POLITECNICO DI TORINO
Repository ISTITUZIONALE

A Full-fledge Simulation Framework for the Assessment of Connected Cars

Original
A Full-fledge Simulation Framework for the Assessment of Connected Cars / Selvaraj, DINESH CYRIL; Hegde, Shailesh; Chiasserini, Carla Fabiana; Amati, Nicola; Deflorio, Francesco; Zennaro, Giuliana. - In: TRANSPORTATION RESEARCH PROCEDIA. - ISSN 2352-1465. - STAMPA. - 52:(2021), pp. 315-322. (Intervento presentato al convegno 23rd EURO Working Group on Transportation Meeting, EWGT tenutosi a Paphos, Cyprus nel 16-18 September 2020) [10.1016/j.trpro.2021.01.037].

Availability:
This version is available at: 11583/2847362 since: 2023-10-10T13:40:21Z

Publisher:
Elsevier

Published
DOI:10.1016/j.trpro.2021.01.037

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

Available online at www.sciencedirect.com

ScienceDirect

Transportation Research Procedia 00 (2020) 000–000

Transportation Research Procedia

www.elsevier.com/locate/procedia

23rd EURO Working Group on Transportation Meeting, EWGT 2020, 16-18 September 2020, Paphos, Cyprus

A Full-fledge Simulation Framework for the Assessment of Connected Cars

Dinesh Cyril Selvaraj[a], Shailesh Hegde[a], Carla Fabiana Chiasserini[a*], Nicola Amati[a], Francesco Deflorio[a], Giuliana Zennaro[b]

[a] *CARS@Polito, Politecnico di Torino, Torino, Italy*

[b] *FCA-CRF, Torino, Italy*

Abstract

Intelligent Transport Systems (ITS) have emerged as an integral part of smart cities, providing increased ease of mobility as well as efficiency and safety in vehicular traffic. Given its wide array of applications, ITS has also become a multidisciplinary field of work where vehicular communications, traffic control, ADAS (Advance Driver Assistance System) sensors, and vehicle dynamics have all to be accounted for. The study of such diverse aspects makes the evaluation of new ITS approaches, algorithms, and protocols not a small feat. For this reason, the availability of an effective, scalable, and comprehensive tool for the investigation and virtual validation of new ITS solutions is paramount. In this work, we present a simulation framework, called CoMoVe (Communication, Mobility, Vehicle dynamics), that effectively addresses the above need, as it enables the virtual validation of innovative solutions for vehicles that are both connected and equipped with ADAS sensors. Our framework encapsulates the important attributes of vehicle communication, road traffic, and dynamics into a single environment, by combining the strengths of different simulators. CoMoVe finds its use to evaluate the impact of vehicle connectivity, while imposing causality on vehicle dynamics and mobility. Such an assessment can greatly facilitate the development of control systems, algorithms, and protocols for real-world ITS.

© 2020 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 23rd EURO Working Group on Transportation Meeting..

Keywords: Connected vehicles; vehicle dynamics; simulation framework; traffic efficiency; safety services

1. Introduction

Intelligent Transport Systems (ITS) are an important component of Smart Cities, aiming at converting cities into digital societies and making the life of their citizens easier in every possible way. ITS applications range from convenience services, such as advanced traveller information and transport pricing systems, to safety applications like collision avoidance at intersections and vulnerable users protection. All of such applications may benefit from vehicle-

* Corresponding author.
This work was partially supported through the CRF-CARS@Polito SAVE project.

2352-1465 © 2020 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 23rd EURO Working Group on Transportation Meeting..

to-everything (V2X) communications, i.e., vehicle to infrastructure (V2I), vehicle to vehicle (V2V), or vehicle to network (V2N), or vehicle-to-device (V2D), as specified by the cellular as well as the IEEE 802.11p standards [1].

While a large body of research exists on vehicular services and the performance of V2X technologies for their support, fewer works [2] have addressed the broader area of ITS performance assessment, accounting for all the related aspects, from road transportation and communications to vehicle control and dynamics. In this work, we address this problem by presenting a comprehensive and flexible framework for the realistic simulation of all system components. Our framework, called CoMoVe (Communication, Mobility, Vehicle dynamics), integrates different simulators and lets them work together through an efficient set of interfaces and a carefully designed information flow. Importantly, some of the composing blocks could be easily replaced by hardware-in-the-loop components, thus allowing for a virtual validation thereof. Additionally, thanks to the fact that CoMoVe simulates all the main ITS components, it enables the study of very diverse aspects such as driver and passanger comfort and eco-driving.

