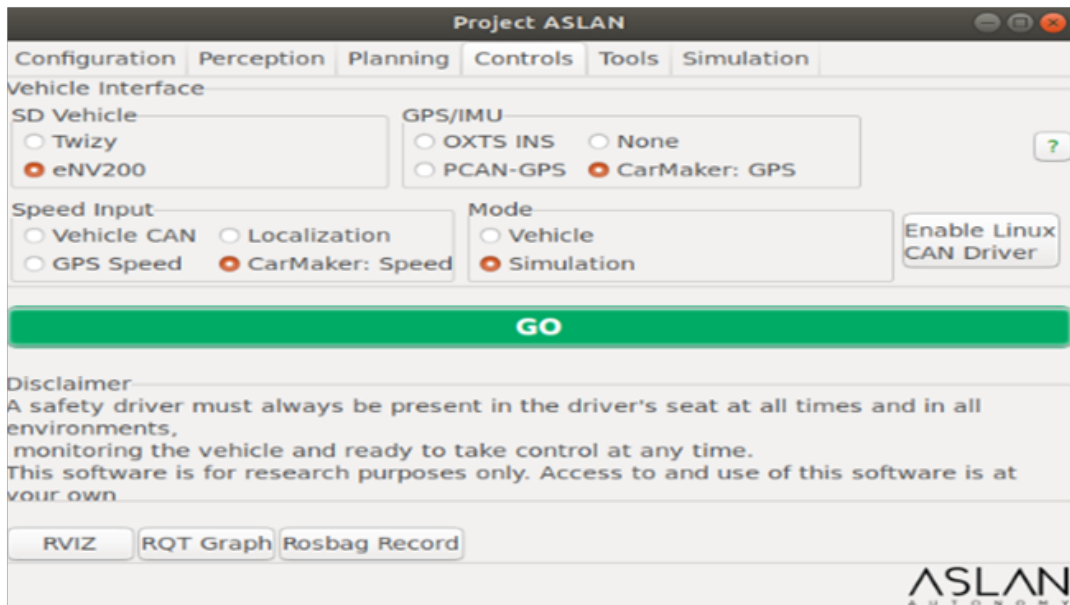


Figure 3.2: Aslan GUI's Controls tab

Finally, the integration begins after it has been configured by starting both Aslan and CarMaker. Aslan commands are transmitted to CarMaker, which controls the operations of the vehicle model in the simulated environment. The responses and environment data of CarMaker are relayed back to Aslan resulting in a co-simulation simulation

3.3 Gazebo co-simulation with Aslan AD stack

The Aslan project also incorporates the use of Aslan-Gazebo Co-Simulation to perform autonomous driving manoeuvres for vehicles like Twizy and eNV200 models in the Gazebo simulation environment. The Gazebo simulation comprises various components, including buildings, flyovers, terrains and other elements as shown in figure 3.3. The main focus of this simulation is to run the vehicle in low-speed urban traffic conditions, with the aim of applying the simulated behaviours to real-life road situations.

The Gazebo simulation framework is freely available and open source. This simulation framework contains a variety of simulated environments, with the considerably preferred option

being the "Baylands" Gazebo world. The Gazebo world can be easily accessed via the Aslan project's graphical user interface (GUI) located in the simulation tab. To begin the Aslan-Gazebo Co-simulation for the Twizy model, ensure correct configuration settings and click the "SD twizy model" button in the Aslan GUI.

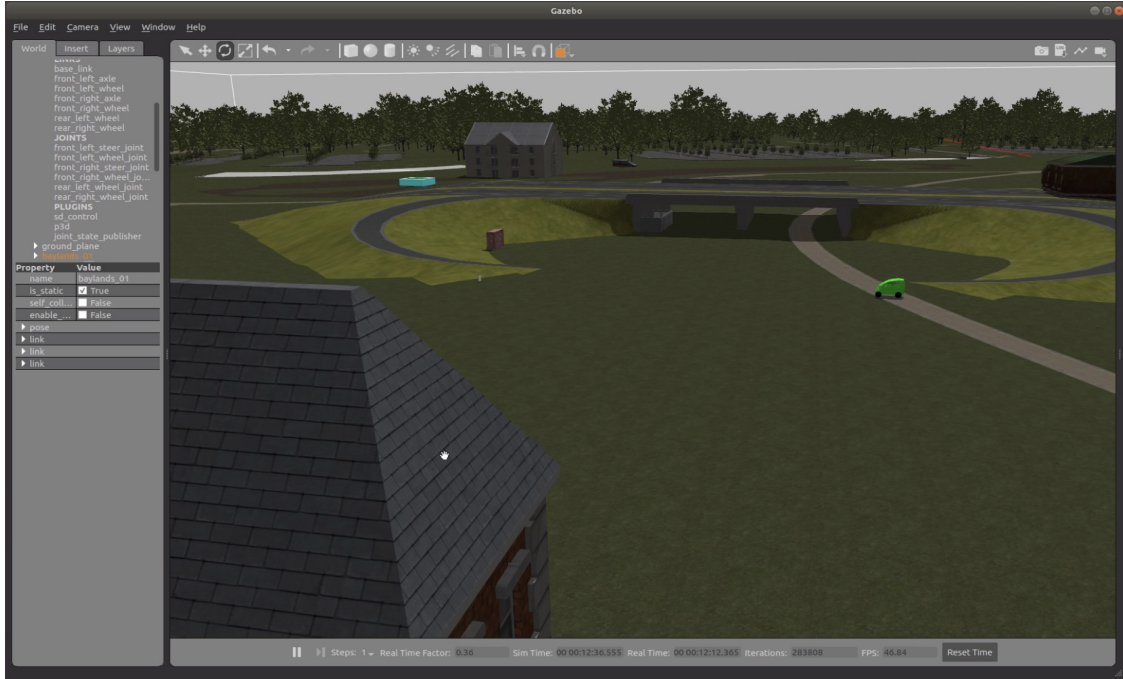


Figure 3.3: Gazebo Baylands park simulation environment with SD Twizy model

The integration of Aslan and Gazebo involves configuring the system by following these steps as shown in figure 3.4: selecting the "Twizy" from the SD vehicle in the control tab of the Aslan interface, choosing "None" for the GPS/IMU settings, selecting "GPS Speed" as the speed input and setting the mode to "simulation". The configuration of the Aslan Interface for Gazebo is almost the same as that of the Aslan-carmaker process, achieved by selecting the "Default Park" setting in the simulation environment. To begin the Gazebo simulation with the Renault Twizy, the user should choose the "SD Twizy model" option in the Gazebo interface. This integration enhances the testing process by enabling the evaluation of the autonomous driving of Twizy's behaviour in a simulated environment.

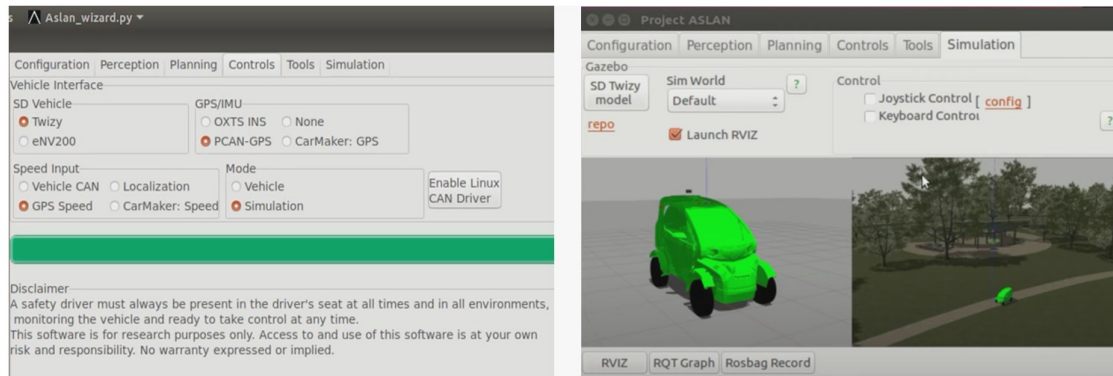


Figure 3.4: 'Controls' and 'Simulation' Tabs in Aslan GUI

During the launch of a Gazebo simulation, the car can be manually controlled using a keyboard or joystick within the simulation environment. At the same time, sensors are used to record the trees, paths and buildings of the "Baylandspark". The recorded data is processed using NDT mapping to create a point cloud map. The same recorded simulation data is used to extract a waypoint path using the Waypoint Saver, this results in a CSV file containing waypoint coordinates. These files facilitate the autonomous replication of manual driving in the simulated environment. The process is initiated by launching RViz to evaluate the functionality of cameras and LIDAR sensors, which offer real-time observations of the simulation environment. Upon selecting the "Park Default World" and initiating the simulation, Gazebo opens and poses the environment, which is identified by an array of red LIDAR points as shown in figure 3.5. Modifying the point size improves visibility by allowing for the detection of reflections originating from the trees around it. Subsequently, the positioning of the vehicle for testing is established. The lidar simulation driver is activated when the Gazebo's Twizy model is launched, as confirmed by the observation of LIDAR points in RViz. Configuring the position of the LIDAR sensor in relation to the base link using the "Face Linked to Localizer" ensures precise collection of data. A Ros Bag is subsequently recorded for the purpose of analysis.

In summary, the Aslan-CarMaker integration procedure includes preparing the software, establishing communication via ROS, building the integration project, configuring interaction settings, and lastly performing simulations to evaluate autonomous algorithms. This collaborative effort provides a controlled and reproducible environment for the development and testing of autonomous driving technology. Several difficulties were detected during the project evaluation, which are discussed in the Results and Discussion chapter. The examination of this Aslan-CarMaker project provided beneficial insights as shown in figure 3.6, for moving ahead to the Autoware-CarMaker project.

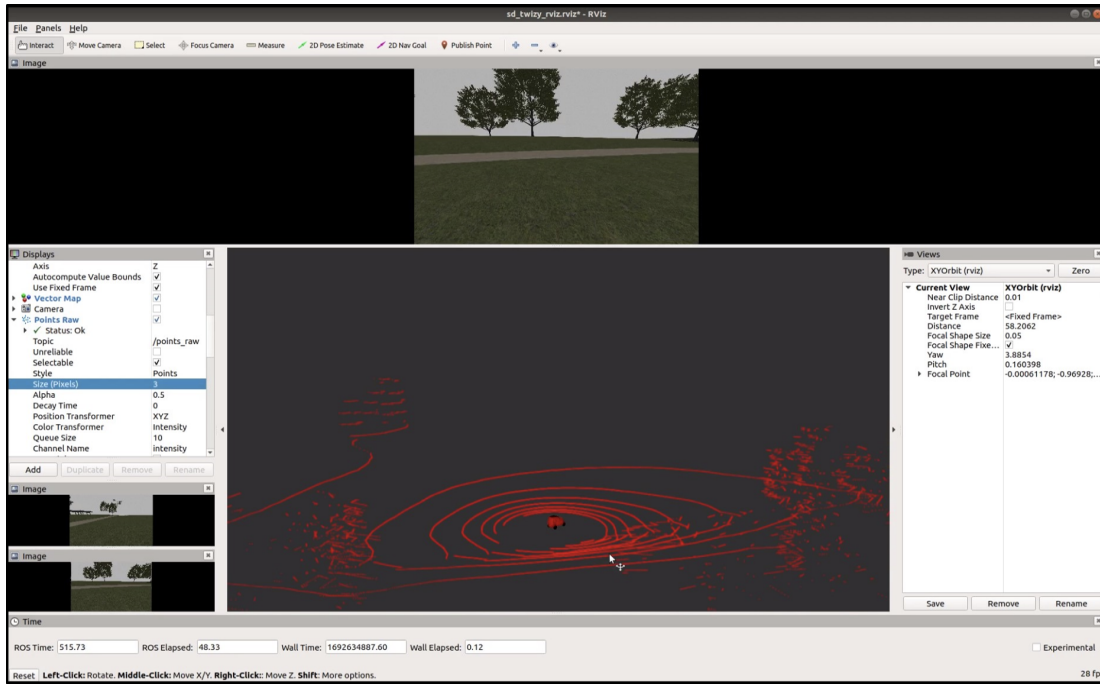


Figure 3.5: Generated PCD from Aslan-Gazebo simulation

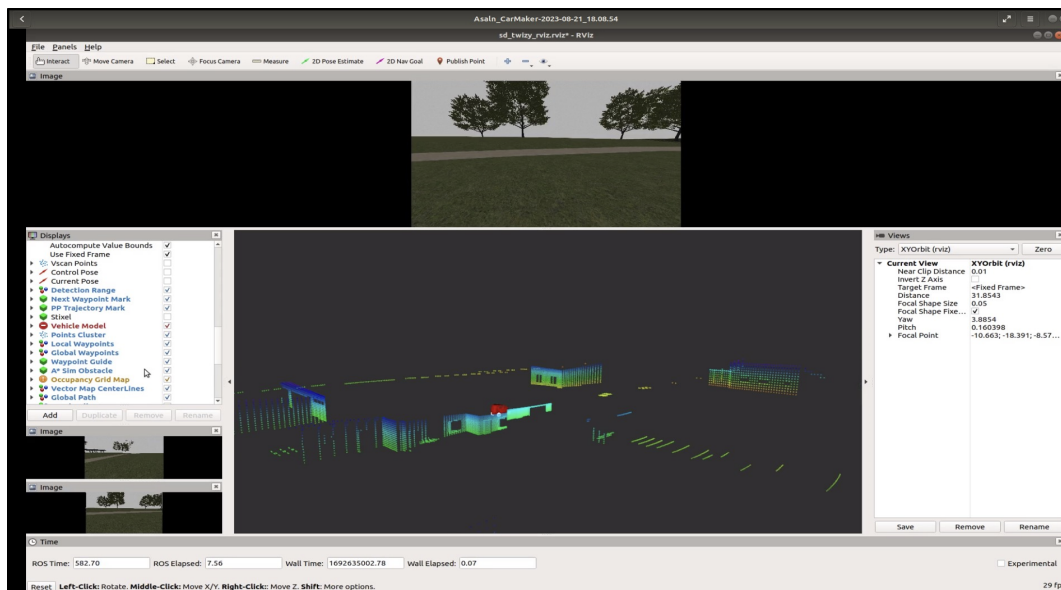


Figure 3.6: Generated 3D point clouds obtained from Aslan-CarMaker open world simulation.

