Journal of Artificial Intelligence Practice (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/jaip.2023.060303
ISSN 2371-8412 Vol. 6 Num. 3

A rapid simulation development platform for autonomous driving based on CARLA and ROS

Zhou Su[1,2,a], Zhu Zhenhua[1,b,*], Zhu Xiaofeng[1,c]

[1]School of Automotive Studies, Tongji University, Shanghai, China
[2]Sino-German College, Tongji University, Shanghai, China
[a]suzhou@tongji.edu.cn, [b]zhuzhenhua@tongji.edu.cn, [c]zhuxiaofeng@tongji.edu.cn
[*]Corresponding author

Keywords: Simulation platform, CARLA, Motion planning, Semantic mapping

Abstract: This paper discusses the development of a rapid simulation development platform based on CARLA and ROS. Firstly, the high cost and difficulty of algorithm verification in real-world experiments, mapping, and planning were introduced. The goal of accelerating research and development efficiency through the use of simulation development platforms was proposed. Based on these requirements, a multi-level platform architecture was designed, and the platform architecture, construction process, and related applications were introduced, creating a rapid development platform for autonomous driving simulation tasks. Finally, using mapping experiments and motion planning experiments as examples, the application of the rapid development platform was introduced.

1. Introduction

The functionalities of autonomous vehicles require extensive real-world testing to meet commercial demands before they can be put into commercial use. However, autonomous vehicle road tests face various issues such as high costs, traffic regulations, unclear accident liability, and low efficiency in extreme testing conditions. Using simulation platforms is an important solution to address the aforementioned problems. By building a simulation development platform for autonomous driving, relevant driving modules can be tested and validated in an offline environment, accelerating the development process.

For the semantic mapping task of Automated Valet Parking (AVP) [1], most parking lots are indoors and cannot use the commonly used GNSS combination positioning system as the vehicle's position ground truth, resulting in a lack of verification methods for the established map. For the motion trajectory planning task, users often have a demand for custom maps to conduct planning tests in specific scenarios. To address these requirements, this paper proposes and develops an autonomous driving rapid simulation development platform.

With the increasing demand for autonomous driving development and testing, the number of autonomous driving simulation software platforms is constantly growing, and their relevant functionalities are constantly improving. According to the software distribution method, they can be divided into commercial simulation software and open-source simulation software. Commercial autonomous driving simulation software includes Prescan, VIRES VTD, and others, while open-

source simulation software includes Gazebo [2], CARLA [3], AirSim [4], and others. Most commercial software is independently developed by commercial companies and they do not disclose their underlying interfaces and source code to the public.

Compared with commercial platforms, open source autonomous driving simulation platforms offer more open interfaces, which are conducive to secondary development and in-depth scientific research. AirSim [4], developed by Microsoft Research, is a simulation platform for unmanned aerial vehicles and autonomous driving based on the Unreal Engine. Its main goal is to serve as a platform for AI research, testing deep learning and end-to-end reinforcement learning algorithms, but its adaptability is relatively poor for mapping and planning tasks due to its lack of panoramic map output. CARLA [3], developed by the Computer Vision Center at the Autonomous University of Barcelona, can import customized scene maps and provides a configurable Python interface that supports flexible sensor and environment configurations. It can customize the import of relevant maps and communicate and simulate through a server-client architecture. Gazebo [2] is an independent robot simulation platform that can load custom simulation environments and provides many sensor plugins. Its dedicated text markup language can easily implement robot function configurations. Gazebo is currently the most compatible simulation tool for the Robot Operating System (ROS), which is the mainstream communication middleware for developing robots and autonomous driving systems. This article plans to build a rapid development platform based on autonomous driving simulator and ROS to test and verify mapping and planning algorithms.

2. Platform Architecture & Implementation

The quick simulation development platform employs a modular layered architecture, as shown in Figure 1, to achieve decoupling of different levels of software through encapsulation. The entire development platform consists of six levels, namely simulator, communication middleware, encapsulation module, application software, algorithm node, and interactive interface.