In particular, our main contributions are as follows:

(i) We developed a simulation framework for the assessment of ITS, accounting for all the most relevant aspects, namely, wireless connectivity, vehicles interactions along roads, vehicle dynamic and control, and ADAS sensors.
(ii) Our design allows for a flexible use of the framework, with the possible removal/replacement of some simulation blocks, the development of different system and vehicle control strategies, and the implementation of either centralized or distributed control and data transfer schemes.
(iii) To show the framework capabilities and its effectiveness in assessing the performance of existing and future ITS applications, we designed a Collision Avoidance (CA) application leveraging the information transmitted by vehicles, and a Cooperative Adaptive Cruise Control (CACC) system taking as input both the transmitted data and those gathered through vehicle on-board sensors.
(iv) We demonstrated the benefits of leveraging the information transmitted by connected cars in real-world scenarios and traffic conditions, in terms of safety level as well as driver/passenger comfort.

The remainder of this paper is organized as follows. Sec. 2 reviews the related studies and highlights the novelty of our work. Sec. 3 introduces the CoMoVe framework architecture and describes the role of each component. Sec. 4 presents the design of the control strategy we developed to implement the Co-operative Adaptive Cruise Control and the Vehicle Collision Avoidance applications. Finally, Sec. 5 presents the simulation scenarios we considered and the results we obtained, while Sec. 6 concludes the paper and sketches some directions for future research.

2. Related Work

Some frameworks combining different simulation blocks for the study of ITS applications have appeared in [3, 4], with a specific focus on vehicle platooning. Other ADAS applications like CA, Adaptive Cruise Control (ACC), and Collision Warning have been addressed in [5], [6], and [7], respectively. In particular, the framework in [6] focuses on ACC and integrates microscopic traffic simulation and vehicle dynamics by leveraging IPG CarMaker (for ADAS) and Simulation of Urban MObility (SUMO). [7] used instead the open-source simulator CARLA (Car Learning to Act) [8] to validate the autonomous driving functionalities like ADAS, using its built-in traffic models. In our work, we have opted for CarMaker as it provides sophisticated vehicle dynamics models [9, 10] and aid hardware-in-the loop testing for automotive industry applications. Even though CARLA provides a reliable platform, our framework with specialised simulators helps researchers to perform in-depth analysis in their respective domain. However, recent upgrades of CARLA prove it can be a viable alternative to CarMaker in our open-source framework.

The work in [3] integrates SUMO, Simulink, and OMNeT++, to analyse string stability of a platoon by varying time headway and some communication parameters. This framework is developed by converting the Simulink data blocks into corresponding C++ code, for synchronization with the other simulators, at the cost of loosing some key features of the Simulink data blocks. Communication and road traffic are dealt with in [2, 4, 11], where [4, 11] present simulation frameworks for the study of packet reception performance under different traffic densities, while [2] focusses on the evaluation of data management strategies for vehicular networks. Unlike [4, 11], our framework accounts for vehicle dynamics, and, unlike [2], we integrate a detailed model of the communication standards.

In conclusion, with respect to previous work, we account for all of the multidisciplinary aspects of ITS, and design a simulation framework that is accurate, flexible, and able to accommodate very diverse applications. Furthermore, the CoMoVe framework can be used to assess different aspects, from traffic efficiency to safety and passenger comfort.

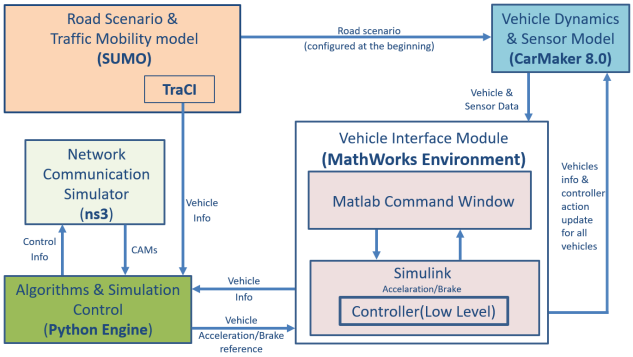

Fig. 1. The CoMoVe framework architecture.