3.4 Creating ROS Bridge (main.cpp bridge)

The ROS bridge utilises C++ programming to convert Autoware.AI messages into Aslan message type, and then It publishes to CMRosIF of Aslan-CarMaker’s project to integrate Autoware and CarMaker. This design as shown in figure 3.7, is essential as the Aslan-CarMaker’s CMRosIF only supports the Aslan message datatype. The bridge is capable of subscribing Autoware messages via the `/vehicle_cmd` topic and publishing Aslan message data type via the `/sd_control` topic to CMRosIF, thereby ensuring compatibility between the two systems to simulate the Car

in CarMaker. Aslan and Autoware have similar data structures for sensor measurements, control commands, localization information, and planning. The architectural design of the bridge is illustrated in the diagram provided. To create the bridge between CarMaker and Autoware using

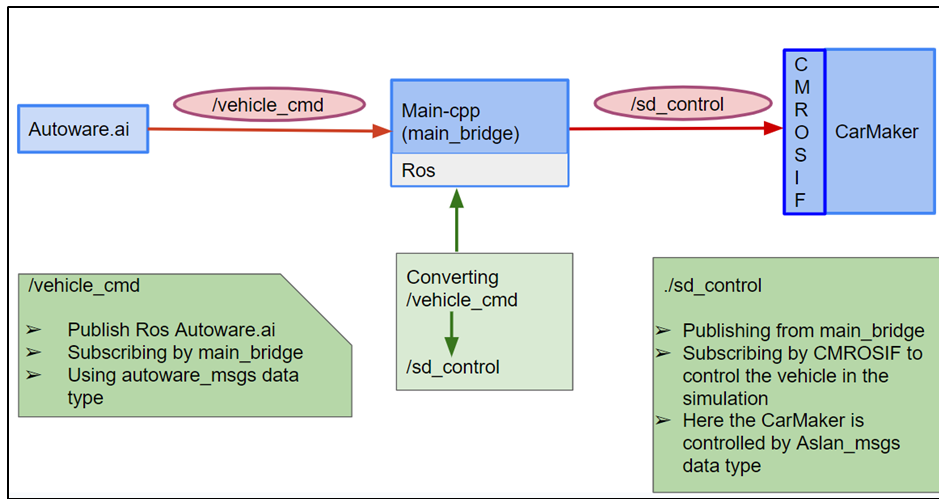


Figure 3.7: Integrated Autoware.AI-CarMaker Architecture with Topics and Messages

ROS, it is necessary to build their custom message architecture in the ROS workspace. By building these messages in the ROS workspace, these messages are compiled into the required binary format. This allows other nodes and packages in the system to properly send, receive and interpret effective communication and integration between them. The procedure to create the bridge is as follows:

1. The 'aslan_msgs' architecture is included as a package in the 'src' folder of the ROS workspace as shown in figure 3.8. The 'aslan_msgs' can be downloaded from Link4 [38].
2. The 'autoware_msgs' architecture is included as a package in the 'src' folder of the ROS workspace as shown in figure 3.8. The 'autoware_msgs' can be downloaded from Link5 [39].

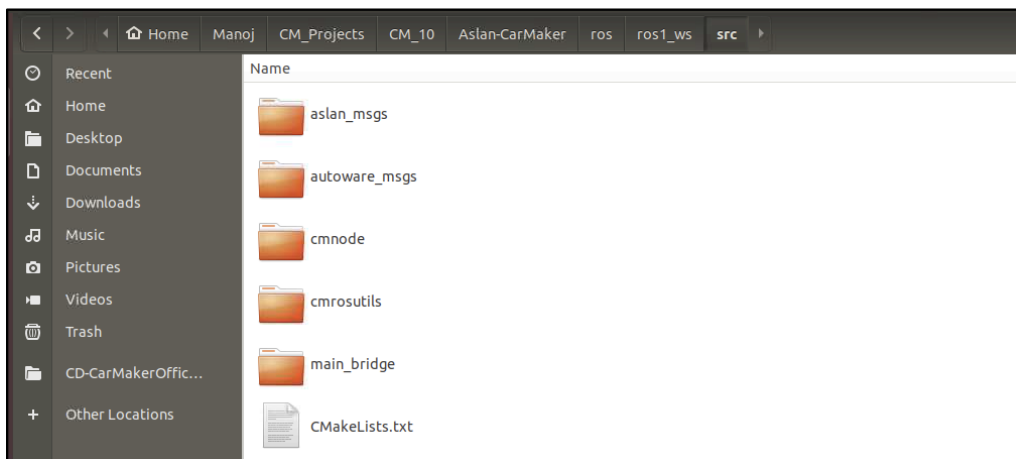


Figure 3.8: Directory illustrating Aslan and Autoware Message Architecture Location

- When integrating 'Autoware_msgs', it is recommended to exclude specific custom messages to reduce the computational power, retaining only include essential ones for vehicle control, namely 'AccelCmd.msg', 'BrakeCmd.msg', 'Gear.msg', 'SteerCmd.msg', 'LampCmd.msg', 'ControlCommand.msg' and 'VehicleCmd.msg' as shown in figure 3.9. In order to control the vehicle, the following set of commands will be exclusively utilised. The remaining msg files within the folder, as well as their corresponding names in the 'CMakeList.txt' file of the 'Autoware_msgs' package can be deleted.

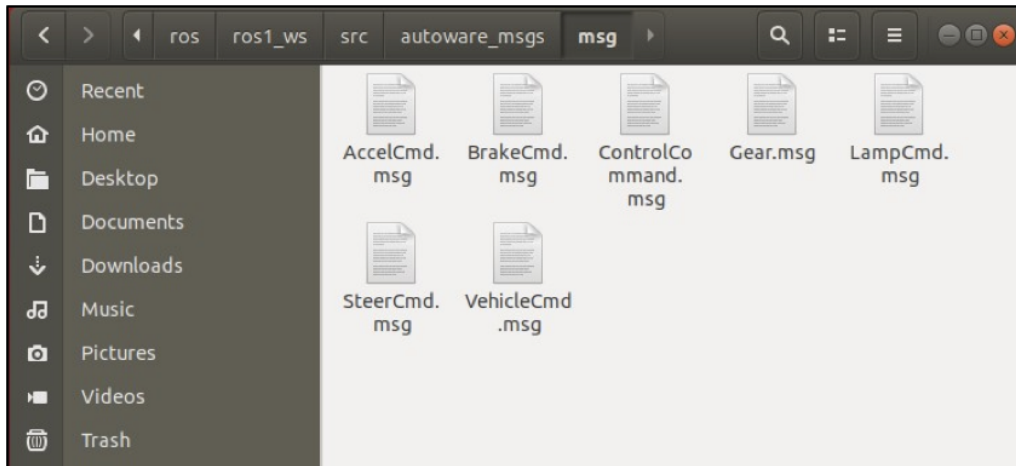


Figure 3.9: Directory illustrating the important essential messages present inside Autoware_msgs

- Subsequently, navigate to the 'rosws' directory and execute the 'catkin make' command, wait until the compilation to finished.
- Then, source the workspace as 'devel/setup.bash'. Now run the command 'ROSMsg list', used to determine the availability of newly established messages.

3.4.1 About the code in main.cpp

The bridge is programmed in C++ to provide better performance and to make this project simpler to manage as shown in figure 3.10. The following steps need to be taken to create the ROS bridge for enhanced data transfer between IPG CarMaker and Autoware.AI:

- Create a ROS package 'main_bridge' in the 'src' folder of the ROS workspace.
- Create a file 'main.cpp' inside the 'src' folder of the 'main_bridge' package. Then, add this file as an executable in the 'CMakeLists.txt' file of the 'main_bridge' package.
- The algorithm development process relies on message analysis, topic identification, and the establishment of logical connections. The program screenshots and execution details are outlined below pictures for comprehensive understanding.

```

main.cpp
> Users > manoj > Desktop > Thesis_1 > Bridge > main.cpp
1  #include <ros/ros.h>
2  #include "aslan_msgs/SDControl.h"
3  #include "autoware_msgs/VehicleCmd.h"
4  class AslanBridge {
5  private:
6  aslan_msgs::SDControl sd_msg;
7  ros::Publisher sd_pub;
8  ros::Subscriber vehicle_cmd;
9  public:
10 AslanBridge(ros::NodeHandle *nh) {
11     sd_pub = nh->advertise<aslan_msgs::SDControl>("/sd_control", 1000);
12     vehicle_cmd = nh->subscribe("/vehicle_cmd", 1000, &AslanBridge::vehicleCmdCallback, this);
13 }
14 void vehicleCmdCallback(const autoware_msgs::VehicleCmd::ConstPtr& msg)
15 {
16     sd_msg.header = msg->header;
17     sd_msg.steer = msg->steer_cmd.steer;
18     if(msg->brake_cmd.brake > 0){
19         sd_msg.torque = -20.0*msg->brake_cmd.brake;
20     }
21     else{
22         sd_msg.torque = 20.0*msg->accel_cmd.accel;
23     }
24     sd_pub.publish(sd_msg);
25 }
26 };
27 int main (int argc, char **argv)
28 {
29     ros::init(argc, argv, "aslan_bridge");
30     ros::NodeHandle nh;
31     AslanBridge nc = AslanBridge(&nh);
32     ros::spin();
33 }

```

Figure 3.10: Complete source code inside main.cpp bridge

4. Execution: The execution process consists of three steps: building the file using the 'catkin_make' command, sourcing it by running 'devel/setup.bash', and finally executing it with the command 'ROSRUN main_bridge main'.

3.5 Implementation of Autoware-CarMaker Communication

Follow the below steps for effective communication between Autoware and IPG CarMaker:

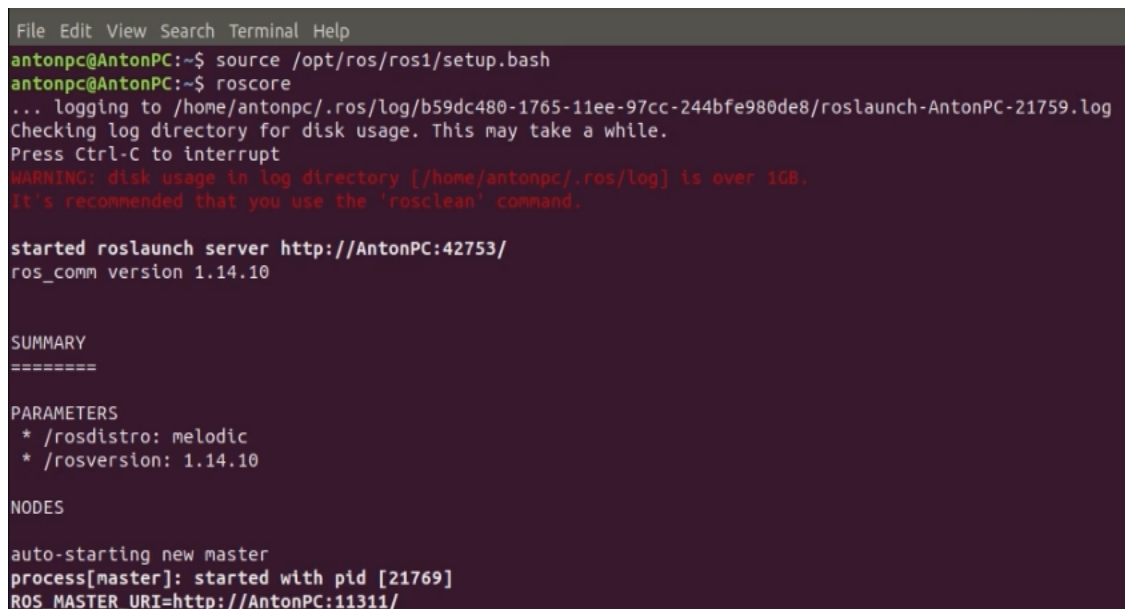
3.5.1 Initialise the ROS Environment and roscore

Launch Terminal_1: Set up the ROS environment and initialize the core components necessary for ROS communication using the following highlighted commands as shown in figure 3.11:

```
source /opt/ros/ros1/setup.bash
roscore
```

Figure 3.11: Commands to initiate 'roscore'

The commands ensure the proper sourcing of the ROS environment and the initiation of the ROS master node (roscore). The ROS master node acts as a central hub for facilitating communication and message exchange among various ROS nodes. Executing these commands establishes the necessary infrastructure for communication in ROS, enabling the launch and interaction of other ROS nodes with the roscore as shown in figure 3.12.



```
File Edit View Search Terminal Help
antonpc@AntonPC:~$ source /opt/ros/ros1/setup.bash
antonpc@AntonPC:~$ roscore
... logging to /home/antonpc/.ros/log/b59dc480-1765-11ee-97cc-244bfe980de8/roslaunch-AntonPC-21759.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
WARNING: disk usage in log directory [/home/antonpc/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://AntonPC:42753/
ros_comm version 1.14.10

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.10

NODES

auto-starting new master
process[master]: started with pid [21769]
ROS_MASTER_URI=http://AntonPC:11311/
```

Figure 3.12: Initiation of the ROS master node

3.5.2 Initializing CarMaker and Selecting the Project

Launch Terminal_2: Execute the custom script to initialise the IPG-CarMaker GUI from the Aslan-CarMaker directory as shown in figure 3.13.

The 'CMStart.sh' script is responsible for setting up the CarMaker environment effectively,

```
cd ~/CM_Projects/CM_10
cd Aslan-CarMaker
./CMStart.sh
```

Figure 3.13: Custom script to initialize the IPG-CarMaker

which includes sourcing the necessary ROS workspace files. Executing the script enables the CarMaker GUI to start with additional dedicated ROS navigation menu options as shown in figure 3.14. This step is crucial for the setting up of CarMaker for integration with ROS.

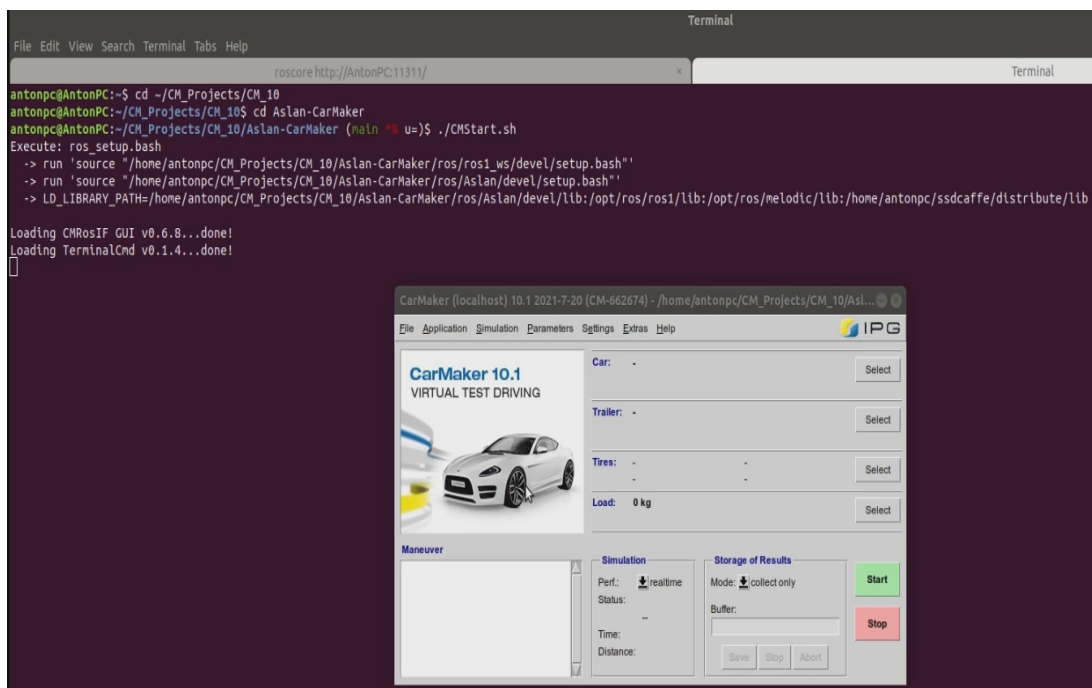


Figure 3.14: CarMaker GUI

The first time the user wants to start CarMaker in the Aslan-CarMaker setup, they need to load the project manually. This can be achieved through the following steps as shown in figure 3.15: Firstly, open the CarMaker Main GUI. Secondly, access the "File" menu and select "Project Folder" from the dropdown. Thirdly, the user has the option to either generate a project by specifying roadways, vehicles, and passengers or choose from an existing project. Finally, choose for "Select..." option and specify the folder containing the local cloned copy, Confirm the selection by clicking "OK."

