Cooperative Autonomous Driving in Simulation

Gonçalo Costa, José Cecílio, António Casimiro
LASIGE, Departamento de Informática, Faculdade de Ciências da Universidade Lisboa, Campo Grande 016, 1749-016 Lisboa; email: fc53352@alunos.ciencias.ulisboa.pt,{jmcecilio, casim}@ciencias.ulisboa.pt

Abstract

Autonomous driving is an area that has been growing in recent years. However, cars are unprepared to cooperate with others nearby, wasting resources and computational power. Thus, cooperative autonomous driving reveals its importance in the future. In this work-in-progress paper, we define, implement and test an architecture for a simulation environment where cooperative autonomous driving protocols can be tested. Additionally, a Manoeuvre Negotiation Protocol is implemented. This protocol will make an existing autonomous driving (AD) stack more resilient in real driving scenarios, improving its robustness and safety.

Keywords: Autonomous Driving stack, Simulator, Manoeuvre Negotiation Protocol, SVL, Apollo

1 Introduction

With the increase in the number of vehicles present on the roads [1], it became necessary to carry out research on autonomous driving, not only to allow better traffic management on the road and thus reduce travel time but also to reduce the number of accidents and, consequently, the amount of money dispended for health recovering [2].

Five levels of automation were considered to achieve a reliable autonomous driving system. At level 1, most components are controlled by the human driver, and only some essential functions are automated (computer controlled). In contrast, at level 5, the vehicle is fully autonomous, with all controls being automated, without any human driver intervention or dependence [2, 3, 4]. Currently, the highest level of automation used in commercialized vehicles is between level 2 and level 2.5, in which vehicles are autonomous up to a certain point, requiring human intervention in adverse or extreme situations [3].

Vehicle to Everything (V2X) is a concept that is very common in the area of autonomous driving and refers to communication between vehicles (V2V) and communication with infrastructures (V2I). The V2X system is crucial for identifying and analyzing obstacles and phenomena, which are then communicated to other vehicles and infrastructures. This way, drivers can learn information without observing it in person, which improves driving safety [5]. Subir et al. [6] reinforce using V2V and V2I communication, addressing the topic of Cooperative Collision Avoidance. This article indicates the advantages of using broadcast messages instead of performing a specific routing.

However, over these years of evolution towards autonomous driving, there has been a significant increase in the number of sensors and cameras in vehicles, increasing the number of cables and data-buses that are crucial for passing and sharing data. Furthermore, the code required for all these components to work correctly is extensive and increasingly complex, which can put reliability at risk. Finally, the increasing technology in vehicles requires more computational power [3] since they comprise a lot of sensors that generate a considerable amount of data to be processed quickly.

This approach of putting as much technology as possible into vehicles refers to scenarios where each vehicle reacts to the surrounding environment. Despite being a viable approach for the present, we must consider that in a future where all vehicles are considered autonomous, there will be a considerable waste of computation since all the vehicles will be performing the same processing of the environment surrounding them [3].

The advantages of cooperative autonomous driving systems have already been demonstrated and discussed in [6]. However, its implementation is not addressed. One of the main difficulties is the costs associated with installing and testing those systems in actual vehicles, in which the prices of cars and infrastructure are too high [6, 7, 8, 9, 10]. Another difficulty associated with the implementation is the high number of hours that would have to be devoted to testing the algorithms on the roads, with algorithms that require hundreds of hours of testing in different conditions (e.g., atmospheric conditions). This makes the process very complicated to carry out [7, 8, 9].

Considering all difficulties, the one that makes the entire implementation process very difficult refers to the safety conditions of humans, in which many tests involve pedestrians and, in general, all tests carried out on the road can compromise the life of any pedestrian who is present in the vicinity [7, 9, 10].

According to the ISO/PAS 21448:2019 [11] standard, several security measures were created to guarantee safety conditions while testing autonomous driving algorithms and protocols. Thus, using simulators of real environments to test algorithms and protocols became necessary since we can test any condition without creating dangerous situations. The main advantage of simulators is the ability to change reality for the different tests that the algorithms need to be trained and evaluated [7, 9, 10]. The work in [9] used the CARLA simulator to train a driving policy through Reinforcement Learning to test it in the real world later. As such, they recreated the route in the simulator and trained the system according to the