3. The CoMoVe Framework

CoMoVe is a flexible, integrated framework that combines the strengths of multiple simulators with a *Python engine as a middleware* to enable the interoperability between them and accommodate diverse control logics. The framework architecture is shown in Fig. 1, where:

(i) the wireless communication network is simulated through the open-source ns3 network simulator[1], including the network infrastructure and vehicles equipped with On-Board Units (OBU);
(ii) the Python engine uses information provided by the other modules and implements the algorithms/logics for the decisions to be made at system/vehicle level;
(iii) the CarMaker [12], module can simulate the road layout, the vehicle mobility and dynamics, as well as the vehicle on-board sensors;
(iv) the MATLAB/Simulink module acts as an interface between the Python engine and CarMaker, but it can also be used to simulate both the vehicle dynamics and the vehicle on-board sensors, thus fully replacing CarMaker;
(v) the road scenario and vehicle mobility can be represented through the open-source simulator SUMO[2], with the programming interface TraCI (Traffic Control Interface) being used to exchange information between SUMO and the Python engine.

Note that the interaction between the different components of CoMoVe is set to happen every 100 ms, as this is a good trade-off between accuracy and computational complexity. This also determines the time granularity with which the information is collected in the framework. However, other time granularities could be accommodated as well.

In the case where the CarMaker block (which is a proprietary software) and SUMO (which is open source) are both used in the framework, they need to use the same road topology. In this work, we considered both of them, with SUMO being the primary source of scenario and mobility information and CarMaker simulating the ADAS sensors and the vehicle dynamics. In other words, the exact replica of SUMO road topology is configured in the CarMaker environment using its scenario builder.

Since the control strategies are implemented in the Python engine, the latter needs to receive the vehicle information provided by CarMaker and SUMO. The TraCI python library allows us to access the necessary information from SUMO directly, and it also provides full control over the SUMO simulation execution. There is instead no direct way to integrate CarMaker with the Python engine, thus we have used CarMaker for Simulink extension as an interface, to extract the vehicle data from the CarMaker environment. To make Simulink (hence, CarMaker) interact with the Python engine, we have exploited the MATLAB commands along with the MATLAB Engine API Python library.

Once the vehicle data are available to the Python engine, these are combined therein with the data generated and transmitted by the vehicles and simulated through the ns3 network simulator. The Python engine can then leverage the collected information and execute the control algorithms of interest. The algorithm output can be delivered as commands to the network or road infrastructure, or to the vehicles and the actuators (e.g., emergency braking system,

[1] https://www.nsnam.org
[2] https://sumo.dlr.de/docs

or automated acceleration). Depending on the type of entity that has to receive the command, the Python engine will need to interact with the appropriate framework component (e.g., ns3, SUMO, Simulink). A practical example is provided in Sec. 4.

3.1. Vehicle Dynamics, Sensor and Mobility Model (CarMaker & SUMO)

Different tools can be used for modelling the vehicles and their dynamics in automotive research and development. A detailed vehicular model includes the axle kinematics, brakes, steering, suspension, and tires, and has to represent the driving dynamics of the vehicle. IPG CarMaker is one of the tools that provides a complete vehicle model and allows for a full analysis of the vehicle dynamics under different test scenarios. In particular, we used the default Volkswagen Beetle vehicles and changed only the gearbox from manual to automatic. Among the multiple sensors available in CarMaker, we have used Radar with a 150-m coverage, attached at the front of the vehicle. Even though CarMaker has the ability to model the vehicles' mobility, we used SUMO to simulate the vehicles interactions and account for such metrics affecting travel time and capacity of the road. SUMO also offers the TraCI interface, which allows us to access the values of simulated vehicles as well as control the simulation execution. The traffic model of all vehicles is defined by SUMO, then these vehicles are created in the CarMaker environment through a software script. At every simulation step, the TraCI command "convert3D" is used to extract the position of the simulated vehicles. With the help of the software script, the position is transformed into the CarMaker location format and updated in its environment.