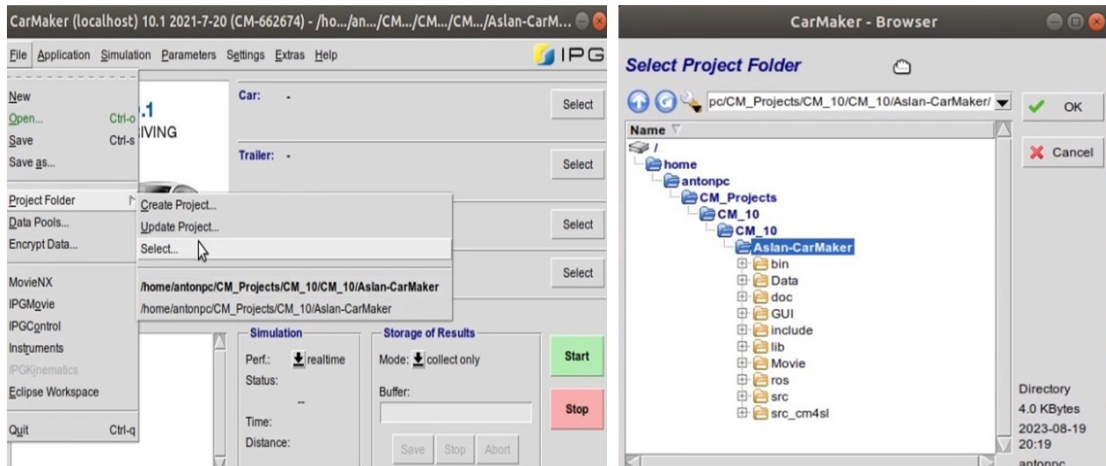


Figure 3.15: Manual Selection of the Project Folder

3.5.3 Configuring CarMaker Executable

Navigate to the CM Main GUI, then go to Application and then choose Configuration/Status. In the Command (executable) field, choose bin/CarMaker.linux64 as shown in figure 3.16.

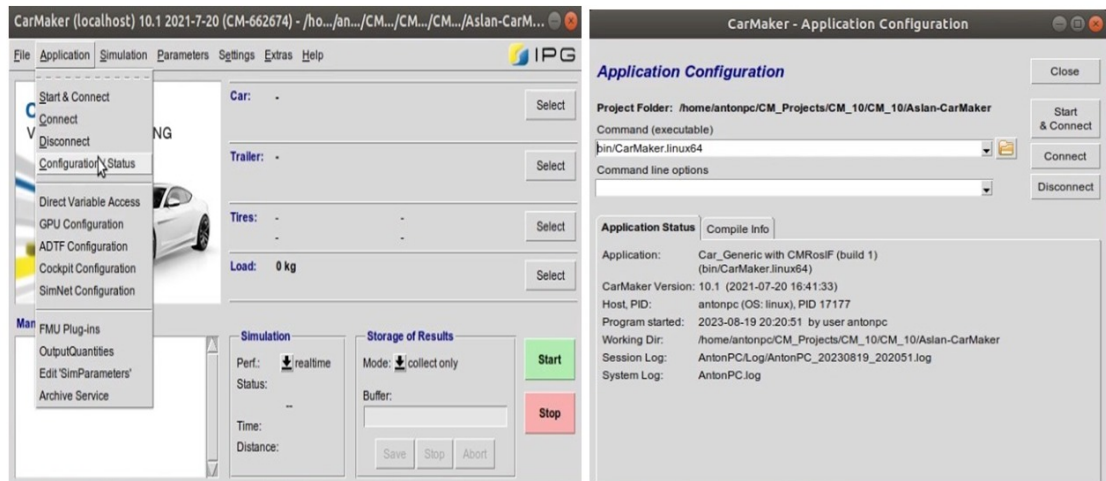


Figure 3.16: Application configuration of CarMaker executable

The necessary libraries for facilitating communication between CarMaker and ROS are incorporated by choosing the custom executable designed exclusively for CarMaker during the build process. This step is crucial to ensuring successful communication between CarMaker and ROS. The appropriate custom executable must be chosen.

3.5.4 Start CMRosIF

To establish communication between CarMaker and ROS, the following steps should be followed as shown in figure 3.17: In CM Main GUI navigate to Extras, CMRosIF and click Launch and Start Application Menu, by option in the CMRosIF menu, the CMRosIF node will be initialised, facilitating the necessary communication setup between CarMaker and ROS. The inclusion of the

CMRosIF option in the Aslan-CarMaker project is a key factor for its utilisation, as it offers essential functionality that is absent in the standard CarMaker GUI. CMRosIF serves as a bridge, enabling two-way communication between CarMaker and ROS. With the successful launch and implementation of CMRosIF, CarMaker is now able to seamlessly interact with ROS nodes and exchange messages. The integration of CarMaker and Autoware offers wide opportunities for testing and development, especially when combining the functionalities of CarMaker and Autoware. The successful initialization of CMRosIF represents a significant milestone in the implementation of communication between Autoware and CarMaker.

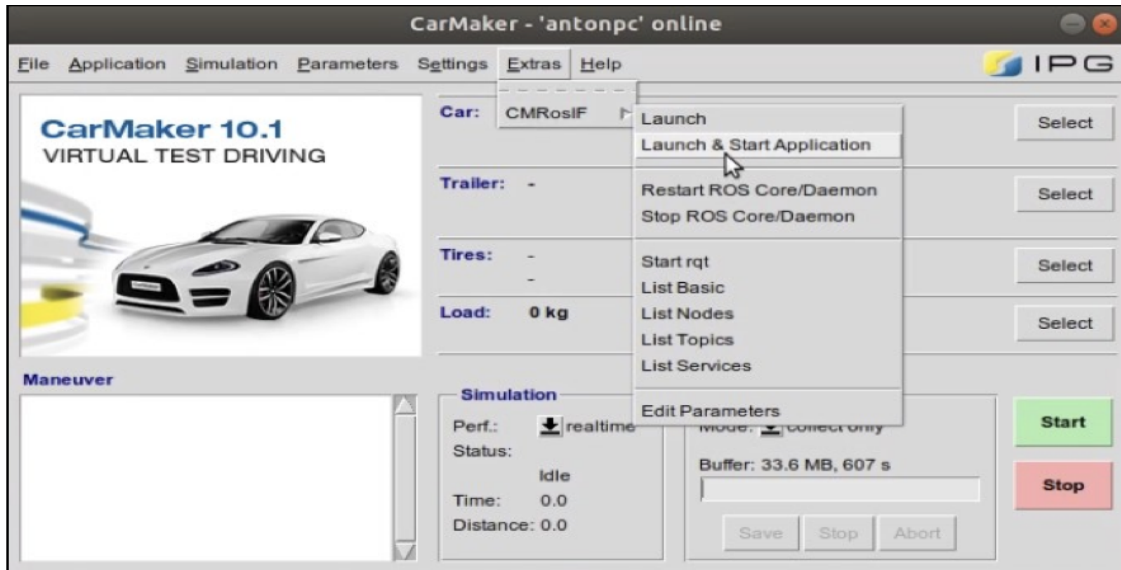


Figure 3.17: CarMaker GUI – CMRosIF (Launch and Start Application)

3.5.5 Initializing CarMaker Keyboard Control

Launch Terminal _4: The CarMaker Keyboard control script can be initialised to enable manual control of the CarMaker simulation through keyboard inputs. To perform this task, execute the specified command as shown in figure 3.18:

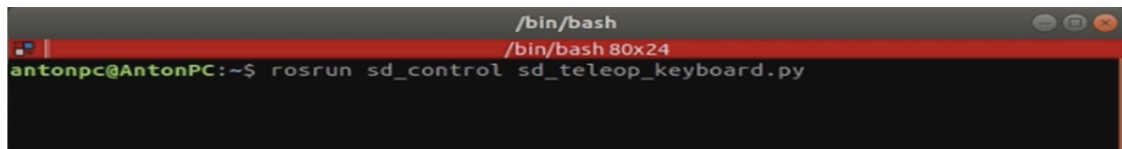
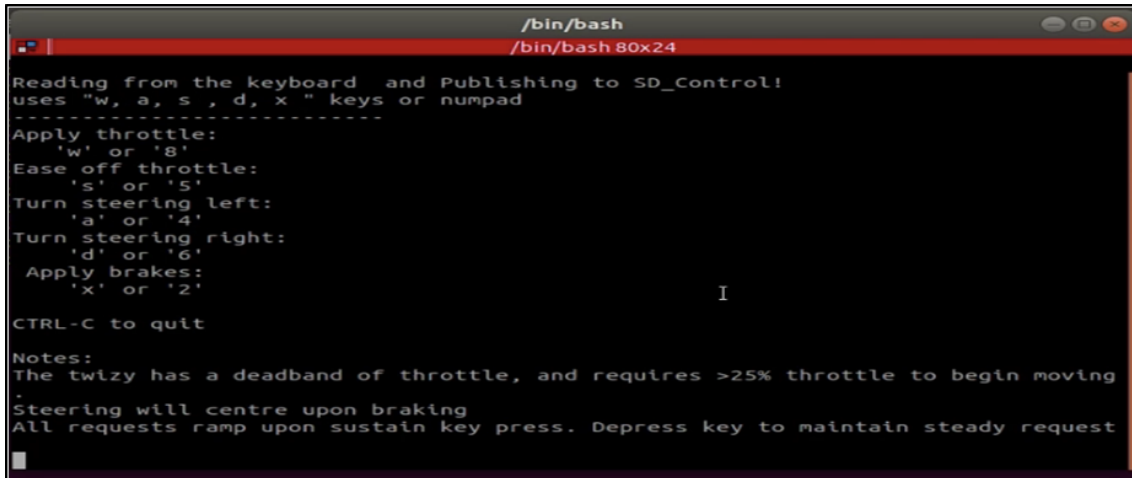


Figure 3.18: Command to run keyboard control python file

These commands initiate the CarMaker Keyboard control script is a Python script, that allows users to interact with the CarMaker simulation and manually control the vehicle’s manoeuvres using keyboard inputs. After initiating the CarMaker Keyboard control script, the user is prompted to provide input in order to control the simulated vehicle. The following inputs can be used to manoeuvre the vehicle manually during the simulation: To accelerate the vehicle’s speed,

to gradually reduce speed, press 's' or '5'; for leftward steering, enter the 'a' or '4' key; for rightward steering, enter the 'd' or '6' key; to apply brakes and decelerate or stop the vehicle, use the 'x' or '2' key. To terminate the CarMaker Keyboard control script and exit the simulation, press Ctrl+C.



```
/bin/bash
/bin/bash 80x24
Reading from the keyboard and Publishing to SD_Control!
uses "w, a, s , d, x " keys or numpad
-----
Apply throttle:
  'w' or '8'
Ease off throttle:
  's' or '5'
Turn steering left:
  'a' or '4'
Turn steering right:
  'd' or '6'
Apply brakes:
  'x' or '2'
I
CTRL-C to quit
Notes:
The twizy has a deadband of throttle, and requires >25% throttle to begin moving
Steering will centre upon braking
All requests ramp upon sustain key press. Depress key to maintain steady request
```

Figure 3.19: Keyboard Interface waiting for vehicle command inputs

The CarMaker Keyboard control script as shown in figure 3.19, in conjunction with the Autoware-CarMaker communication, facilitates comprehensive testing and refinement of autonomous driving functionality within a simulated environment.

3.5.6 Configuring and loading the open world test run in CarMaker GUI

Loading the Open World test run initialises the simulation environment in preparation for testing. In the CarMaker graphical user interface as shown in figure 3.20, navigate to the "File", "Open" options and select "Open World". Confirm your selection by clicking "OK".

The CarMaker open-world environment includes buildings, lanes, grass fields, and various features, providing a realistic simulation for testing autonomous driving capabilities. This feature enables the incorporation of pedestrians and animals, thereby generating dynamic scenarios that replicate real-world traffic conditions. By configuring and populating the environment with these elements, one can conduct comprehensive evaluations of the performance of autonomous vehicles in complex and challenging scenarios. Testing includes various scenarios such as intersections, traffic congestion, pedestrian crossings, and other relevant factors. These scenarios facilitate the evaluation of the vehicle's decision-making and control algorithms.

Prioritizing safety, data collected from simulations help refine algorithms and ensure efficiency and safety before real-world road testing, contributing to the successful deployment of autonomous vehicles.

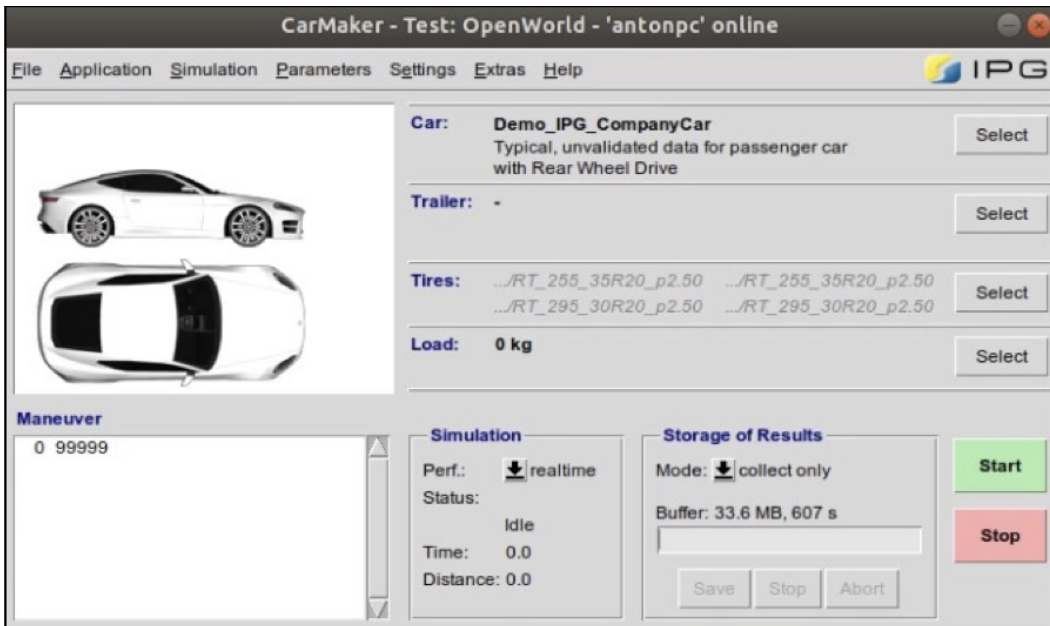


Figure 3.20: Loading procedure of the open world test run in CarMaker GUI

3.5.7 Testing the working of Keyboard control script with CarMaker

To verify the functionality of the `sd_teleop_keyboard.py` script with IPG Movie and visualise the simulation, navigate to the "File" menu in CarMaker GUI and select "IPG Movie" and observe the vehicle's response to the keyboard commands for manual control, as shown in figure 3.21.