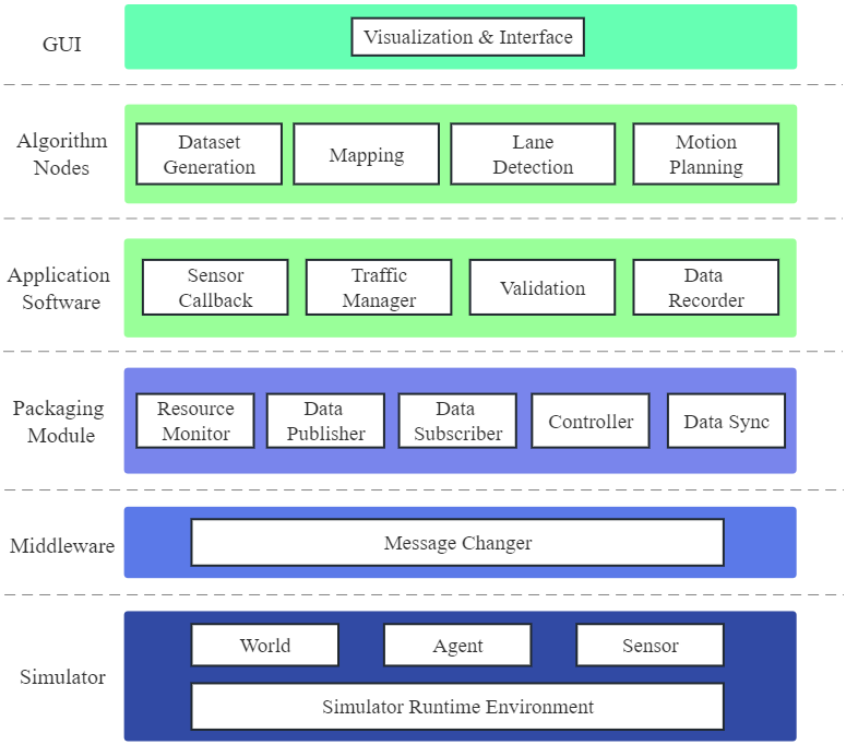

Figure 1: Rapid Simulation Development Platform Architecture

The simulator includes a runtime environment, a world model, an agent model, and a sensor model. After the simulator is configured, simulation data is asynchronously communicated to the upper-layer application through the communication middleware. This paper uses data formats and communication middleware based on the robot operating system. The encapsulation module includes basic functions such as data synchronization, data sending and receiving units, and vehicle controllers that interact with the simulator, serving as a bridge between the application software and the simulator along with the communication middleware. The application software layer utilizes the encapsulated basic functions to accomplish more complex logical applications, such as processing sensor data, controlling the movement of multiple vehicles, and a recorder for data persistence.

Each algorithm node is an independent running process that integrates application software and adds algorithm functions to jointly accomplish functions such as semantic mapping and motion planning, and displays corresponding results through the visualization module. Platform users can adjust and update algorithm modules to speed up system development by viewing the effects of relevant algorithms through the visual interface. The functions and construction process of each level will be detailed in the following sections.

2.1. Simulation Environment Configuration

The autonomous driving simulator is an important component of the platform and serves as the source of simulation data. Open-source simulators have better openness and scalability than commercial products, as modifications to the model can be directly made by editing the source code, and they are updated more frequently, making them more suitable for rapid development platforms. The rapid development platform uses data formats and communication middleware based on ROS, and CARLA and Gazebo have better ROS support in open-source simulators. Therefore, this paper explores the use of CARLA and Gazebo for simulation.

The CARLA simulator is mainly used for autonomous driving road simulation, and a Python script can be used to write a client that communicates with the CARLA server. CARLA provides a rich environment of cities, straight roads, intersections, and crossroads, among many other different driving scenarios, and also provides basic path elements and topological connections, which provide strong support for simulating real driving scenarios and global planning work. Figure 2(a) and (b) show the non-layered map and aerial view map of Town02 in CARLA, respectively.

Figure 2: a) Non-layered map b) Aerial view map

The preceding section of requirements analysis also highlighted the need for more ground elements in semantic mapping, as well as the need for custom maps in motion planning. Furthermore, the map editor for CARLA is still immature, and modifying the map requires the support of multiple third-party software, which is inconvenient. Another robot simulator, Gazebo, offers greater freedom in

map editing and is currently the best simulator for supporting ROS, with a wide range of plug-in support. It is mainly used for simulating robots. By writing world and model files to build the environment, Gazebo can also be used in the field of autonomous driving. Figure 3 shows a self-constructed autonomous valet parking scene, which includes various types of ground marking elements, static pillar targets, dynamic vehicle targets, and more. Different categories of ground elements are distinguished by different colors: blue for parking space lines, magenta for lane lines, red for center dashed lines, yellow for indication signs, cyan for sidewalks, green for stop lines, and brown for deceleration zones. Gray pillars represent the pillars in the underground parking lot. In addition to the parking lot shown in the figure, there are multiple layouts of different parking lots with different parking space lines and ground markings. With the aid of dynamic vehicle targets, various parking environments can be simulated.