3.2. Communication Network (ns3)

All vehicles are equipped with an OBU for V2X communications, which can be implemented using Dedicated Short Range Communications like in the IEEE 802.11p standard, or the cellular LTE technology. Importantly, ETSI has defined two types of messages: (i) the Cooperative Awareness Messages (CAM), i.e., a sort of heartbeat messages periodically sent by each vehicle and carrying information about the sender's position, heading, acceleration, and timestamp [13], and (ii) the Distributed Environmental Notification Messages (DENM), which are triggered on the occurrence of certain events defined by the application and carry information related to the occurred event [13].

Among the existing network simulators (ns3, OMNeT++), we have selected ns3, which allows for all types of V2X communications. For concreteness, we have focused on LTE-based communications between vehicles and network infrastructure, and used the LTE-EPC Network simulator (LENA) to simulate them. Each vehicle thus periodically broadcasts CAMs, which are received by the base station (eNB) covering the vehicle and transferred, through the LTE core network, to a network node hosting a database and a server. The database is used to store the information carried by the CAMs (in particular, acceleration, speed, position, and lane ID). The server instead runs the control logics, namely, an algorithm for vehicle collision detection and adaptive cruise control. If any of the algorithms outputs an alert or a command for a vehicle, such information is inserted into a DENM and delivered by the LTE network to the intended user.

To recreate a real-world urban environment, we have created buildings around the eNB by using the ns3 Buildings module. The buildings module also implements the propagation loss model to be used with the LTE module, accounting also for wall penetration losses. Since the Python engine plays the role of middleware, all the above mentioned ns3 functionalities are scripted in Python with the help of ns3 Python bindings and the simulation script is integrated into the Python engine.

3.3. Vehicle Interface Module (MATLAB/Simulink)

As mentioned, Simulink is used to integrate CarMaker in the framework. To make the simulators communicate with each other and satisfy the CAMs requirements, we have designed a Simulink block to control the execution of the CarMaker simulation, which pauses the execution for every 100 ms. The quantities in the CarMaker environment are manipulated with the help of the "readfromdict" and "writetodict" Simulink library blocks. Additionally, we have explored the usage of the "Adaptive Cruise Control with Sensor Fusion" model provided by MathWorks, to replace the CarMaker simulator with Simulink data blocks.

3.4. Algorithms and Simulation Control (Python Engine)

The main functions of the Python engine are : (i) to connect Simulink (hence, CarMaker if present) with the ns3 simulator, (ii) to host the control strategies, and (iii) to allow the exchange of the vehicle location information between SUMO and CarMaker. Note that the Python engine initializes the simulation through the following four steps. First, it finds the shared MATLAB environment and connects to the same through the MATLAB Engine API. Second, it executes the MATLAB function to create a list of Simulink data blocks that help us to extract the vehicle information from the CarMaker environment. Third, it establishes a connection with SUMO with the help of TraCI commands. Fourth, a list of ns3 nodes are created to represent the vehicles involved in the simulation. Once the initialisation phase is completed and a connection between all the simulators has been successfully created, the Python engine starts the simulation by scheduling an ns3 event to extract vehicle data from CarMaker and SUMO through MATLAB Engine API and TraCI Python libraries. The simulation is primarily controlled by ns3 events, scheduled to run every 100 ms.

4. Control Strategy

In this section, we describe the control logics that we developed within the Python engine, to implement the Cooperative Adaptive Cruise Control (CACC) system and the Collision Avoidance (CA) application. The former is achieved by adjusting the acceleration of a vehicle so as to achieve a desired speed. In the case of non-connected cars, ACC exploits the information received (at the vehicle) from the Radar sensor. In the case of a connected car, instead, the system becomes cooperative (i.e., CACC is applied), as it leverages the instructions received at the vehicle from the network server, which, on its turn, computed the suitable value of acceleration based on the information carried by the received CAMs [14]. As for the CA application, this exploits CAMs only, builds on a trajectory-based prediction system to identify the vehicles on collision course, and computes the acceleration (deceleration) that the vehicles should apply to avoid crashing into each other. Although the control strategy algorithm is implemented in the same way for all the vehicles, for clarity of presentation, below we will refer to a particular vehicle, denoted as ego car, and refer to other vehicles as neighbouring vehicles.