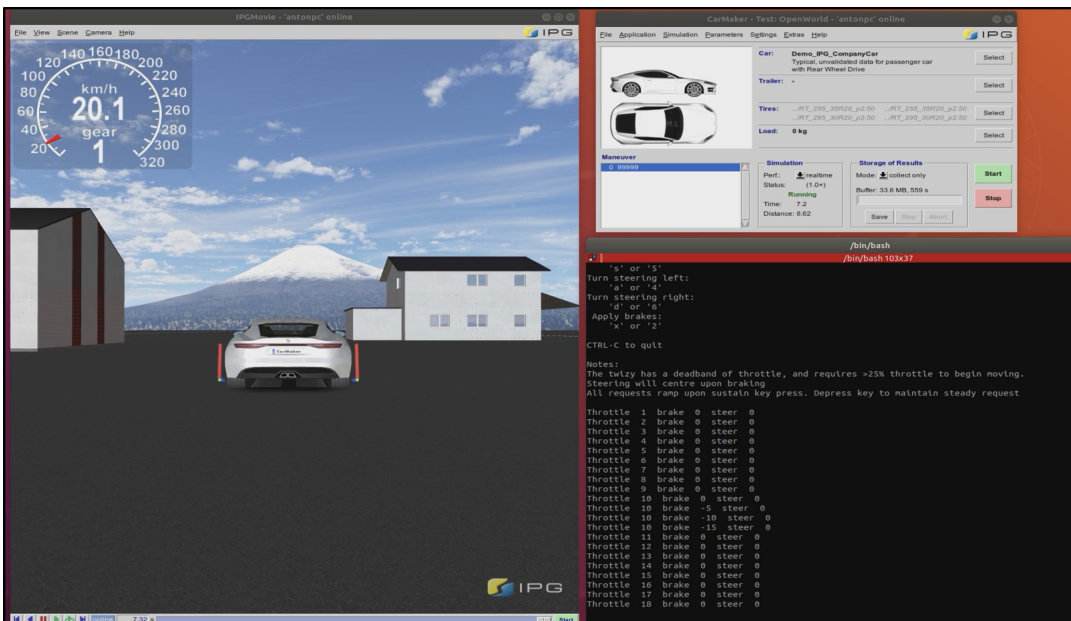


Figure 3.21: Verifying the keyboard control functionality with IPG Movie

3.6 Information about ROS topics before starting Autoware and bridge

The following steps need to be taken to analyze the available messages transferring before starting Autoware.AI and ROS bridge:

Open Terminal_5:

1. Visualize the available ROS topics using the command - "**rostopic list**"
2. Visualize the information about `/sd_control`, where `/sd_control` topic is responsible to send and receive messages from CMRosIF. This can be visualized using the command - "**rostopic info /sd_control**".

The output of command - `rostopic info /sd_control` gives the following information:

```
type: aslan_msgs/SDControl
```

```
Publishers:
```

```
* /teleop_twist_keyboard
```

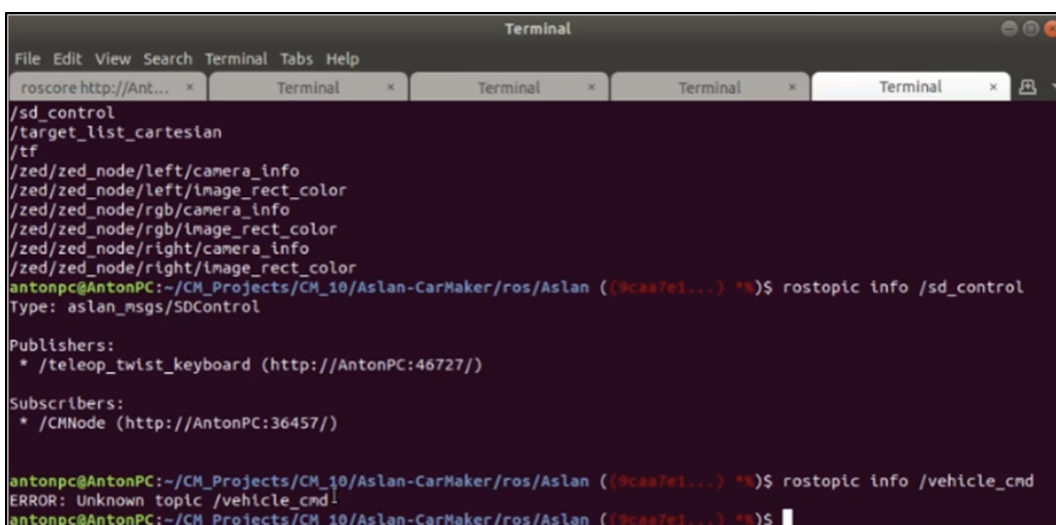
```
Subscribers:
```

```
* /CMNode
```

Here the messages are `aslan_msgs` which are going through `/sd_control` topic.

3. The visualisation of information related to the `/vehicle_cmd` topic, which is generally responsible for sending and receiving Autoware messages through it. This can be visualized using the command - "**rostopic info /vehicle_cmd**".

However, the output of the command `rostopic info /vehicle_cmd` now returns the error: no info found and cannot find the `/vehicle_cmd` in the rostopic list because Autoware has not yet been initialised as shown in figure 3.22.



```
Terminal
File Edit View Search Terminal Tabs Help
roscore http://Ant... x Terminal x Terminal x Terminal x Terminal x
/sd_control
/target_list_cartesian
/transform
/zf
/zf/zf_node/left/camera_info
/zf/zf_node/left/image_rect_color
/zf/zf_node/rgb/camera_info
/zf/zf_node/rgb/image_rect_color
/zf/zf_node/right/camera_info
/zf/zf_node/right/image_rect_color
antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/Aslan ((9caa7e1...))$ rostopic info /sd_control
Type: aslan_msgs/SDControl

Publishers:
* /teleop_twist_keyboard (http://AntonPC:46727/)

Subscribers:
* /CMNode (http://AntonPC:36457/)

antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/Aslan ((9caa7e1...))$ rostopic info /vehicle_cmd
ERROR: Unknown topic /vehicle_cmd!
antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/Aslan ((9caa7e1...))$
```

Figure 3.22: Information of `/sd_control` and `/vehicle_cmd` topics before initializing Autoware and ROS bridge

3.7 Launch Autoware User Interface

The Autoware.AI user interface is called the "Runtime Manager," which grants programmers access to necessary operations for manual and autopilot vehicle navigation. These operations include important functions such as localization, mapping, object detection, sensing and mission planning. Additionally, the system includes Rviz, which can be accessed via the Runtime Manager located in the lower right corner. Rviz enhances the visualisation of these functionalities. It effectively integrates and presents data, enhancing understanding of various processes such as localization on HD maps, object detection, path planning and path following. Autoware.AI heavily depends on the Robot Operating System (ROS), which is limited to Unix-based platforms. figure 3.23 depicts the overall structure of the Autoware GUI.

Open Terminal _6: To initiate the Autoware.AI runtime manager in the ROS melodic system and utilise its functionalities, execute the command:
"roslaunch runtime_manager runtime_manager.launch".

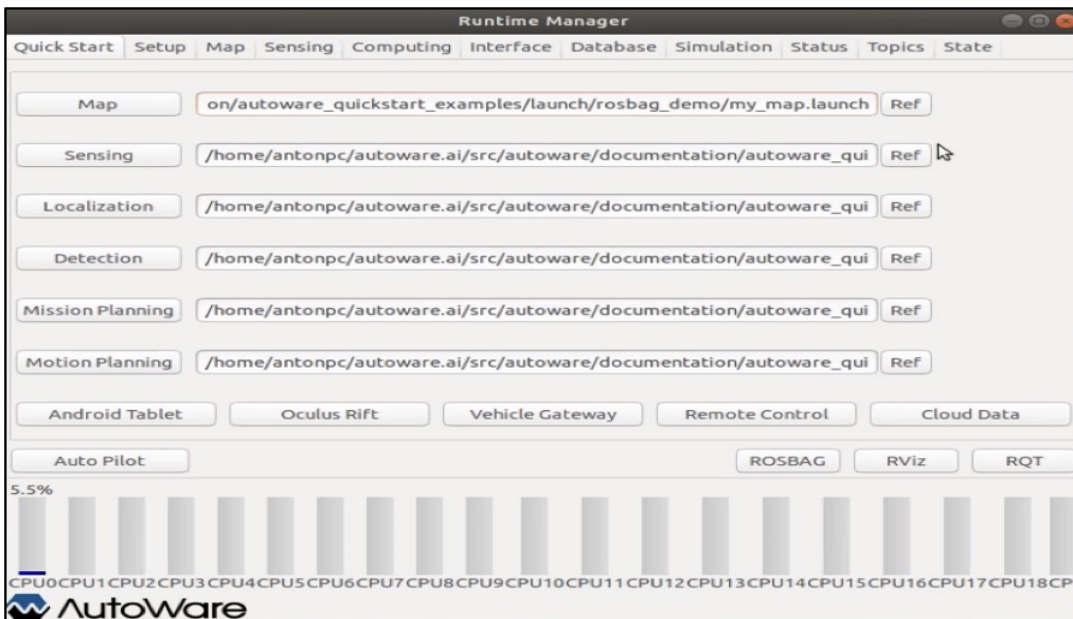


Figure 3.23: Initialization of Autoware Runtime Manager

3.8 Launch ROS bridge

It is necessary to source the setup script of the ROS workspace by using the command:
".devel/setup.bash".

This process guarantees the accurate configuration of the ROS environment settings. To initiate the execution of the "main" file in the "main_bridge" package, the command "roslaunch main_bridge main" should be executed as shown in figure 3.24. This particular main_bridge is presumed to function as a key communication hub, enabling the exchange of information between

CarMaker and Autoware. The ROS bridge facilitates the smooth and efficient interchange of data and coordination between different topics. By performing these commands, one establishes a functional connection between CarMaker and Autoware. To initiate the ROS bridge, the following steps must be taken in terminal_7.

Launch Terminal_7:

```
.devel/setup.bash  
roslaunch main_bridge main
```

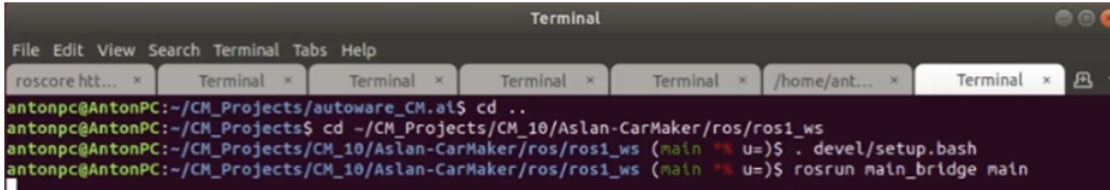


Figure 3.24: Initialization of main.cpp bridge

3.9 Information about ROS topics after starting Autoware.AI and bridge

To determine whether the messages are proceeding in the right direction after starting Autoware.AI and ROS bridge. The following steps need to be taken to implement the connection between Autoware.AI and the ROS bridge.

Open Terminal_8:

1. Visualizing the available ROS topics using command -
rostopic list We can see the /vehicle_cmd topic in the rostopic list
2. Visualizing the information about /vehicle_cmd, where /vehicle_cmd topic is responsible to send and receive messages between Autoware and Aslan_bridge. This can be visualized using the command - rostopic info /vehicle_cmd. The output of command - rostopic info /vehicle_cmd gives the following information:

type:

Autoware_msgs/VehicleCmd.

Publishers:

none

Subscribers:

*/aslan_bridge

Here the messages are Autoware_msgs which are going through /vehicle_cmd topic.

3. Visualizing the information about /sd_control, as shown in figure 3.25, where /sd_control

topic is responsible to send and receive messages from CMRosIF. This can be visualized using the command - `rostopic info /sd_control`. The output of command - `rostopic info /sd_control` gives the following information:

type:

aslan_msgs/SDControl

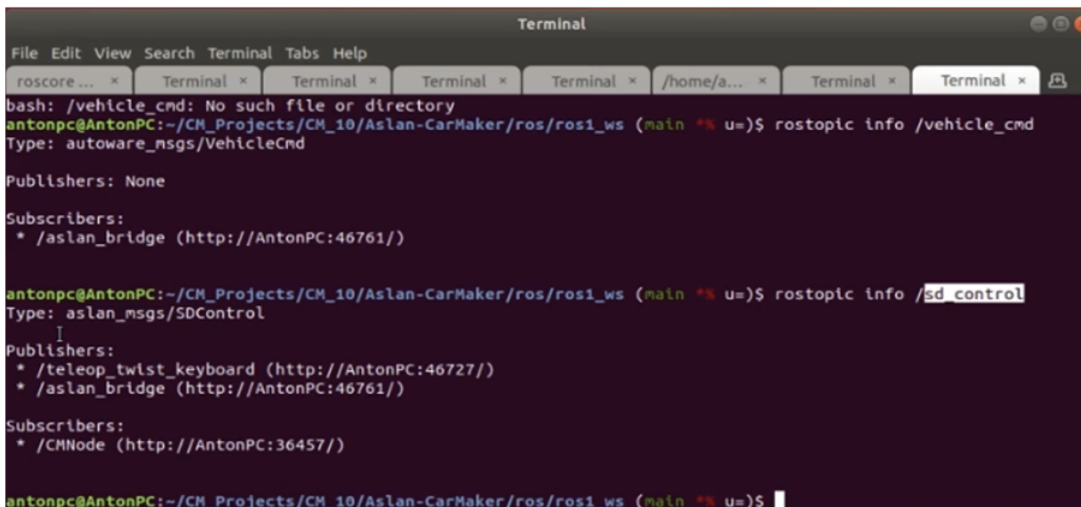
Publishers:

*/teleop_twist_keyboard

*/aslan_bridge **Subscribers:**

*/CMNode

Here the messages are `aslan_msgs` which are passing through `/sd_control` topic.



```
Terminal
File Edit View Search Terminal Tabs Help
roscore ... x Terminal x Terminal x Terminal x Terminal x /home/a... x Terminal x Terminal x
bash: /vehicle_cmd: No such file or directory
antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/ros1_ws (main *% u=)$ rostopic info /vehicle_cmd
Type: autoware_msgs/VehicleCmd
Publishers: None
Subscribers:
* /aslan_bridge (http://AntonPC:46761/)

antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/ros1_ws (main *% u=)$ rostopic info /sd_control
Type: aslan_msgs/SDControl
Publishers:
* /teleop_twist_keyboard (http://AntonPC:46727/)
* /aslan_bridge (http://AntonPC:46761/)
Subscribers:
* /CMNode (http://AntonPC:36457/)

antonpc@AntonPC:~/CM_Projects/CM_10/Aslan-CarMaker/ros/ros1_ws (main *% u=)$
```

Figure 3.25: Information of `/sd_control` and `/vehicle_cmd` topics after initializing Autoware and ROS bridge

3.10 Start Simulation with the bridge running

Ensure that both the ROS bridge and the IPG CarMaker Movie applications are running simultaneously. This section provides a comprehensive guide for configuring the Autoware Runtime Manager. It provides the steps required for establishing a connection between CarMaker and Autoware, as well as visualising the CarMaker simulation within Autoware’s RViz. Furthermore, it discusses the different selections offered by the Autoware Runtime Manager. The Autoware.AI Runtime Manager is an open-source graphical user interface (GUI) tool that incorporates a diverse set of algorithms for an autonomous driving stack as shown in figure 3.26. The software has been specifically designed to support the advancement of the Autoware autonomous system. The user-friendly GUI provides a range of tabs that facilitate the launching and handling of ROS nodes. Additionally, it allows for the setting of sensor drivers and seamless connection with visualisation tools such as RViz and RQT, that help in the visualisation of sensor data. This section discusses the fundamental functionalities that are commonly employed during the research process [40].