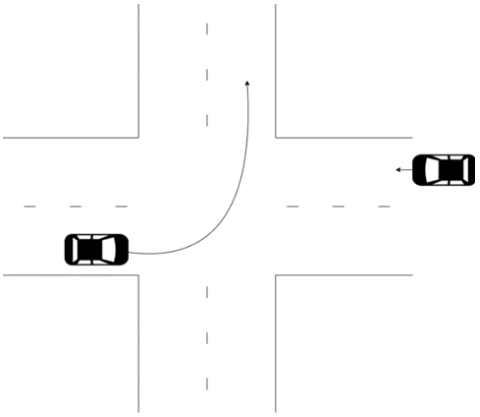

Figure 1: Vehicle performing a manoeuvre at an intersection

real-world scenario. The work done in [12] also presents a protocol for coordination between vehicles and implements it using the V-REP simulator. The author then carried out some tests on the protocol, namely implementing purposeful communication failures, to verify the protocol's robustness. All these works indicate methods to test algorithms and protocols, maintaining the security of people and infrastructures.

In this work, we will explore the advantages of cooperative driving systems, where vehicles can cooperate with each other in the surrounding environment. We will use a simulator to implement and test a cooperative protocol for manoeuvre interception. The protocol was designed to guarantee greater safety conditions for each vehicle while reducing the construction complexity and code necessary for correct operations [13]. This work is also motivated by the need to test the protocol in different scenarios to observe its behaviour. Using failure scenarios to analyze their impact on the protocol is crucial. It is also crucial to consider real-world factors such as message losses, latency variations and malicious behaviours that can influence the autonomous driving system [3, 4, 6, 7, 9, 10].

In this work, a realistic simulator (SVL [14]) is used. It allows the simulation of a driving scenario in its entirety, considering other vehicles, pedestrians, traffic signs and traffic rules. Furthermore, each vehicle corresponds to an instance of the Apollo autonomous driving stack [15], along with the protocol proposed in [13]. A network simulator is also integrated to recreate real conditions where failures can be introduced.

2 Autonomous Driving Protocol

The Manoeuvre Negotiation Protocol used [13] is based on a solution where vehicles have different priorities, depending on the manoeuvre they intend to perform. However, to avoid starvation, a lower-priority vehicle can increase its priority and carry out the manoeuvre it wants to perform. Suppose the time to complete the manoeuvre is less than the time that the higher-priority vehicle takes to reach the intersection. In that case, the vehicle can increase its priority and make the intersection, as shown in Figure 1.

The protocol was designed to handle specific conditions, like Priority Violation. It is considered that there has been a Priority Violation when a vehicle that has increased its priority cannot complete the intersection, causing vehicles with higher priorities to be unable to perform normal traffic behaviour. To prevent this, the protocol predicts the time for the vehicle to reach the intersection and pass the intersection. If the time interval of the passing vehicle intersects the time interval of the vehicle present on the road of the intersection, the protocol will not allow the vehicle to cross the intersection.

Each vehicle (p_i) also has a membership that indicates to other vehicles, with higher priority, that it intends to perform a manoeuvre.

The protocol updates the membership every T_M unit of time. It is also responsible for calculating which vehicles have higher priority than p_i and which can reach the intersection during the execution of the manoeuvre. This way, three entrances are created for the three possible manoeuvres (go straight ahead, turn left and turn right). Timestamps are used to check the correctness of the membership to guarantee security. Finally, the value of Manoeuvre Opportunity (*MO*) is determined, which only becomes *True* if all vehicles with the highest priority are within the communication range. The vehicle is in an admissible crossing opportunity if the membership is empty and fresh.

This protocol is also responsible for describing the states and messages that vehicles send when carrying out a manoeuvre and scenarios where messages may not reach the intended destination due to network failures. Before any vehicle wants to carry out a manoeuvre, it must connect to the server and store its information (e.g., location). Then, this information is used to create the membership. When a vehicle intends to perform a manoeuvre, it invokes the *tryManoeuvre* procedure, which generates a request tag, including the timestamp when the request was made, the requester ID, and the manoeuvre to be performed.

Next, the vehicle checks its state, and if it is in the *NORMAL* (no manoeuvre is being performed) or *TRYGET* state (agent intends to execute a manoeuvre), it invokes the Membership Algorithm (*MA*). If the vehicle can perform the grant request after invoking the *MA*, it sends it and starts a timer named *tRETRY*. If the vehicle receives all responses and all come with the *GRANT* message (sent when the agent accepts the required manoeuvre), the vehicle changes its state to *EXECUTE* and executes the manoeuvre. Otherwise, it sends a *RELEASE* message (sent when the vehicle intends to revoke the GRANT) to all agents, changes its state to *TRYGET* and starts the *tRETRY* command for a new round, avoiding a deadlock situation. If the vehicle does not receive all the responses within the *tRETRY* time, the vehicle changes its state to *TRYGET* and executes the *tryManoeuvre* procedure again.