Let us first describe the (C)ACC system. In the case of non-cooperative ACC, i.e., when only the Radar signal is employed, the logic is implemented at the vehicle; on the contrary, when CACC is used, i.e., both Radar and CAMs can be exploited, the logic leveraging the CAMs is implemented at a server in the network infrastructure. In the latter case, the minimum between the Radar-based output and the CAM-based output is selected as final acceleration determined by the CACC system.

In both ACC and CACC, the system may operate in Velocity control mode or in Distance control mode. Velocity control mode is activated when, given an ego car, there are no vehicles within the ego car's safety distance [15]. The suitable acceleration is thus calculated to maintain the desired cruise speed. Distance control mode, instead, is triggered when one or more neighbouring vehicles fall within the ego car's safety range. In the case of multiple neighbouring vehicles, the closest vehicle to the ego car's trajectory is selected as the target vehicle, and the goal is to control the relative distance and velocity between the ego car and the target vehicle.

In Velocity control mode and Distance control mode, the acceleration/deceleration rate of the ego vehicle is calculated through the P controller with k_{v1}, k_{v2}, k_d, k_{dd} as proportional gains [12] (see Fig. 2). The controller takes the desired distance, desired cruise speed, and relative distance (d_s) and velocity (d_v) between ego car and target vehicle as inputs. Gain k_{v1} is associated with d_v and k_d determines the ratio of the output response to the difference between d_s and the desired distance. The gains k_{v2} and k_{dd} are used to regulate the ego car acceleration to maintain the desired cruise speed and desired distance, respectively.

Furthermore, the output acceleration/deceleration rate is restricted to certain limits as reported in [16], in order to ensure a minimum level of comfort and safety for the passengers.

In the case of connected cars, CACC and CA work in a synergic manner, as detailed below. CA uses a trajectory-based prediction system to detect vehicles on collision course; in particular, we have adapted the collision detection system proposed in [5] to match our needs. Through the CAM information, the algorithm predicts the vehicle positions for the next 10 seconds, with 0.1 seconds as a difference between subsequent predicted time instances. Furthermore, to correct possible errors due to GPS or mismatches between the predicted trajectories and the road layout, we exploit the CAM LaneID, RoadID, and LinkID fields to make sure that the predicted vehicle position is placed on a meaningful

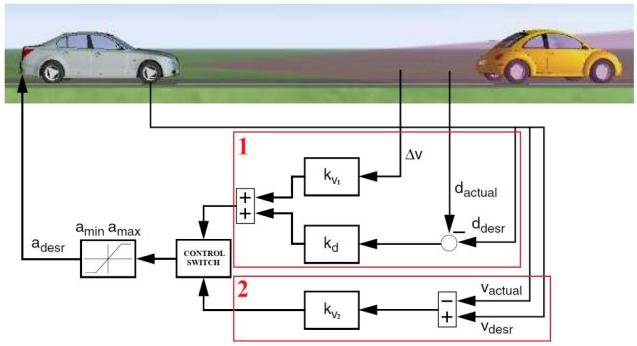

Fig. 2. ACC control scheme [12].

stretch of the road layout. Then we calculate the relative distance (D_p) between the ego car and every neighbouring vehicle. If D_p is below the safety distance threshold at any point in the next 10 seconds, we use CACC to find the desired ego car acceleration and avoid the collision. Note that, in this case, CACC does not use the fixed proportional gains in [16], rather it employs variable proportional gains calculated based on the distance between the ego and the target vehicle. By doing so, the ego car can react to the situation pre-emptively and gradually reduce its speed.

When both the CACC and the CA are in place, the control algorithm chooses the minimum acceleration as the desired ego car acceleration. This value is delivered to the ego car through a DENM message, which is transmitted from the LTE network to the car, so that the latter can accelerate/decelerate accordingly.