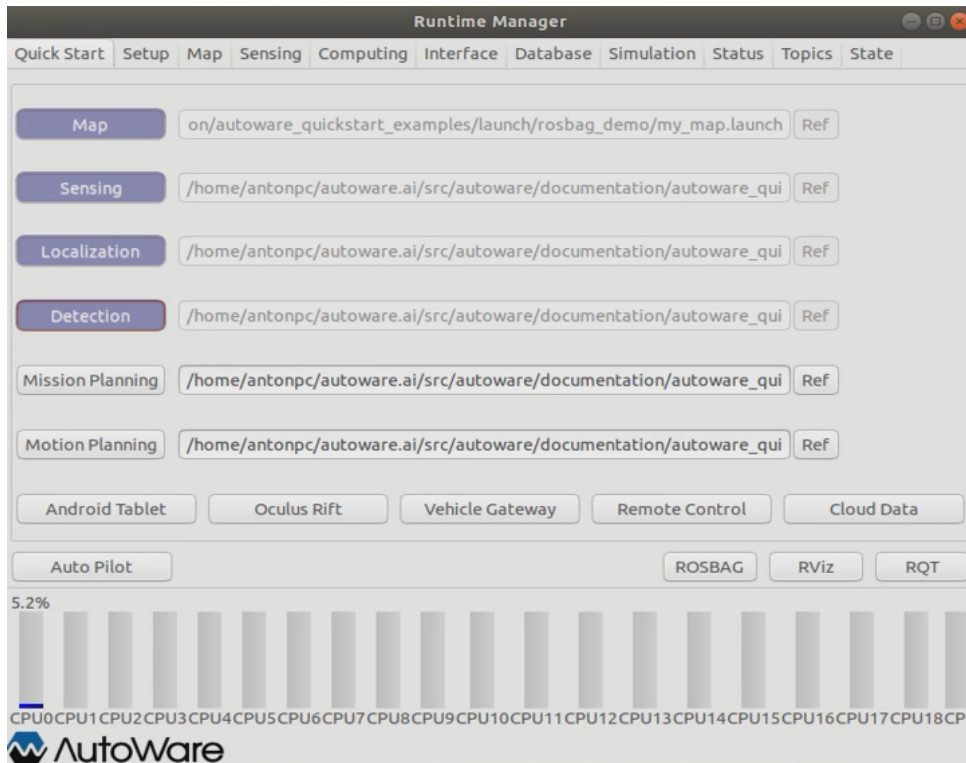


Figure 3.26: Configuration of Autoware Runtime Manager (Quick Start Tab)

1. The Quickstart tab in Autoware serves as an initial setup tool for users setting up autonomous vehicle simulations. Within this tab, there are specific sections to assist users. The Map section allows users to load 3D maps using the `my_map.launch` file. For activating vehicle sensors, the Sensing section offers the `sensing.launch` file. The Localization section provides accurate vehicle positioning capabilities through the `localization.launch` or `my_localization.launch` files. To monitor the environment around the vehicle, users can access the Detection section and integrate the `detection.launch` file. While the software has sections dedicated to planning the vehicle's route, namely Mission Planning and Motion Planning, they can be left optional for tasks like ROS bridge assessments. For convenience, each category in the Quickstart tab provides options for direct file path input or a "Ref" button to browse and select the required file.
2. In the displayed "setup" window, as shown in figure 3.27, choose "Velodyne" as the localizer due to its widespread usage in automotive applications, while the Hokuyo 3D URG is commonly used in indoor robotics. To accurately align the spatial relationship between the Velodyne sensor and the vehicle base, click on the "TF" button located underneath the "Base Link to Localizer" option. By performing this action, the system will publish a topic that describes the vehicle's control position. It is important to take into account the significance of the parameters `[x]`, `[y]`, `[z]`, `[yaw]`, `[pitch]` and `[roll]`. These parameters enable users to input and modify a relative position and orientation of the Velodyne sensor with respect to the `base_link`. By performing this procedure, one can ensure that the data collected by the

sensor is accurately synchronised with the real-time positioning and orientation of the vehicle within the simulated environment. Furthermore, to get an adequate visualisation of diverse vehicle models in RViz, the "Vehicle Model" option allows for the importation of URDF files. In the case that an URDF file has not been defined Autoware will automatically select the default URDF file The "Vehicle Info" option allows the ability to input complete data related to the particular vehicle model.

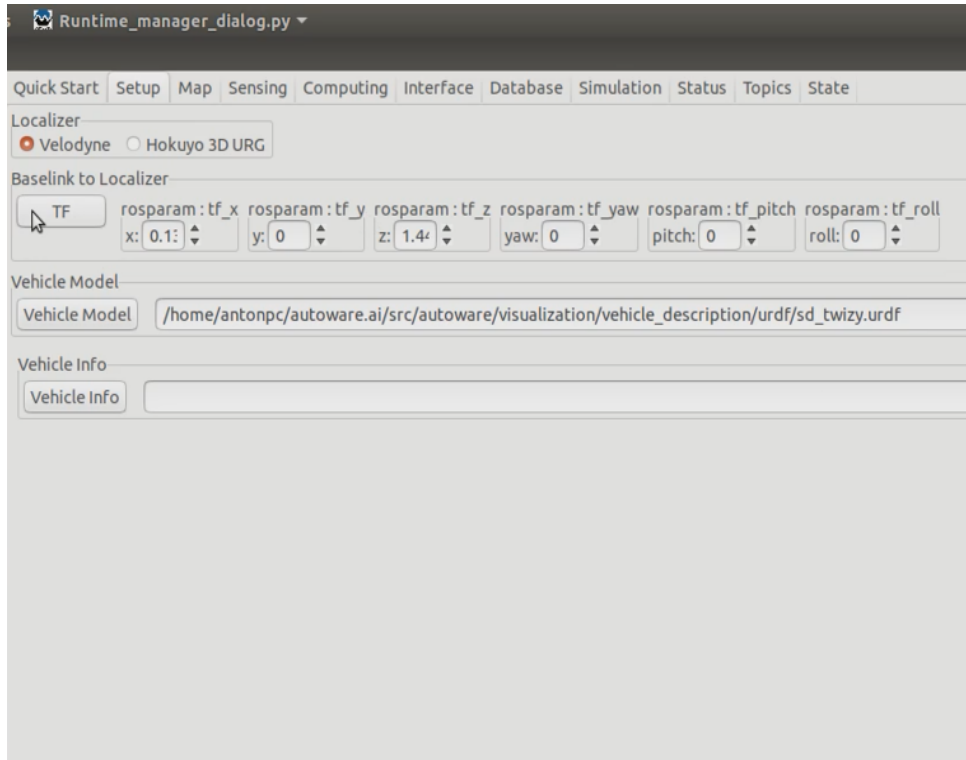


Figure 3.27: Configuration of Autoware Runtime Manager (Setup Tab)

3. Within the "Map" tab, as shown in figure 3.28, users have the capability to access and load a diverse range of map formats. The tab allows users the ability to upload PCD, which is a representation of the environment in a 3-D coordinate system that has been recorded using LIDAR sensors. Generally, the data is acquired via ROSbags by recording the topics. In the experimental setting, the primary focus was on extracting point cloud maps from the recorded ROSbags. The Map tab provides options for integrating vector maps that include road elements, lanelet maps for detailed lane information, and coordinate system transformations. Furthermore, the PCD Filter is utilised to enhance the quality of the point cloud data, whereas the PCD Binarizer is employed to transform the data into a binary format. Autoware's 3D maps differ from traditional 2D maps by including detailed information about roads and stationary objects in the surrounding environment. The localization precision of this technology is approximately 10 cm, which exceeds the standard metrics of GPS. Moreover, Autoware has the capability to generate real-time, detailed 3D maps using LiDAR data.

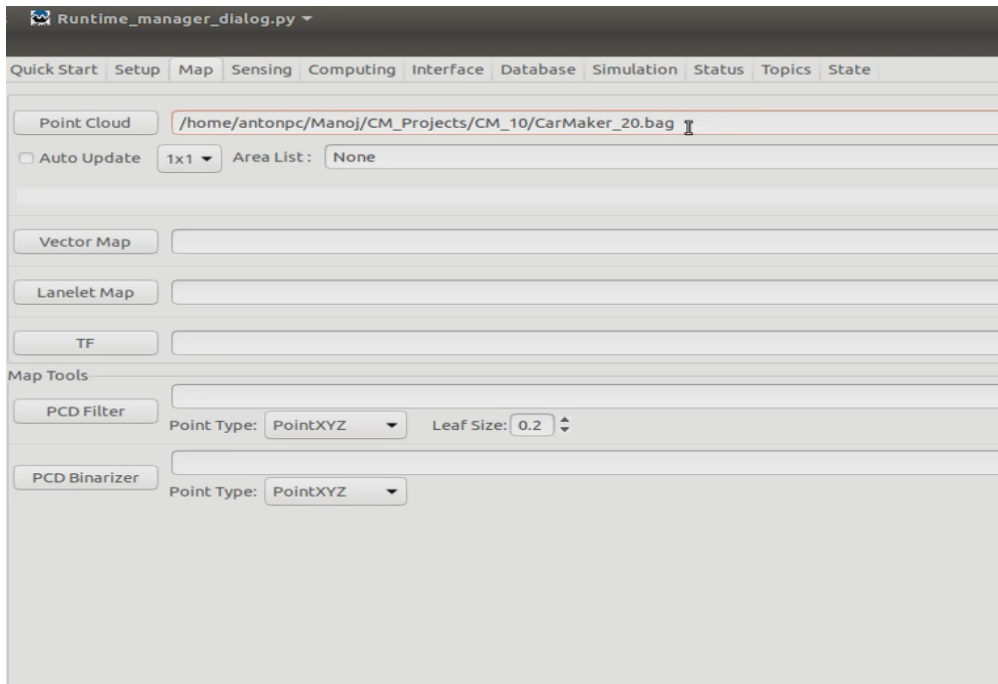


Figure 3.28: Configuration of Autoware Runtime Manager (Map Tab)

4. The Sensing tab provides a specialised interface, as shown in figure 3.29, designed for the processing and optimisation of LIDAR data, which is essential for autonomous systems applications.

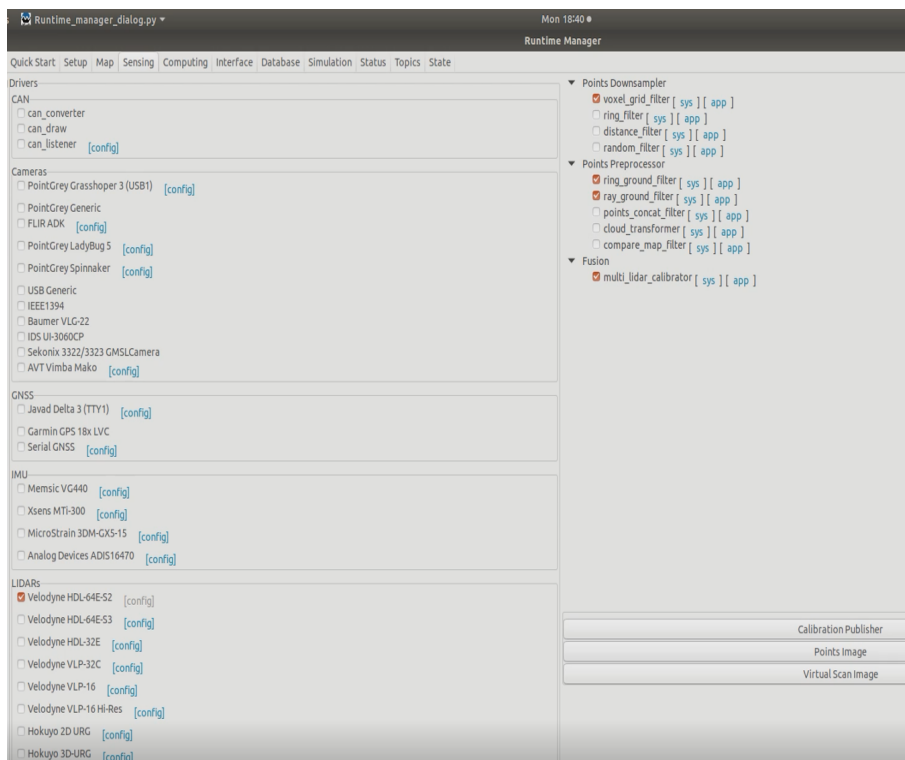


Figure 3.29: Configuration of Autoware Runtime Manager (Sensing Tab)

The Point Downsampler is crucial, with a particular focus on the `voxel_grid_filter`, which effectively reduces the data volume without compromising its integrity. The Points Preprocessor section contains options for enhancing point cloud data.

Two tools, namely the `ring_ground_filter` and `ray_ground_filter`, have been developed to accurately distinguish between ground and elevated data points, this capability is particularly important in terrains with diverse elevations, such as those containing potholes. The Fusion category includes an option called `multi_lidar_calibrator`, to harmonise data inputs from multiple LIDAR sensors for sensor data fusion, it aims to provide a unified and comprehensive environmental view. Collectively, The Sensing tab functions as a central hub for integrating various applications and functionalities essential for enhancing and optimising LIDAR data. This enhances surrounding perception and enables precise autonomous navigation.

5. The "Computing" tab in Autoware offers autonomous driving functionalities. This tab consists of four sections: Localization, Detection, Mission Planning, and Motion Planning as shown in figure 3.30. The Localization section employs tools such as `gnss_localizer` and `lidar_localizer` to determine the precise position of the vehicle. The Detection section of the vehicle allows it to observe its surroundings and identify entities on the road through the use of different sensors. After data processing, the vehicle enters the Planning phase to determine its primary route. The Motion Planning segment is responsible for the vehicle's immediate decisions, such as lane adjustments or obstacle evasion, taking into account the variability of road conditions.