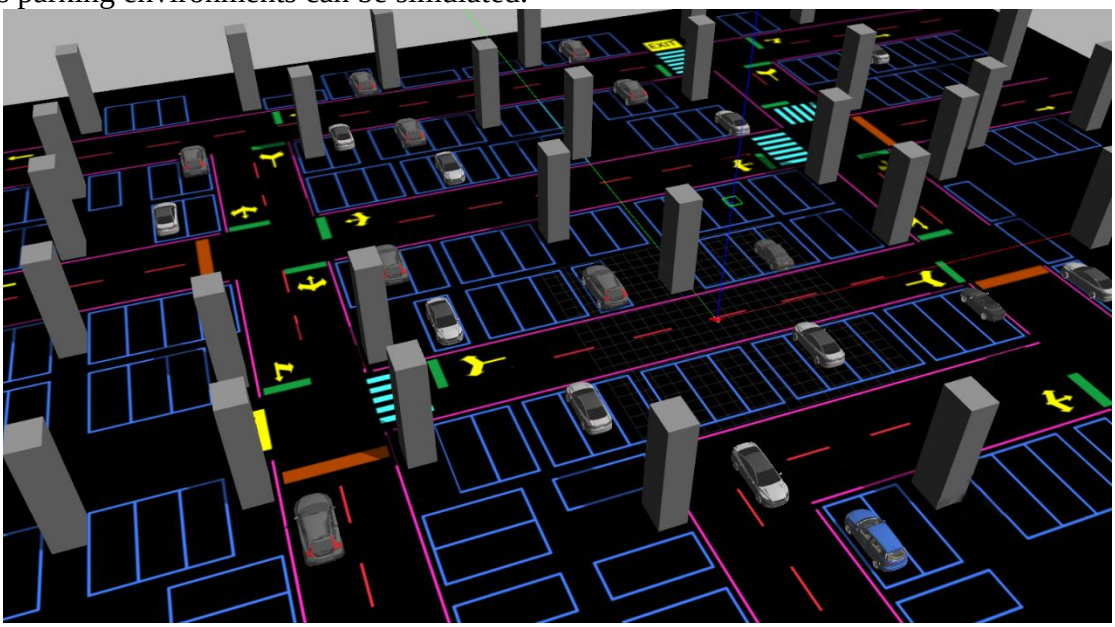

Figure 3: Custom map scenes built in Gazebo

Compared to the Unreal Engine used by the CARLA simulator, Gazebo's rendering engine is relatively simple. However, on the other hand, Gazebo consumes less computational resources, allowing for the use of more intelligent agents and sensors. After constructing the simulation environment, the next step is to configure the vehicle and sensor models.

2.2. Vehicle and Sensor Configuration

Regarding the necessary vehicle models, CARLA has pre-existing realistic vehicle models that can be directly used, whereas for Gazebo, one needs to build the model from scratch. In CARLA, the platform uses the Mercedes-Benz Coupe as the data acquisition car blueprint, whereas in Gazebo, a differential wheeled robot is used as the data collection vehicle.

In the simulation development platform, software applications for image data recording and algorithm nodes need to construct a perception system using cameras. Hence, RGB cameras are mounted at different positions on the vehicle model for data collection, storage and subsequent environment perception tasks. In addition to the primary perception devices, additional sensors have also been configured to support simulation work. The specific sensor configurations and their primary functions are shown in the Table 1.

The RGB camera at the front of the vehicle is used to capture the front view images, generate CARLA target detection data packages and support local storage and training. The image size

parameters and installation positions of some sensors are based on the corresponding settings of the data acquisition vehicles in KITTI [5].

Table 1: Sensor Configuration

Sensor devices	Main function
RGB Camera	Providing environmental images
GNSS	Providing vehicle position information
IMU	Providing vehicle movement information
Obstacle Sensor	Providing dynamic and static obstacle information
Collision Detector	Providing auxiliary information
Road Occupancy Detector	Providing auxiliary information

2.3. Communication Middleware and Packaging Modules

The middleware is responsible for data transmission, as mentioned earlier. The rapid development platform utilizes the data format and communication mechanism of ROS. The most commonly used communication mechanism in ROS is the publisher-subscriber model. In this platform, the simulator will act as the publisher of messages and periodically send sensor data to all subscribers. Using the data sending buffer mechanism can achieve connectionless data transmission. When the publisher discovers that there are no subscribers, it will stop sending messages to save data bandwidth.