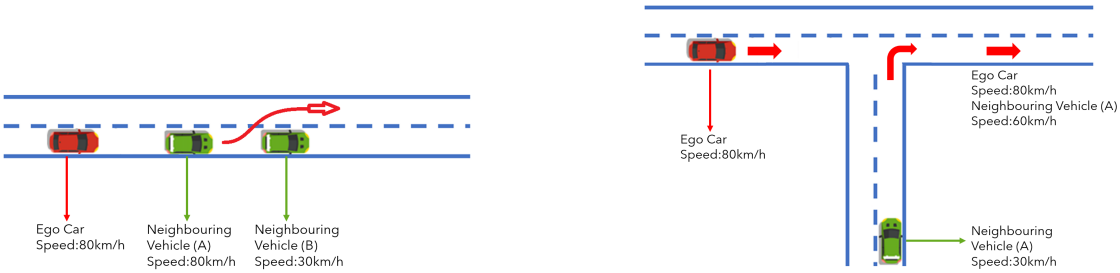

Fig. 3. Cut-Out scenario (left) and T-Junction scenario (right).

5. Performance Results

We now show how CoMoVe can be used to study the impact of connected vehicles on the performance of comfort and safety applications. To this end, we consider two preliminary simulation scenarios, as depicted in Fig. 3: (1) Preceding vehicle Cut-out from the lane, and (2) Vehicles crossing a T-Junction. Also, to emphasize the importance of V2X, we compare the case where vehicles are equipped with Radar only (i.e., only ACC is active), and the case where vehicles are equipped with both Radar and OBU (i.e., both CACC and CA are active).

In Scenario1, three cars travel on the same lane. The left bottom plot in Fig. 4 presents the velocity of the neighbouring vehicle A, which is in front of the ego car and is travelling at the desired cruise speed of the ego car. The ego car thus maintains the desired distance gap with zero acceleration (left top plot in Fig. 4). However, once the control algorithm detects the presence of a slower vehicle on the same lane travelling in the same direction, the ego car decelerates to maintain the desired time headway with the neighbouring vehicle B, which is in front of the neighbouring vehicle A. As expected, the Radar-based acceleration tends to increase to maintain the cruise speed, which depends on the speed of neighbouring vehicle A. Once A realises the presence of B, A moves to the adjacent lane to avoid the collision. This abrupt action by A brings B within the Radar-range of the ego car. After the cut-out, the ego car choses the Radar-based acceleration, as shown by the markers in the left bottom plot of Fig. 4 to maintain the desired distance, while the Prediction-based algorithm choses to accelerate by considering vehicle A: by design, the control algorithm chooses, as desired acceleration for ego car, the minimum between Prediction and Radar.

The right plots in Fig. 4 depict the results obtained with and without OBU. We recall that, without OBU, the Radar signal is the sole source of data for the ego car to decide the acceleration (ACC), while, with OBU, CACC can be used.

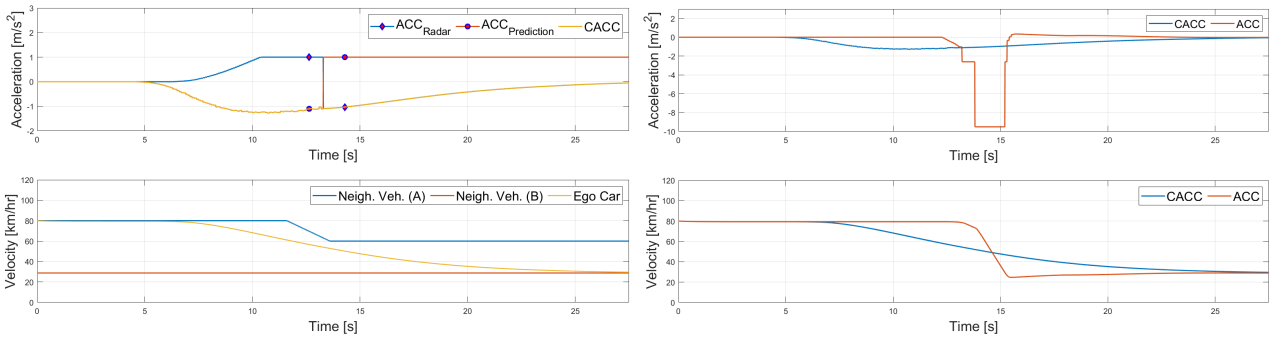

Fig. 4. Scenario1. Left: ego car acceleration (top) and speed trend (bottom). Right: acceleration (top) and speed (bottom) under ACC and CACC.