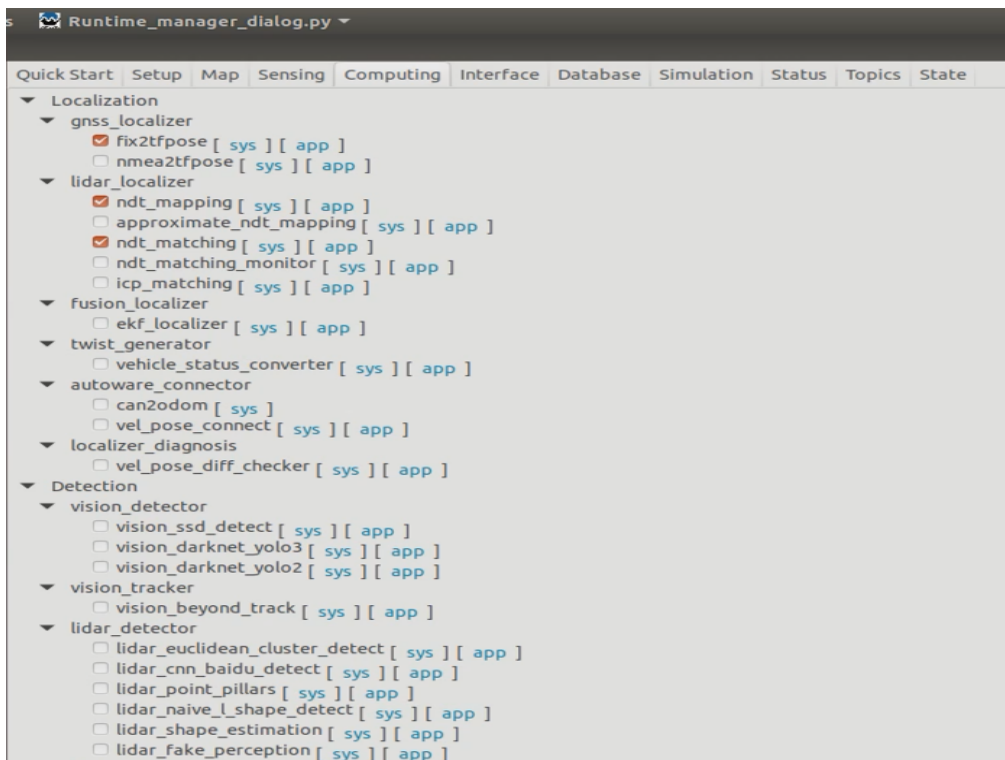


Figure 3.30: Configuration of Autoware Runtime Manager (Computing Tab)

- The "Interface" tab provides options for manual and Autopilot vehicle control operations. This tab displays a range of control sliders for users, such as Accelerator, Brake, Steer, Torque, Velocity and Angle as shown in figure 3.31. These controls allow users to directly manipulate the vehicle's motion within the CarMaker and RViz simulation environment. The design of this system allows for easy adjustment of the car's parameters during the simulation.

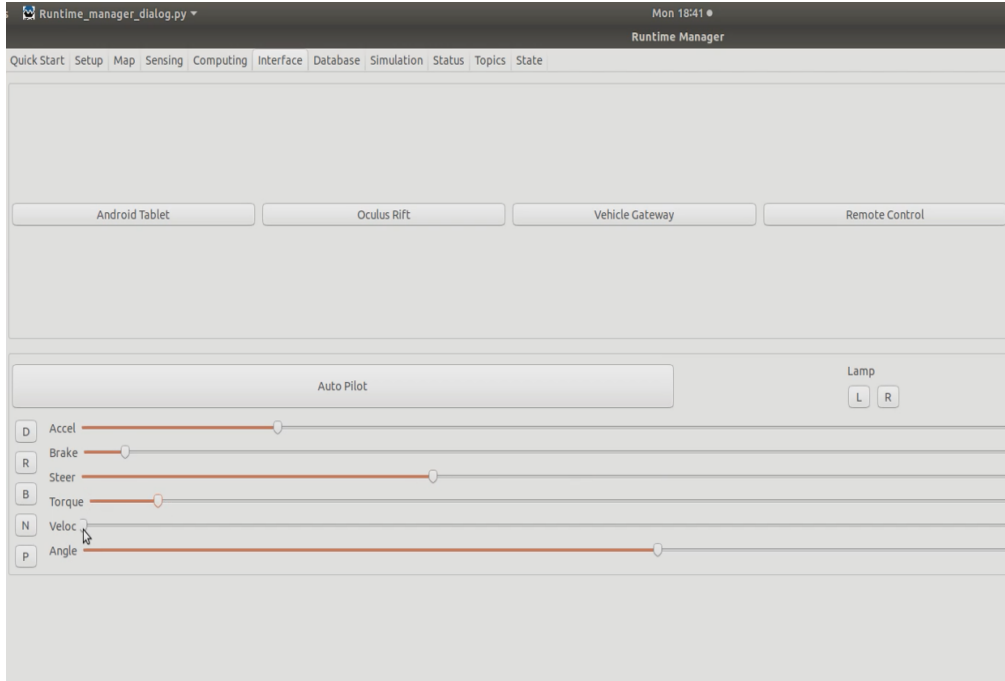


Figure 3.31: Controlling IPG Movie Simulation Using Interface Tab of Autoware Runtime Manager

- The Simulation tab as shown in figure 3.32 provides features for working with ROSbags, which are data storage files containing information from topics such as /points-raw that are generated during CarMaker simulations. The result of this process is the generation of files, specifically in the format of CarMaker_20.bag. In this tab, users can control the playback of ROSbags and convert them into Point Cloud Data (PCD) for applications such as vehicle localization. The Simulation tab enables the system to utilise recorded ROSbags instead of real-time data, facilitating the generation of PCD from pre-existing information. The "Database" tab in Autoware provides various functionalities, providing different options related to the map, position, sensors, ROSbag, Rviz, and RQT. The "Topics" tab provides users with a comprehensive list of available ROS topics. The "State" tab provides up-to-date information on the current status of the Autoware system. To establish effective connectivity between CarMaker and Autoware for co-simulation, it is essential to configure the necessary settings.

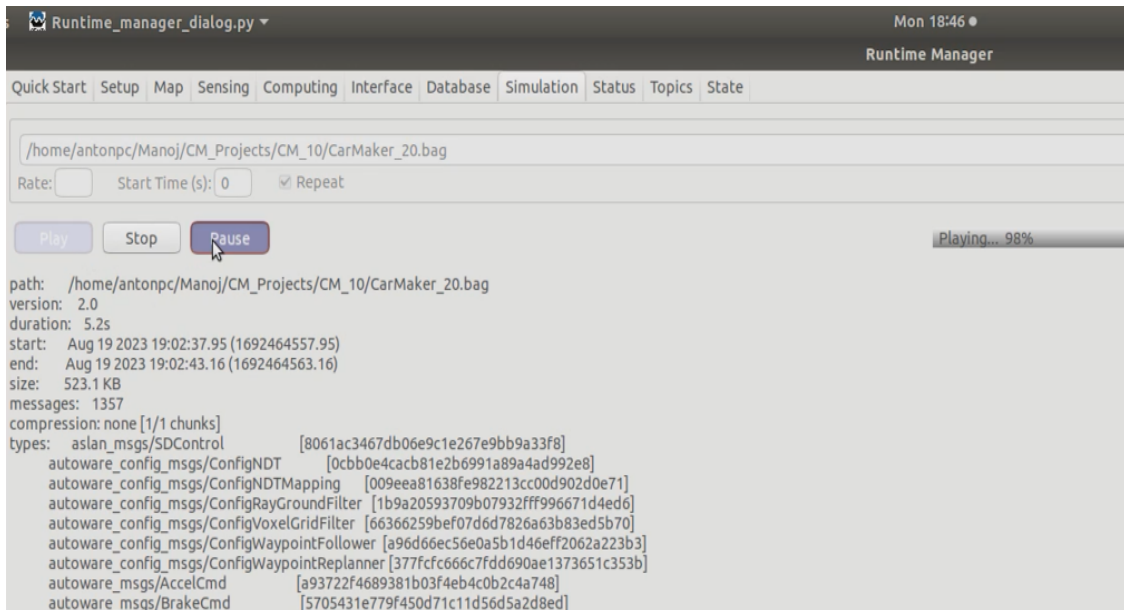


Figure 3.32: Configuration of Autoware Runtime Manager (Simulation Tab)

Configuring RViz for LIDAR Point Cloud Data Visualization:

To visualise the LIDAR point cloud data of the surrounding environment of CarMaker in RViz, click the "RViz" button located at the bottom right of the Runtime Manager as shown in figure 3.33. Once the RViz is launched, please select 'Fr_LidarRSI' as the 'Fixed Frame'. By setting the fixed frame as 'Fr_LidarRSI', the visualisation is adjusted in accordance with the angle of view of the LIDAR sensor. This feature ensures a uniform alignment and orientation of LIDAR data in the RViz, facilitating precise interpretation and analysis of the environment detected by the LIDAR sensor, regardless of the vehicle's movement.

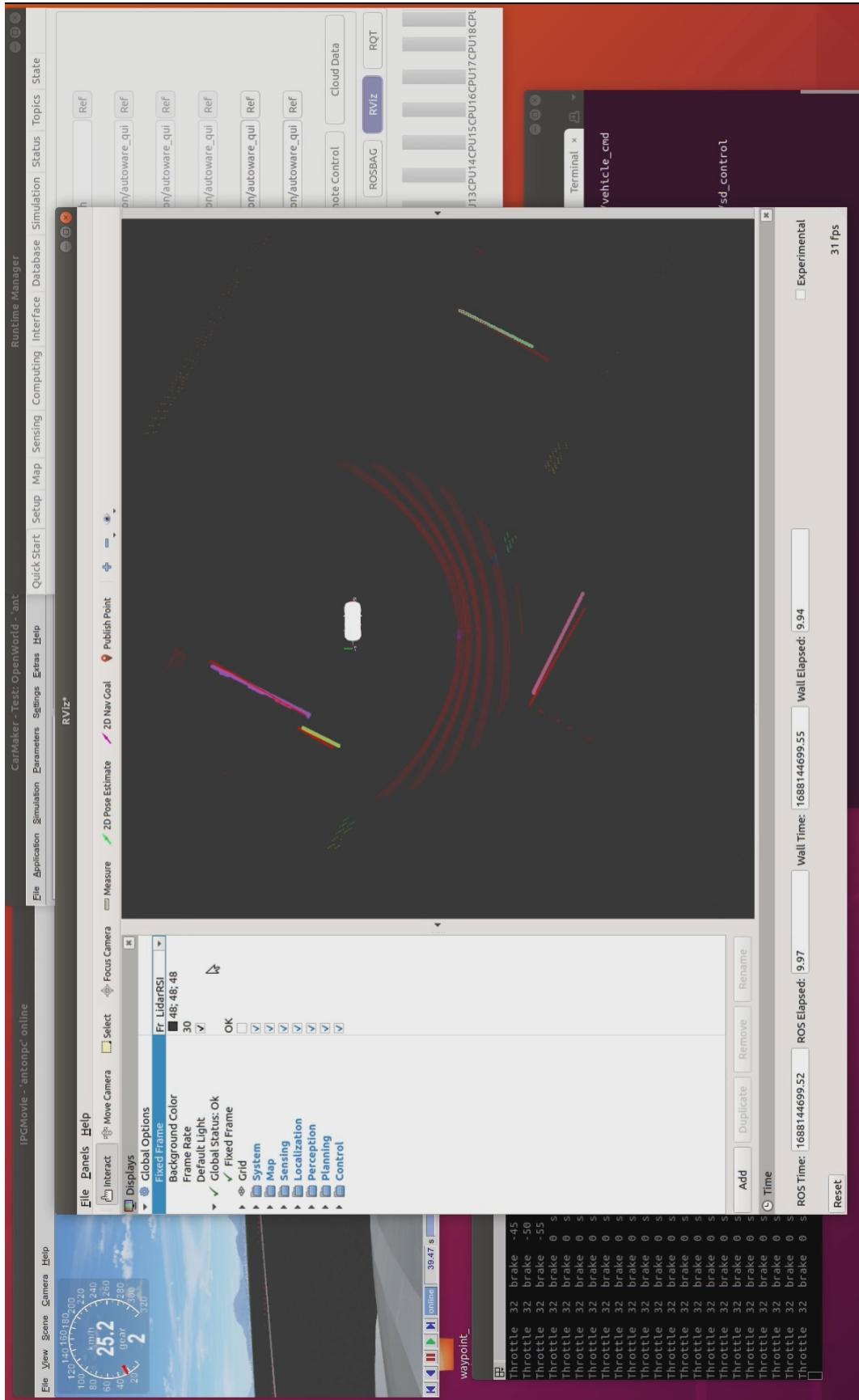


Figure 3.33: CarMaker Open World Environment PCD Map Visualisation in Autoware's RViz

Chapter 4

Results and Discussion

This thesis's findings and discussion section is divided into several sections and subsections, each of which focuses on a specific area of the research.

4.1 Evaluation of simulators and Co-Simulation Platforms

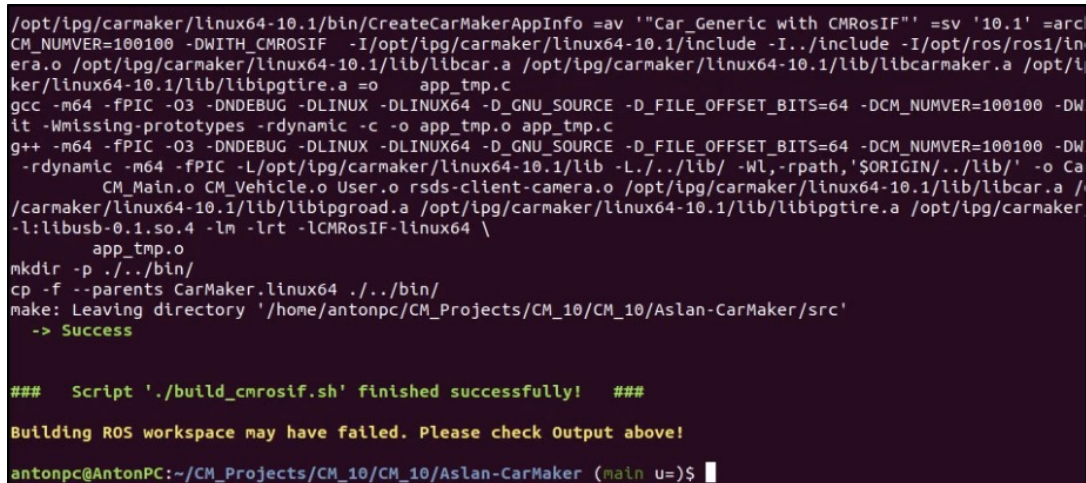
This study discusses the identification of the most effective simulators and co-simulation platforms for testing and development of autonomous driving (AD) vehicles. To evaluate how well these systems performed in carrying out crucial AD activities, several tests were developed. The investigation revealed that the platform "CARLA-Autoware" has demonstrated promising potential in terms of achieving high precision and dependability in various activities such as lane changes, obstacle avoidance, and emergency braking. Furthermore, due to the open-source software, it is continuously evolving to meet the increasing demands of the field and has proven to be effective in performing various tasks, making it a strong competitor in the field. This section of this research also highlights the significance of specific simulation tools, specifically identifying "IPG CarMaker" as a leading choice for analysing vehicle dynamics. CARLA excels in effectively utilising its visual and sensing simulation capabilities. To improve the reality of simulations, it is recommended to utilise the SUMO application for modelling huge-scale traffic movements. Significant alternatives in the commercial vehicle simulation software sector also include IPG CarMaker, ANSYS, dSPACE, PreScan etc. It is important to recognise that those applications are private, which may present challenges for researchers seeking to customise them for specific study goals or unique requirements.