In addition to the publisher-subscriber model, there is also a request-service mechanism. The requestor of the service will periodically send messages to the service provider, who will execute the relevant operation according to the established callback function after receiving the request. The request-service mechanism does not require periodic execution and is suitable for operations that are not convenient for high-frequency running, such as publishing large-scale map data. Due to its high computing power consumption, this mechanism is suitable for such applications.

On top of the communication middleware, this article has self-developed basic encapsulation modules based on ROS. The purpose of these modules is to further encapsulate the interface of the communication middleware, encapsulate the functions of data sending and receiving in classes, handle communication buffers, data synchronization, data exceptions, etc. by themselves, and convert the data types used for communication in ROS to the data format defined in the algorithm.

The encapsulation modules can reduce the cost of using middleware for data input and output in upper-layer software, allowing them to focus on the development of algorithm logic.

2.4. Application Software and Algorithm Nodes

The codes related to the platform application for both application software and algorithm nodes have been highly abstracted from the underlying simulator and simulation data. Both of them mainly focus on the implementation of algorithms and logical functional aspects.

The difference lies in the fact that the application software needs to execute a certain task related to the simulator, such as receiving simulation data and performing post-processing, or controlling the movement of a certain agent in the simulator, and the relevant code is still directly related to the simulator or ROS middleware. On the other hand, the algorithm node only concerns data format and content, and does not distinguish between real vehicle test data and simulation experiment data at the algorithm node layer. Although there may be differences in the data format between real vehicle and simulation, such as the scale factor of AVM images, only different configuration files need to be called to adapt to different data, which enhances the adaptability and flexibility of the algorithm.

The application software layer completely decouples data from algorithms, and related algorithms such as mapping, detection, calibration, planning, etc. can be directly deployed in the algorithm node

layer, each as an independent process to complete the simulation experiment task together. The output and intermediate results of the algorithm node will be displayed through a visual interface.

2.5. Data Visualization and User Interface

In algorithm development, data visualization is also a crucial aspect. In semantic mapping and motion planning, sensor data, intermediate results from algorithms, and system outputs all require visualization methods for qualitative or quantitative analysis.

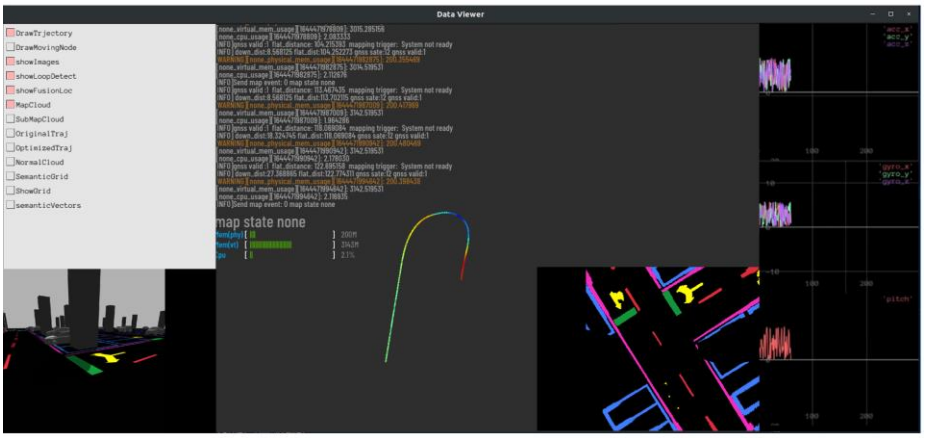

Figure 4: Data visualization and interactive interface

There are already some visualization tools available for use with ROS, such as RVIZ [6], which can directly display data in ROS format through message subscription. However, for some intermediate results of semantic mapping systems, there is a lack of ROS data formats that can be directly applied. This necessitates the design of specialized visualization tools.

At the top level of the platform is the visualization and user interaction tool. In this article, a corresponding graphical interface was designed based on third-party libraries QT and Pangolin, which allows users to view simulation data and intermediate algorithm results, as shown in Figures 4 and 5. In addition to data visualization, users can manually control the movement of the simulation vehicle through clicking interface buttons, allowing for flexible data acquisition.