In a scenario where the vehicle is in the *GRANT* state (vehicle gives in to another to perform a manoeuvre) but wants to perform a manoeuvre, it switches to the *GRANTGET* state (vehicle gives a GRANT message but intends to perform a manoeuvre) and will invoke the *tryManoeuvre* procedure when the grant timer ends.

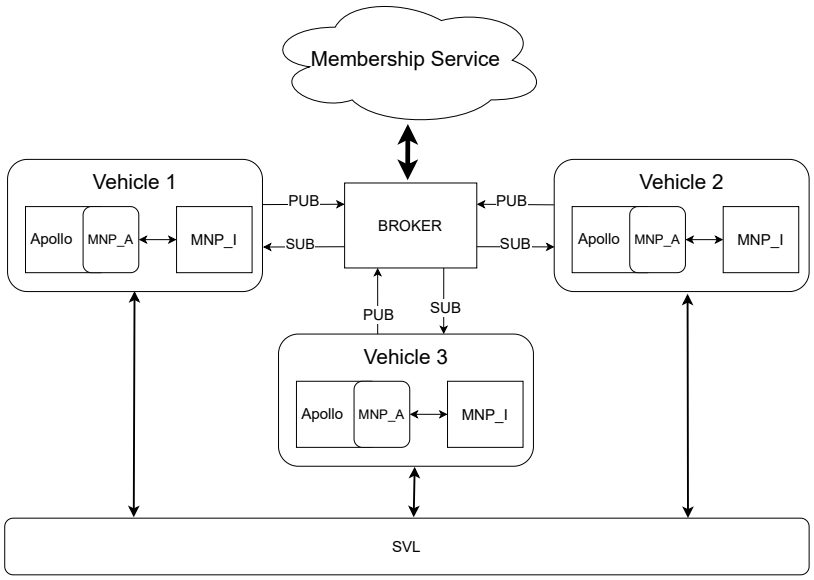

Figure 2: Architecture of the simulator

This protocol also covers scenarios where communication can fail. If the *RELEASE* message does not reach the vehicles, they check the last position of the vehicle to whom the *GRANT* message was sent and inferred, using sensors, whether it is outside the intersection or not. The protocol can also discard messages sent in previous rounds to avoid interfering with the current round.

3 Autonomous Driving Simulator

Regarding the choice of the simulator, it was decided to choose the SVL simulator [14] because it is open source, largely customizable and has a realistic graphics engine that can be easily extended. In addition, the SVL allows the creation of modules that can be used to establish communication between vehicles, being crucial for the work. Regarding the simulator's architecture, SVL has a section responsible for loading the map, vehicles, sensors and environmental settings. The SVL is then responsible for sending the results obtained by the sensors to the AD Stack. The Autonomous Driving (AD) Stack, after receiving inputs from the SVL, is responsible for applying the protocol and updating the vehicle status in the SVL. This update corresponds to the manoeuvre or strategic route changes by the vehicle. There is also a Visual section that receives the output of the SVL, corresponding to the vehicle circulating on the road according to the actuators provided by the AD stack.

Regarding the AD stack, the Apollo stack is used since it is easily incorporated with the SVL, which makes the implementation of the protocol much easier. Apollo runs inside a docker container, making it possible to create multiple instances of Apollo, each corresponding to an autonomous vehicle.

Figure 2 represents the system architecture implemented in this work. It comprises modules corresponding to the Manoeuvre Negotiation Protocol, Apollo AD stack and the network simulator used to carry out communication tests between the different vehicles.

In our architecture, each vehicle comprises an Apollo AD stack, a Manoeuvre Negotiation Protocol Agent (MNP_A) and a Manoeuvre Negotiation Protocol Instance (MNP_I). The MNP_A fetches data from the Apollo AD stack and sends it to MNP_I. Then, MNP_I updates the membership, applies the protocol and returns the protocol commands to the Apollo AD stack through the MNP_A. All the MNP_I and the membership are connected to a broker. This broker implements pub/sub mechanisms to support the integration of multiple vehicles in a seamless way. All vehicles registered on the broker are able to use the protocol since the messages exchanged are broadcasted to a specific topic that everyone is subscribed to. The protocol also defines specific message types to update the membership. Access to these topics is restricted to the Membership Service.

Lastly, all the vehicles are connected to the SVL simulator to represent their status.