4.2 Discussion about Aslan-CarMaker project

During the installation and building of the Aslan-CarMaker project, several challenges were encountered, and specific issues were addressed to ensure smooth operation:

4.2.1 Building ROS Workspace:

Error messages indicating an "undefined reference" to `SimCore_Anim DeleteMsgBuf` were encountered during the initial construction of the ROS workspace. This problem took place due to challenges in connecting multiple dependencies, ultimately leading to the eventual failure of the build process. However, these issues were resolved by rebuilding the entire script, resulting in the successful completion of the ROS workspace build as shown in figure 4.1.



```
/opt/ipg/carmaker/linux64-10.1/bin/CreateCarMakerAppInfo =av '"Car_Generic with CMRosIF"' =sv '10.1' =arcl
CM_NUMVER=100100 -DWITH_CMROSIF -I/opt/ipg/carmaker/linux64-10.1/include -I../include -I/opt/ros/ros1/in
era.o /opt/ipg/carmaker/linux64-10.1/lib/libcar.a /opt/ipg/carmaker/linux64-10.1/lib/libcarmaker.a /opt/i
ker/linux64-10.1/lib/libipgtire.a =o app_tmp.c
gcc -m64 -fPIC -O3 -DNDEBUG -DLINUX -DLINUX64 -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -DCM_NUMVER=100100 -DW
it -Wmissing-prototypes -rdynamic -c -o app_tmp.o app_tmp.c
g++ -m64 -fPIC -O3 -DNDEBUG -DLINUX -DLINUX64 -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -DCM_NUMVER=100100 -DW
-rdynamic -m64 -fPIC -L/opt/ipg/carmaker/linux64-10.1/lib -L../lib/ -Wl,-rpath,'$ORIGIN/../lib/' -o Ca
CM_Main.o CM_Vehicle.o User.o rsds-client-camera.o /opt/ipg/carmaker/linux64-10.1/lib/libcar.a /
/carmaker/linux64-10.1/lib/libipgroad.a /opt/ipg/carmaker/linux64-10.1/lib/libipgtire.a /opt/ipg/carmaker
-l:libusb-0.1.so.4 -lm -lrt -lCMRosIF-linux64 \
app_tmp.o
mkdir -p ../bin/
cp -f --parents CarMaker.linux64 ../bin/
make: Leaving directory '/home/antonpc/CM_Projects/CM_10/CM_10/Aslan-CarMaker/src'
-> Success

### Script './build_cmrosif.sh' finished successfully! ###

Building ROS workspace may have failed. Please check Output above!

antonpc@AntonPC:~/CM_Projects/CM_10/CM_10/Aslan-CarMaker (main u=)$
```

Figure 4.1: Successful reconstruction of the ROS Workspace after Error resolution

4.2.2 Compatibility with CarMaker 11.0:

The project encountered compatibility errors when using CarMaker 11.0. To resolve this, the decision was made to downgrade to CarMaker 10.1, which successfully addressed the compatibility concerns.

4.2.3 CarMaker Keyboard Control Script:

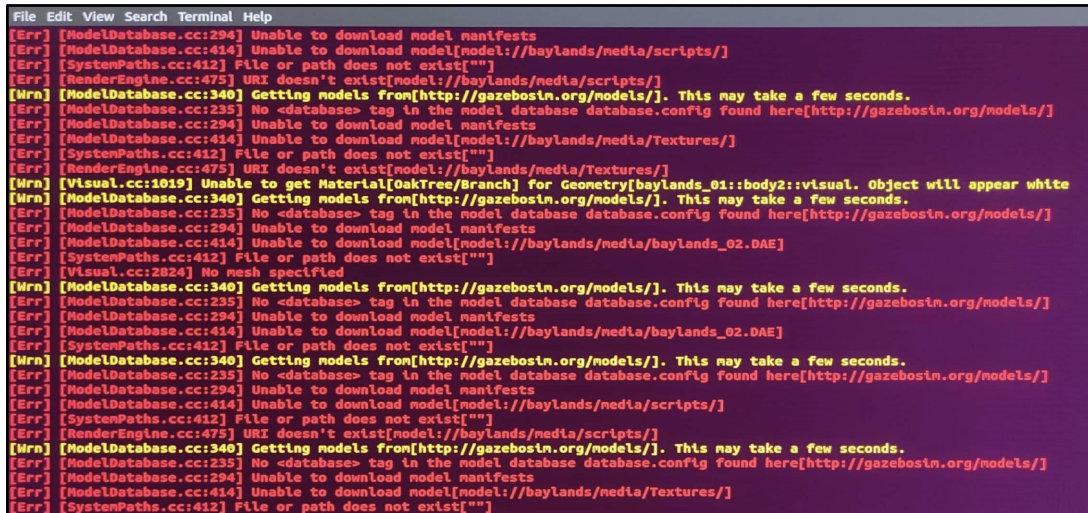
The `CarMaker_keyboard_control.py` script experienced intermittent issues and did not meet the expected performance. Re-running the Python file proved effective in resolving this intermittent issue.

4.2.4 Errors in Gazebo Simulation:

Various challenges were encountered in the Gazebo simulation during the evaluation process of the Aslan-CarMaker project. The primary challenges arose from discrepancies in versions, leading to the incorrect loading of Bayland Park and the Renault Twizy 3D model. The simulation did not execute according to the anticipated outcome. Additionally, there were issues encountered during the upgrade process of the Gazebo model database. The following presents a comprehensive analysis:

1. Problem with Automatic Gazebo Model Database Update:

In the Aslan-CarMaker project, there were difficulties encountered in the initial stages of managing the process of loading the Gazebo model database as shown in figure 4.2. The problem was fixed by downloading and incorporating all the necessary model files into the project through a manual process. The Gazebo models used in this study, including cars, pedestrians, and buildings, were sourced from the GitHub repository located at Link6 [41].



```
File Edit View Search Terminal Help
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:414] Unable to download model[model://baylands/media/scripts/]
[Err] [SystemPaths.cc:412] File or path does not exist[""]
[Err] [RenderEngine.cc:475] URI doesn't exist[model://baylands/media/scripts/]
[Wrn] [ModelDatabase.cc:340] Getting models from[http://gazebo.in.org/models/]. This may take a few seconds.
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[http://gazebo.in.org/models/]
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [SystemPaths.cc:412] File or path does not exist[""]
[Err] [RenderEngine.cc:475] URI doesn't exist[model://baylands/media/Textures/]
[Wrn] [Visual.cc:1019] Unable to get Material[OakTree/Branch] for Geometry[baylands_01::body2::visual. Object will appear white
[Wrn] [ModelDatabase.cc:340] Getting models from[http://gazebo.in.org/models/]. This may take a few seconds.
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[http://gazebo.in.org/models/]
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:414] Unable to download model[model://baylands/media/baylands_02.DAE]
[Err] [SystemPaths.cc:412] File or path does not exist[""]
[Err] [Visual.cc:2824] No mesh specified
[Wrn] [ModelDatabase.cc:340] Getting models from[http://gazebo.in.org/models/]. This may take a few seconds.
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[http://gazebo.in.org/models/]
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:414] Unable to download model[model://baylands/media/baylands_02.DAE]
[Wrn] [ModelDatabase.cc:340] Getting models from[http://gazebo.in.org/models/]. This may take a few seconds.
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[http://gazebo.in.org/models/]
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:414] Unable to download model[model://baylands/media/scripts/]
[Err] [SystemPaths.cc:412] File or path does not exist[""]
[Err] [RenderEngine.cc:475] URI doesn't exist[model://baylands/media/scripts/]
[Wrn] [ModelDatabase.cc:340] Getting models from[http://gazebo.in.org/models/]. This may take a few seconds.
[Err] [ModelDatabase.cc:235] No <database> tag in the model database database.config found here[http://gazebo.in.org/models/]
[Err] [ModelDatabase.cc:294] Unable to download model manifests
[Err] [ModelDatabase.cc:414] Unable to download model[model://baylands/media/Textures/]
[Err] [SystemPaths.cc:412] File or path does not exist[""]
```

Figure 4.2: Errors encountered with Gazebo's Model Database Automatic upgradation

2. The failure in loading the Park File (Bayland Model):

The Bayland model, which is also referred to as the park file, was not present in the current database. To resolve this matter, the model was specifically obtained from a previous database. After acquiring the "model.tar.gz" file, it was extracted and subsequently moved to the ".gazebo" directory. The Bayland model was obtained from the official website of Gazebo Simulator, specifically from the following URL: Link7 [42]. The challenges mentioned above were successfully resolved, leading to the successful running of the Gazebo simulation in the Aslan-CarMaker project framework.

The issues highlighted the need for version compatibility and careful project integration. CarMaker and Aslan's interface need enhancement, especially data exchange and visualisation. Future efforts to address version-related issues, improve data translation, and improve communication protocols may create a seamless integration that allows exact vehicle recognition and visualisation in Aslan's simulation environment.

4.3 Discussion about ROS Bridge Algorithm and Autoware-CarMaker project Evaluation

The Autoware-CarMaker project is currently in its early stages of development. The project involved the development of a new C++-programmed ROS bridge to enhance communication and data exchange between Autoware and CarMaker through the Aslan-CarMaker's (CMRosIF) interface. The bridge allows for the control of car manoeuvres in the CarMaker environment through the use of the Autoware runtime manager, as shown in figure 4.3. This development enables the use of the Autoware runtime manager's interface tab to steer the vehicle in the CarMaker simulation environment to control various vehicle actions such as braking, acceleration, steering, torque, angle and velocity.

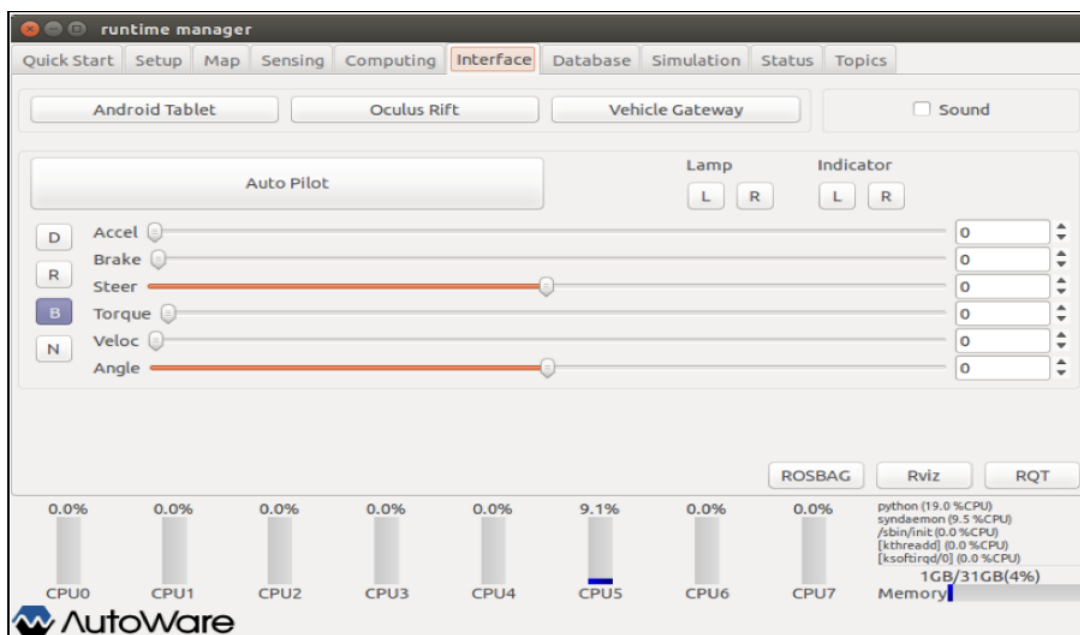


Figure 4.3: Controlling vehicle in IPG Movie simulation Using Interface Tab of Autoware Runtime Manager

Furthermore, Autoware's RViz allows for the visualisation of the vehicle, as shown in figure 4.4, and the creation of high-definition (HD) maps can be achieved by recording sensor data in ROSbag, a process similar to that employed in the Autoware-CarMaker project. This capability is advantageous for the development and testing of autonomous driving (AD) applications.

The C++ programmed ROS bridge is a key element of this project. It subscribes to the messages from 'VehicleCmd' topic and publishes to the 'SDControl' topic, acting as a middleware between Autoware and CMRosIF. The 'SDControl' topic is responsible for car steer control in CarMaker via the CMRosIF interface. The communication link between the two platforms has proven to be reliable providing a strong basis for their interaction. Despite initial positive results, further research and development are necessary in the Autoware-CarMaker project to improve efficiency and fully utilise the capabilities of Advanced Driver Assistance Systems (ADAS).

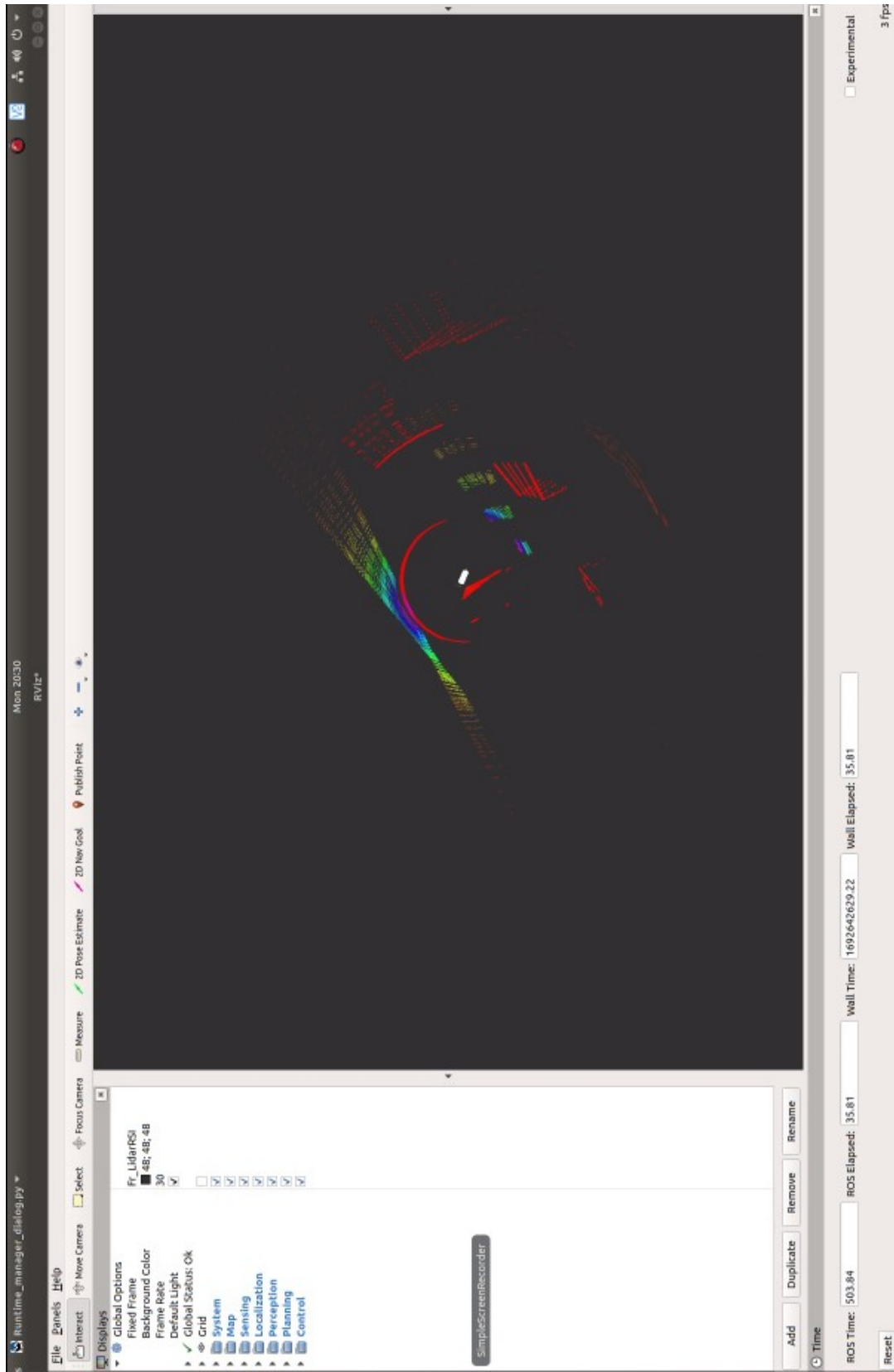


Figure 4.4: Generation of PCD Map in Autoware's RViz for CarMaker Open World Environment

Chapter 5

Conclusion and Future Scope

This chapter presents a conclusion, discusses the limitations of the research, and suggests potential possibilities for future research.