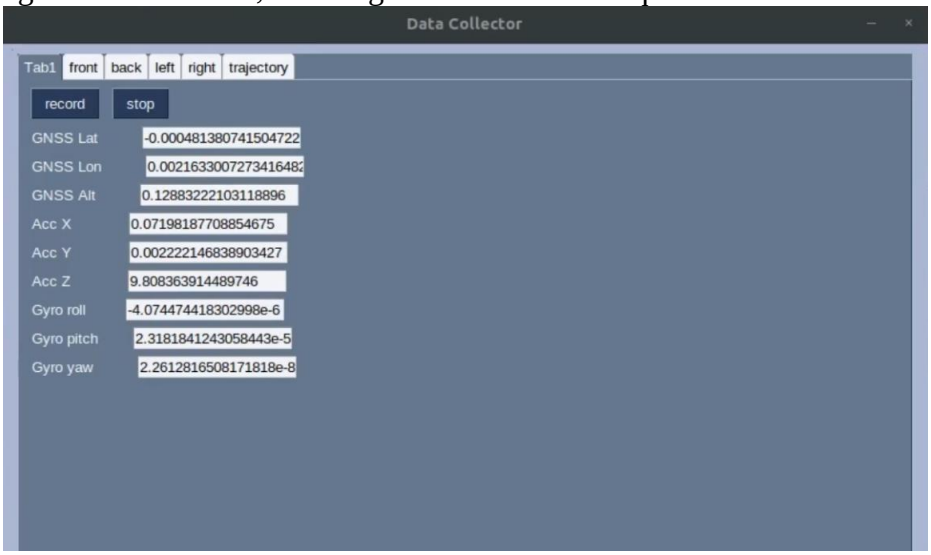

Figure 5: Data collection and visualization tools

3. Platform Applications

Based on the rapid simulation development platform, various simulation experiments can be conducted. This section will introduce the mapping and planning simulation experiments based on the rapid development platform.

3.1. Semantic Mapping

In this section, we present a semantic-based map building system that includes image preprocessing, front-end, back-end, loop detection, and map optimization modules [7]. The mapping system utilizes semantic segmentation and semantic point cloud as the consistent data formats, making full use of perception information to construct highly consistent maps. The front-end utilizes integration on Lie groups to perform trajectory estimation, which is more accurate and effective than Euler integration, and keyframes are selected based on the trajectory estimation results. The loop detection module employs a two-stage detection algorithm based on semantic instances. The algorithm accurately identifies loop frames and generates loop constraints by computing the relative frame poses through up-sampled semantic point clouds. Loop detection and pose estimation can also be used for initialization in relocation tasks. The back-end solves the pose graph with inter-frame constraints, effectively eliminating some accumulated errors and improving the consistency of the semantic sub-map. The map optimization module obtains an optimized semantic map through semantic grid, completing the map building task.

This experiment verifies the semantic graph building algorithm using several simulation environments built by the rapid development platform. The trajectory of single vehicle mapping contains multiple semantic elements involving multiple loops. Due to the influence of sensor noise, there may be certain errors between the trajectory obtained by trajectory estimation algorithm and the true value. These errors can be eliminated by using optimization algorithms. Different levels of sensor noise were set up in the single-vehicle mapping experiments to simulate the real scenario, and the mapping results are shown in Figure 6.

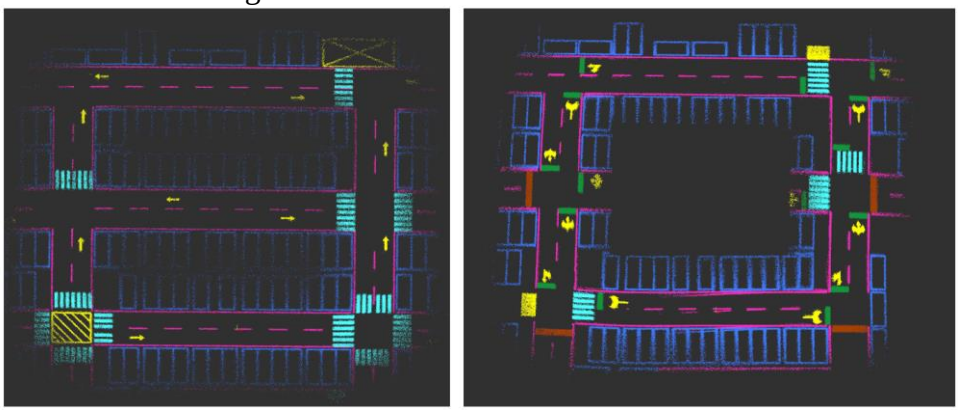

Figure 6: Single-vehicle semantic mapping experiment

In addition to single-vehicle mapping, this paper also conducted simulation experiments on multi-vehicle collaborative mapping, as shown in Figure 7. Multi-vehicle collaborative mapping, also known as crowdsourced mapping, refers to the use of vehicles equipped with low-cost mass-produced sensors to replace the original high-cost, high-precision data acquisition vehicles for mapping tasks.