4 Implementation of the Simulator

Implementing the architecture defined in Figure 2, requires docker since the Apollo AD stack runs inside a docker container. In our implementation, there are two containers per vehicle: one that supports and runs Apollo and another container responsible for the protocol's operation. In each Apollo instance, the Planning and Control modules connect to the MNP_A module to receive information about the vehicle, such as speed, position, acceleration and trajectories.

The connection between MNP_A and MNP_I is made by sending UDP messages. Then, MNP_I uses MQTT topics to communicate with the broker and other vehicles to create safe manoeuvres. Each MQTT message includes specific information concerning the intention of each vehicle. For instance, if a vehicle intends to start a manoeuvre, it must publish a message with the vehicle identification (vehicle ID), a timestamp, a manoeuvre code, and the trajectory.

All instances are subscribed to the same topic. The vehicles filter each message, and if it corresponds to a message sent

by itself, it is discarded. If the vehicle ID is different, the message is processed by the MNP_I module, allowing it to change the values of acceleration, velocity and trajectory of the vehicle. If necessary, it sends updated information to the MNP_A module, which will update the information inside the Apollo container by changing the information in the Control module and completing the execution flow.

Finally, the communication between Apollo and SVL simulator is done through a network bridge. If the entire simulation is performed on one machine, this bridge refers to the local host. Otherwise, the bridge refers to the IPv4 address of the machines running the Apollo stack. In this way, SVL supports multiple systems connected simultaneously.

The entire architecture can be implemented in a single machine. However, it requires a machine with a considerable amount of graphics and processing power to process the sensor results from the AD stack and to render the world in which the cars are. In addition, in the implementation carried out in this work, multiple autonomous vehicles will be used. They will receive and process data, increasing the requirements for a machine to process the simulation. Thus, it is important to spread the processing across several machines when needed.

5 Conclusion

This work proposes an architecture to develop a simulator for testing cooperative autonomous driving protocols. The solution comprises integrating an autonomous driving stack and the SVL simulator that represents the environment and the physical conditions/status of the sensors presented in the vehicles, as well as integrating a cooperative autonomous driving protocol. In the first prototype developed, two vehicles were considered and we concluded that the simulator requires a machine with a considerable amount of graphics and processing power, which may suggest spreading the processing across several machines if more vehicles are considered. More experiments and implementations are needed to evaluate the performance of the simulator and the efficiency of the protocol. Additionally, scenarios where large latencies and losses in communication must be evaluated, as well as when the protocol is presented to a malicious vehicle.

Acknowledgments

This work was supported by the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

References

[1] J. Dargay and D. Gately, "Income's effect on car and vehicle ownership, worldwide: 1960–2015," *Transportation Research Part A: Policy and Practice*, vol. 33, no. 2, pp. 101–138, 1999.

[2] S. Mariani, G. Cabri, and F. Zambonelli, "Coordination of autonomous vehicles: taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–33, 2021.

[3] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1243–1274, 2018.

[4] I. Yaqoob, L. U. Khan, S. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2019.

[5] A. Abacus, "Vehicle-to-everything (V2X) communication – the design engineer's guide." `https://www.avnet.com/wps/portal/abacus/solutions/markets/automotive-and-transportation/automotive/communications-and-connectivity/v2x-communication/`, 2023. [Online; accessed 7-February-2023].

[6] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *IEEE communications magazine*, vol. 44, no. 1, pp. 74–82, 2006.

[7] J. Seymour, Q.-H. Luu, *et al.*, "An empirical testing of autonomous vehicle simulator system for urban driving," in *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp. 111–117, IEEE, 2021.

[8] D. Zhao, Y. Liu, C. Zhang, and Y. Li, "Autonomous driving simulation for unmanned vehicles," in *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 185–190, IEEE, 2015.

[9] B. Osiński, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6411–6418, IEEE, 2020.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

[11] ISO, "ISO/PAS 21448:2019(en) Road vehicles — Safety of the intended functionality." `https://img.auto-testing.net/testingimg/202003/19/071723321.pdf`, 2023. [Online; accessed 7-February-2023].

[12] J. P. V. Pinto *et al.*, *Design and implementation of a protocol for safe cooperation of self-driving cars*. PhD thesis, 2019.

[13] A. Casimiro, E. Ekenstedt, and E. M. Schiller, "Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems," *arXiv preprint arXiv:1906.04703*, 2019.

[14] SVL, "Introduction." `https://www.svlsimulator.com/docs/getting-started/introduction/`, 2022. [Online; accessed 9-December-2022].

[15] Apollo, "Apollo Platform." `https://developer.apollo.auto/developer.html`, 2022. [Online; accessed 12-December-2022].