5.1 Conclusion

This research has demonstrated that in the continuously evolving realm of autonomous driving, virtual testing and co-simulation have proven to be valuable tools for efficient development. After extensive exploration of co-simulation platforms, it was determined that the Carla-Autoware.ai co-simulation emerged as the most prominent, while CarMaker was identified as the most adept simulator, specifically for vehicle dynamics.

The analysis of the Aslan-Carmaker project identified potential challenges and complications in reconstructing the project, but it also serves as a significant source of code and documentation for further research which was discussed in Chapter 4.

Another significant accomplishment of this thesis was the development of a novel ROS bridge. This bridge enables data flow between Autoware.AI and CarMaker for inter-process communication. Until now, no one has accomplished the co-simulation operation of Autoware.AI and CarMaker on a single Linux machine. This integration not only facilitated seamless communication but also enabled the manual control of a car within the CarMaker environment through the Autoware.AI runtime manager's control tab.

In conclusion, this thesis has successfully achieved its main objective including the Implementation of the interface, and obtaining satisfactory results. This study emphasised the importance of choosing the right platforms and tools, and it laid a solid foundation for further work on co-simulation for autonomous vehicles.

5.2 Future Scope

Integrate 3D ANTON model in IPG CarMaker: The aim of this future study will be to incorporate a 3D simulation model of ANTON, including simplified sensor and actuator models, into IPG CarMaker. For integration, the 3D object description files should be either in OBJ, MOBJ, DAE, or KMZ file extensions. The model has to be precisely scaled in order to fit into the CarMaker environment. The main purpose of this Integration is to thoroughly assess and verify the algorithms and software prior to their deployment in the actual ANTON vehicle. Link8 [43]

Testing the performance of the developed ROS bridge: This study will involve conducting comprehensive testing and validation on the developed coupling interface in conjunction with the ANTON simulation model in IPG-CarMaker. This process will also include performing various test scenarios, evaluating the performance of the interface in terms of data exchange, accuracy, and efficiency, and analysing the results to assess the effectiveness of the developed ROS bridge.

There is a prospect to utilize CarMaker-11.0 latest version for testing and simulation as it has numerous enhanced features compared to the previous versions of CarMaker-10.1.

Considering the industry's shift towards ROS2, it is advisable to work on ROS2-based autonomous driving stack such as Autoware.auto. this provides enhanced capabilities such as real-time performance, safety and privacy.

Bibliography

- [1] Autoware Foundation, *Autoware - the World's Leading Open-Source Software project for autonomous driving*, GitHub, 2022. [Online]. Available: <https://github.com/autowarefoundation/autoware>.
- [2] R. Ungati, *IPG Users Guide | PDF | Computer Simulation | Steering*, Scribd. [Online]. Available: <https://www.scribd.com/document/516617686/3-IPG-Users-Guide> (visited on 08/14/2023).
- [3] V. Ondřej, T. de Borba, and D. Ömer, “Holistic Environment for Development and Testing of Cooperative, Connected and Automated Mobility Functions,” FISITA World Congress 2023, Barcelona, September 2023.
- [4] V. Ondřej and et.al, *Introduction to ANTON*, www.thi.de, 2022. [Online]. Available: <https://www.thi.de/forschung/carissma/c-isafe/projekt-anton/> (visited on 09/14/2023).
- [5] H. Németh and et.al., *Proving Ground Test Scenarios in Mixed Virtual and Real Environment for Highly Automated Driving*. Springer eBooks, 2019, pp. 199–210. DOI: 10.1007/978-3-658-26107-8_15. (visited on 08/14/2023).
- [6] V. Ricciardi and et.al., “Estimation of Brake Friction Coefficient for Blending Function of Base Braking Control,” *SAE International Journal of Passenger Cars - Mechanical Systems*, vol. 10, pp. 774–785, 2017. DOI: 10.4271/2017-01-2520. (visited on 03/18/2022).
- [7] S. Bogomil and D. Barry, *Aslan-CarMaker*, last accessed: 2023-08-06, GitHub, 2021. [Online]. Available: <https://www.github.com/IPG-Automotive-UK/Aslan-CarMaker>.
- [8] S. La Bua, *Testing ROS Bridge on the TurtleBot3-SLB Labs*, SLB LABS, 2020. [Online]. Available: <https://www.slblabs.com/2020/08/04/testing-ros-bridge-on-the-turtlebot3/> (visited on 08/07/2023).
- [9] H. Kagalwala and et.al., “Implementation Methodologies for Simulation as a Service (SaaS) to Develop ADAS Applications,” *SAE International Journal of Advances and Current Practices in Mobility*, vol. 3, pp. 2123–2135, 2021. DOI: 10.4271/2021-01-0116. [Online]. Available: <https://par.nsf.gov/servlets/purl/10316066> (visited on 08/14/2023).
- [10] J. Blanc, “Control Systems Design for Automatic Drive of a Passenger Car in Critical

- Scenarios,” Ph.D. dissertation, University of Windsor, 2016, pp. 1–178. [Online]. Available: <https://scholar.uwindsor.ca/cgi/viewcontent.cgi?article=6803&context=etd>.
- [11] IPG CarMaker, *Benefits of CarMaker*, ipg-automotive.com. [Online]. Available: <https://ipg-automotive.com/en/products-solutions/software/carmaker/>.
- [12] N. Brown and et.al., “Development of an Energy Efficient and Cost Effective Autonomous Vehicle Research Platform,” *Sensors*, vol. 22, p. 5999, 2022. DOI: 10.3390/s22165999. [Online]. Available: <https://www.mdpi.com/1771158> (visited on 10/26/2022).
- [13] IPG Automotive, *Solutions For Virtual Test Driving CarMaker Virtual Development and Testing of Passenger Cars and Light-Duty Vehicles*. [Online]. Available: <https://ipg-automotive.com/de/>. (visited on 08/19/2023).
- [14] IPG Automotive, *Usage of CarMaker with ROS*, ipg-automotive.com, 2019. [Online]. Available: <https://ipg-automotive.com/de/support/supportanfrage/faq/how-can-i-use-carmaker-with-ros-90/>.
- [15] IPG Automotive, *Aslan-carmaker project CMRosIF_UsersGuide*, GitHub, 2019. [Online]. Available: <https://github.com/IPG-Automotive-UK/Aslan-CarMaker/tree/main/doc> (visited on 08/19/2023).
- [16] Ros.org, *ROS Tutorials, documentation and source*, wiki.ros.org, 2022. [Online]. Available: <https://wiki.ros.org/ROS/Tutorials>.
- [17] Autoware Foundation, *Autoware.AI driving vehicle platform*, Cloudfront.net, 2020. [Online]. Available: <https://d3i71xaburhd42.cloudfront.net/49c0769b9100ed7148648ce11740055f04ca864d/2-Figure1-1.png> (visited on 09/14/2023).
- [18] Autoware Foundation, *Autoware.AI-User Manuals*, GitHub, 2017. [Online]. Available: <https://github.com/CPFL/Autoware-Manuals> (visited on 08/08/2023).
- [19] Trossen Robotics, *AgileX Robotics Autoware Open Source Autonomous Kit*. [Online]. Available: <https://www.trossenrobotics.com/Shared/Agilex/agilex-robotics-autoware-user-manual-2.pdf> (visited on 08/08/2023).
- [20] G. Rong and et.al., “LGSVL Simulator a High Fidelity Simulator for Autonomous Driving,” in 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 2020, pp. 1–6. [Online]. Available: <https://arxiv.org/pdf/2005.03778.pdf>.
- [21] Bshin-lge, *Simulator-2019.05-obsolete/Docs/docs/autoware-instructions.md at Master:lgsvl/simulator-2019.05-obsolete*, GitHub, 2019. [Online]. Available: <https://github.com/lgsvl/simulator-2019.05-obsolete/blob/master/Docs/docs/autoware-instructions.md> (visited on 08/08/2023).
- [22] V. R. Aparow and et.al, “A Comprehensive Simulation Platform for Testing Autonomous Vehicles in 3D Virtual Environment,” *2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR)*, pp. 115–119, 2019. DOI: 10.1109/icmsr.2019.8835477. (visited on 07/30/2023).

- [23] Xinetzone, *Apollo Auto documentation*, daobook.github.io, 2021. [Online]. Available: <https://daobook.github.io/apollo/README.html> (visited on 09/15/2023).
- [24] M. Reke and et.al., "A Self-Driving Car Architecture in ROS2," *2020 International SAUPEC/RobMech/PRASA Conference*, pp. 1–6, 2020. DOI: 10.1109/saupec/robmech/prasa48453.2020.9041020.
- [25] S. F. Santonato, "A Complete End-To-End Simulation Flow for Autonomous Driving Frameworks. Master thesis, Politecnico di Torino University," webthesis.biblio.polito.it, 2020. [Online]. Available: <http://webthesis.biblio.polito.it/id/eprint/16703>.
- [26] L. Drive, *Using Autoware.AI as a Framework for OEM-backed Autonomous Vehicle Projects*, www.linkedin.com, 2022. [Online]. Available: <https://www.linkedin.com/pulse/using-autoware-framework-oem-backed-autonomous-vehicle-projects-/> (visited on 08/14/2023).
- [27] S. Stevic and et.al., "Development and Validation of ADAS Perception Application in ROS Environment Integrated with CARLA Simulator," *2019 27th Telecommunications Forum (TELFOR)*, 2019. DOI: 10.1109/telfor48224.2019.8971063. (visited on 08/07/2023).
- [28] SUMO Foundation, *Eclipse SUMO - Simulation of Urban MObility (SUMO User Documentation)*, Eclipse SUMO - Simulation of Urban MObility, 2018. [Online]. Available: <http://sumo.dlr.de/index.html> (visited on 08/18/2023).
- [29] L. P. Alvarez and et.al., "Microscopic Traffic Simulation using SUMO," *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2575–2582, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:54462985>.
- [30] K. Tong and et.al., "Overview of Tools Supporting Planning for Automated Driving", 2020, pp. 1–8. [Online]. Available: <https://arxiv.org/pdf/2003.04081.pdf> (visited on 08/07/2023).
- [31] IPG Automotive GmbH, *CarMaker Tips and Tricks (CarMaker ROS Interface)*, GitHub, 2019. [Online]. Available: https://github.com/IPG-Automotive-UK/Aslan-CarMaker/blob/main/doc/3-004-1_CMROSInterface.pdf (visited on 08/19/2023).
- [32] MathWorks team, *Exchange Data with ROS Publishers and Subscribers*, Mathworks.com, 2023. [Online]. Available: https://de.mathworks.com/help/examples/ros/win64/ExchangeDataWithROSPublishersAndSubscribersExample_01.png (visited on 09/15/2023).
- [33] ROS.org, *ROS Melodic Installation and tutorials*, wiki.ros.org, 2020. [Online]. Available: <http://wiki.ros.org/melodic/Installation/Ubuntu>.
- [34] Ros.org, *Installation and Documentation of Catkin_Tools*, catkin-tools.readthedocs.io, 2014. [Online]. Available: <https://catkin-tools.readthedocs.io/en/latest/installing.html> (visited on 08/09/2023).
- [35] ROS.org, *ROS Melodic Environment setup*. [Online]. Available: <http://wiki.ros.org/melodic/Installation/Ubuntu>.

- [36] IPG Automotive-UK, *Aslan-carmaker*, 2021. [Online]. Available:
<https://github.com/IPG-Automotive-UK/Aslan-CarMaker>.
- [37] Autoware Foundation, *Autoware.AI Documentation and useful resources*. [Online]. Available: <https://github.com/Autowarefoundation/Autoware/>.
- [38] Project Aslan, *Aslan Messages and Packages*. [Online]. Available:
https://github.com/project-aslan/Aslan/tree/melodic/src/msgs/aslan_msgs.
- [39] Autoware Foundation, *Autoware.AI messages and packages*. [Online]. Available:
https://github.com/autowarefoundation/autoware_ai_messages/tree/master.
- [40] K. C. Konda, "Integration of Vehicle Interface to Autoware. Technische Hochschule Ingolstadt," Faculty of Electrical Engineering and Information Technology, Technische Hochschule Ingolstadt, 2022. [Online]. Available:
<https://opus4.kobv.de/opus4-haw/files/3450/I001098976Thesis.pdf>.
- [41] Gazebo.org, *Gazebo database of SDF models*, 2013. [Online]. Available:
https://github.com/osrf/gazebo_models.
- [42] Gazebo.org, *Baylands park model files*, 2013. [Online]. Available:
<http://models.gazebo.org/baylands/>.
- [43] IPG Automotive, *Customization of 3D movie objects in IPG CarMaker*, 2018. [Online]. Available:
https://ipg-automotive.com/uploads/tx_pbfaqtickets/files/115/Scale_3D.pdf.

Appendix

ROS Bridge (main.cpp) Program:

```
1 // Import ros library
2 #include <ros/ros.h>
3
4 // Import SDControl.h file to communicate with Aslan CarMaker
5 #include "aslan_msgs/SDControl.h"
6
7 // Import VehicleCmd.h file to communicate with Autoware.ai
8 #include "autoware_msgs/VehicleCmd.h"
9
10 // Create class Aslan Bridge
11 class AslanBridge {
12
13     private:
14         aslan_msgs::SDControl sd_msg;
15         ros::Publisher sd_pub;
16         ros::Subscriber vehicle_cmd;
17
18     public:
19         AslanBridge(ros::NodeHandle *nh) {
20             sd_pub = nh->advertise<aslan_msgs::SDControl>("/sd_control", 1000);
21             vehicle_cmd = nh->subscribe("/vehicle_cmd", 1000,
22                 &AslanBridge::vehicleCmdCallback, this);
23         }
24
25         void vehicleCmdCallback(const autoware_msgs::VehicleCmd::ConstPtr& msg)
26             /* Converting and assigning /vehicle_cmd data to /sd_sontrol data */
27             sd_msg.header = msg->header;
```

```

28     sd_msg.steer = msg->steer_cmd.steer;
29     if(msg->brake_cmd.brake > 0){
30         sd_msg.torque = -20.0*msg->brake_cmd.brake;
31     }
32     else{
33         sd_msg.torque = 20.0*msg->accel_cmd.accel;
34     }
35
36     /* Publishing the new sd_msg to control the vehicle*/
37     sd_pub.publish(sd_msg);
38 }
39 };
40
41 // Initiate C++ main function
42 int main (int argc, char **argv)
43 {
44     // Initiate the ROS node
45     ros::init(argc, argv, "aslan_bridge");
46
47     // Define the ROS node handler
48     ros::NodeHandle nh;
49
50     // Create an instance of AslanBridge
51     AslanBridge nc = AslanBridge(&nh);
52
53     ros::spin();
54 }

```