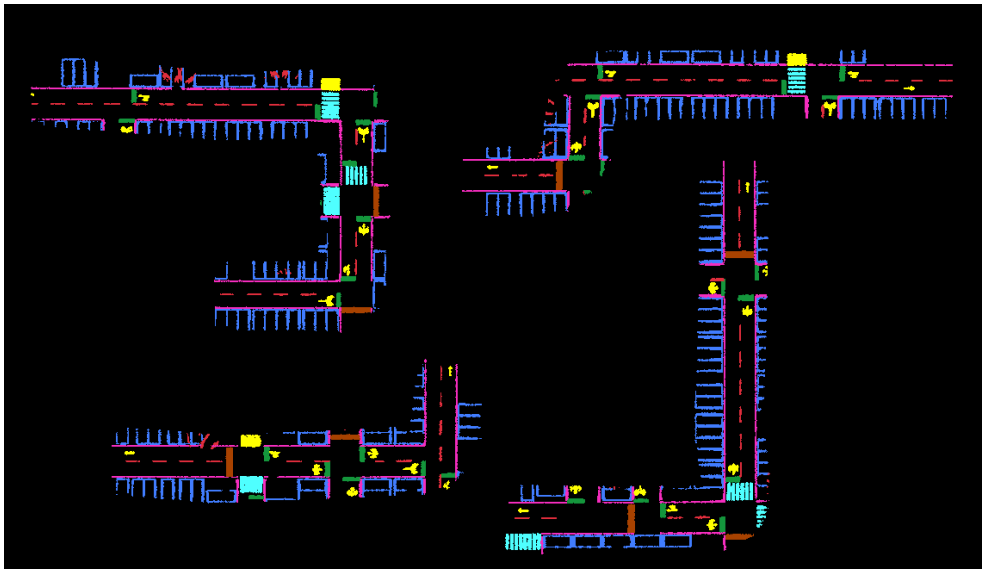

Figure 7: Multi-vehicle collaborative mapping experiment

3.2. Motion Planning

In order to perform local trajectory planning for structured roads, the Lattice Planner [8] based on the Frenet coordinate system [9] is used in this article. This approach is more suitable for high-speed scenarios, has low complexity, is easy to debug, and can meet the needs of mass production. Therefore, the main algorithm used for planning nodes in this platform is the Lattice Planner, whose process is as follows:

(1) Convert the current vehicle pose information to the Frenet coordinate system to obtain the initial state of the vehicle in the Frenet coordinate system. The equations for converting the parameters in the Cartesian coordinate system to the parameters in the Frenet coordinate system are as follows:

$$\begin{cases} s = s_r \\ \dot{s} = \dfrac{v_x \cos\Delta\theta}{1-\kappa_r l} \\ \ddot{s} = \dfrac{a_x \cos\Delta\theta - \dot{s}^2\left[l'\left(\kappa_x \dfrac{1-\kappa_r l}{\cos\Delta\theta} - \kappa_r\right) - (\kappa_r' l + \kappa_r l')\right]}{1-\kappa_r l} \\ l = sign((y_x - y_r)\cos\theta_r - (x_x - x_r)\sin\theta_r)\sqrt{(x_x - x_r)^2 + (y_x - y_r)^2} \\ l' = (1-\kappa_r l)\tan\Delta\theta \\ l'' = -(\kappa_r' l + \kappa_r l')\tan\Delta\theta + \dfrac{1-\kappa_r l}{\cos^2\Delta\theta}\left(\kappa_x \dfrac{1-\kappa_r l}{\cos\Delta\theta} - \kappa_r\right) \end{cases} \tag{1}$$

(2) Calculate the lookahead distance based on the current velocity and obtain the target state of the vehicle at the lookahead point in the Frenet coordinate system.

(3) Sample the trajectory state, including the trajectory time t, target velocity v, and lateral displacement d with respect to the reference line. These three planning parameters can be used to obtain the sampled state.

(4) Construct the polynomial planning functions s(t) and d(s) for lateral and longitudinal

displacements, respectively. In order to ensure that the output trajectory can be executed by the vehicle controller smoothly, the high-order polynomial used to describe the vehicle motion trajectory must satisfy two requirements: the first derivative of curvature with respect to time must be continuous, and the first derivative of vehicle running acceleration with respect to time must also be continuous [10]. Using a third-order polynomial to generate a trajectory has a certain degree of smoothness, which can ensure the continuity of position and velocity, but cannot specify the boundary conditions of acceleration. In addition, the degree of smoothness will also be affected by kinematics and inertial loads. If a trajectory with continuous acceleration is required, it is necessary to determine the initial and final states of position, velocity, and acceleration, which will establish six boundary conditions. At least a fifth-order polynomial is required to describe the trajectory planning. In some specific scenarios, it may be necessary to define higher-order polynomials to increase trajectory smoothness, but as the order increases, the computation time of trajectory planning will also increase, which will affect the overall efficiency of the algorithm. Therefore, when the initial and final motion states of the autonomous driving vehicle are given, using a fifth-order polynomial to describe the motion process is the most cost-effective solution for the first derivative of acceleration with respect to time [11]. The fifth-order polynomial function is represented as:

$$\begin{cases} s(t) = c_1 t^5 + c_2 t^4 + c_3 t^3 + c_4 t^2 + c_5 t + c_6 \\ v(t) = 5c_1 t^4 + 4c_2 t^3 + 3c_3 t^2 + 2c_4 t + c_5 \\ a(t) = 20c_1 t^3 + 12c_2 t^2 + 6c_3 t + 2c_4 \end{cases} \tag{2}$$

(5) After obtaining the planning functions for lateral and longitudinal displacements, time interpolation is performed to obtain the trajectory points in the Frenet coordinate system with respect to the reference line. Finally, the trajectory points are transformed from the Frenet coordinate system to the Cartesian coordinate system to obtain the sampled trajectory in the physical world. Since both the lateral and longitudinal displacements are obtained through high-order polynomial interpolation, the resulting trajectory in the Cartesian coordinate system is also smooth. The equations for converting parameters in the Frenet coordinate system to parameters in the Cartesian coordinate system are as follows:

$$\begin{cases} x_x = x_r - l\sin\theta_r \\ y_x = y_r + l\cos\theta_r \\ \theta_x = \arctan(\dfrac{l'}{1-\kappa_r l}) + \theta_r \\ v_x = \sqrt{[\dot{s}(1-\kappa_r l)]^2 + (\dot{s}l')^2} \\ a_x = \ddot{s}\dfrac{1-\kappa_r l}{\cos\Delta\theta} + \dfrac{\dot{s}^2}{\cos\Delta\theta}[(1-\kappa_r l)\tan\Delta\theta\Delta\theta' - (\kappa_r' l + \kappa_r l')] \\ \kappa_x = \left((l'' + (\kappa_r' l + \kappa_r l')\tan\Delta\theta)\dfrac{\cos^2\Delta\theta}{1-\kappa_r l} + \kappa_r \right)\dfrac{\cos\Delta\theta}{1-\kappa_r l} \end{cases} \tag{3}$$

(6) The sampled trajectory is then subject to collision detection, curvature constraints, and optimal trajectory scoring. The sampled trajectory is a series of smooth trajectories that satisfy the velocity constraints, but it still needs to meet the strong constraints of collision avoidance and vehicle kinematic curvature, as well as the cost constraints of staying away from obstacles and approaching the reference line, etc. The purpose of scoring the sampled trajectory is to obtain an optimal, smooth trajectory that satisfies all the constraints. This trajectory is also the one output by the Lattice planner

to the controller for the vehicle to follow.

Finally, in this section, all motion planning algorithm nodes were simultaneously run to test the entire closed-loop system, and the running conditions were observed. We conducted motion planning experiments on the CARLA TOWN02 map by randomly generating the starting position of the vehicle and manually setting the planning endpoint. Then we ran simulations to verify the effectiveness of the system.

By selecting the endpoint of the planning task, the global planning node first searched for the global path and used it as reference line information. The local trajectory planning node generated a collision-free and smooth driving trajectory based on this information. Finally, the ROS RVIZ toolkit was used to visualize the vehicle, obstacles, and trajectory. In the simulation experiments on the TOWN02 map, several typical scenarios were captured to demonstrate the effectiveness of the algorithm, as shown in Figures 8-10.

Figure 8: Lane-changing condition

Figure 9: Following condition

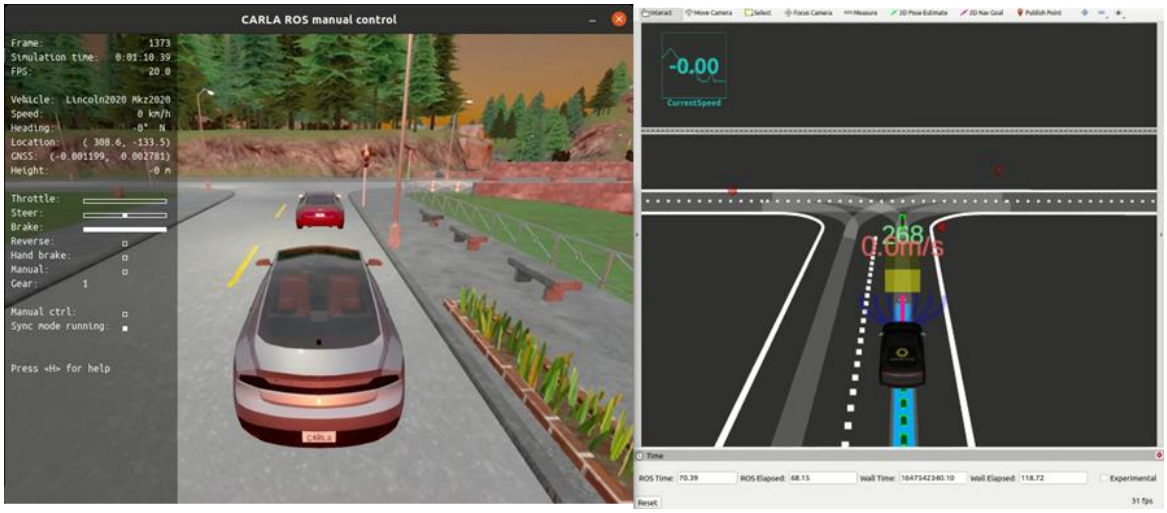

Figure 10: Braking condition

4. Conclusion

The paper presents the setup of a rapid development platform based on CARLA simulator, covering various aspects such as platform architecture, simulation environment settings, and platform modules. The platform has been successfully tested through simulation for tasks such as semantic mapping and motion planning.

References

[1] Qin T, Chen T, Chen Y, et al. Avp-slam: Semantic visual mapping and localization for autonomous vehicles in the parking lot[C]//2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020: 5939-5945.

[2] Koenig N, Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator[C]//2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566). IEEE, 2004, 3: 2149-2154.

[3] Dosovitskiy A, Ros G, Codevilla F, et al. CARLA: An open urban driving simulator[C]//Conference on robot learning. PMLR, 2017: 1-16.

[4] Shah S, Dey D, Lovett C, et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles[C]//Field and service robotics. Springer, Cham, 2018: 621-635.

[5] Geiger A, Lenz P, Urtasun R. Are we ready for autonomous driving? The kitti vision benchmark suite[C]//2012 IEEE conference on computer vision and pattern recognition. IEEE, 2012: 3354-3361.

[6] Kam H R, Lee S H, Park T, et al. Rviz: a toolkit for real domain data visualization[J]. Telecommunication Systems, 2015, 60: 337-345.

[7] Gao X, Zhang T, Liu Y, et al. 14 lectures on visual SLAM: from theory to practice[J]. Publishing House of Electronics Industry, Beijing, 2017.

[8] Fan H, Zhu F, Liu C, et al. Baidu apollo em motion planner[J]. arXiv preprint arXiv: 1807. 08048, 2018.

[9] Werling M, Ziegler J, Kammel S, et al. Optimal trajectory generation for dynamic street scenarios in a frenet frame[C]//2010 IEEE International Conference on Robotics and Automation. IEEE, 2010: 987-993.

[10] Xu W, Wei J, Dolan J M, et al. A real-time motion planner with trajectory optimization for autonomous vehicles [C] //2012 IEEE International Conference on Robotics and Automation. IEEE, 2012: 2061-2067.

[11] Takahashi A, Hongo T, Ninomiya Y, et al. Local path planning and motion control for agv in positioning [C] //Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems'. (IROS'89)'The Autonomous Mobile Robots and Its Applications. IEEE, 1989: 392-397.