Without OBU, vehicle B gets in the Radar range at the last moment, so the ego car activates the partial brake (-2.6 m/s^2) in AEB mode for a brief time. Since the reduction in velocity is not enough to avoid imminent collision, it uses full brake force of -9.6 m/s^2 (if consistent with tire-pavement friction conditions) to avoid the collision (as evident from the steep reduction of velocity in the right bottom plot of Fig. 4). On the other hand, with the help of CAMs, the ego car detects the collision earlier and applies the brake gradually, as confirmed by the smooth speed reduction in the bottom right plot of Fig. 4.

In Scenario 2, the bottom left plot of Fig. 5 shows that, since the Radar sensor does not have any vehicle in its range, the controller keeps maintaining the desired cruise speed of the ego car. With the help of the CAMs, the control algorithm identifies the presence of the other vehicles and can predict that the slow-moving vehicle is on the ego car collision course. Therefore, the control algorithm decides to reduce the velocity of the ego car by decelerating. From the top left plot of Fig. 5, we also notice that ACC is increased to compensate the velocity reduction. The reason behind this behaviour is that there is no vehicle in the Radar range of the ego car, so it tries to maintain the desired cruise speed. Once the vehicle falls in the Radar range, the Radar-based acceleration also reduces to maintain the desired distance between the vehicles. As shown in Fig. 3, after crossing T-junction, the neighbour vehicle accelerates to 60 km/h. Then, the ego car also accelerates gradually to maintain the desired distance gap between the ego and the neighbour vehicle.

The comparison between ACC and CACC is presented in the right plots in Fig. 5. As in Scenario 1, the ACC mode makes the ego car activate the AEB to avoid the collision, while in the CACC mode the ego car can detect the presence of neighbouring vehicles and gradually reduce its velocity to avoid the collision. The acceleration variation of the ego car is much smoother under CACC, thus ensuring passenger comfort and safety.

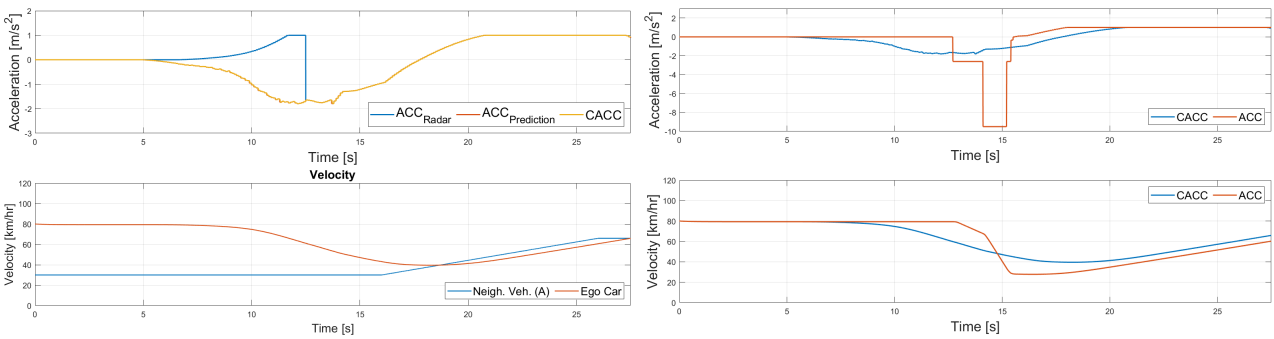

Fig. 5. Scenario2. Left: Ego car acceleration (top) and speed trend (bottom). Right: acceleration (top) and speed (bottom) under ACC and CACC.

Fig. 6 depicts the ego car's acceleration based on different forecasting periods (10, 7, and 5 seconds). For Scenario 1, Fig. 6(left) shows that, as expected, foreseeing the collision before 10 seconds performs better than the others. The acceleration trend of other time instances indicates a steep reduction in the acceleration, while the 10-second prediction instance shows a gradual reduction in the acceleration. For Scenario 2, Fig. 6(right) shows that the prediction period of 7 seconds performs better than the others. In particular, with forecasting time of 10 seconds, the acceleration is smooth in the initial phase, however it is not sufficient to keep the safety distance between the vehicles. Thus, at the later stage it chooses a higher deceleration value compared to the 7-second case.

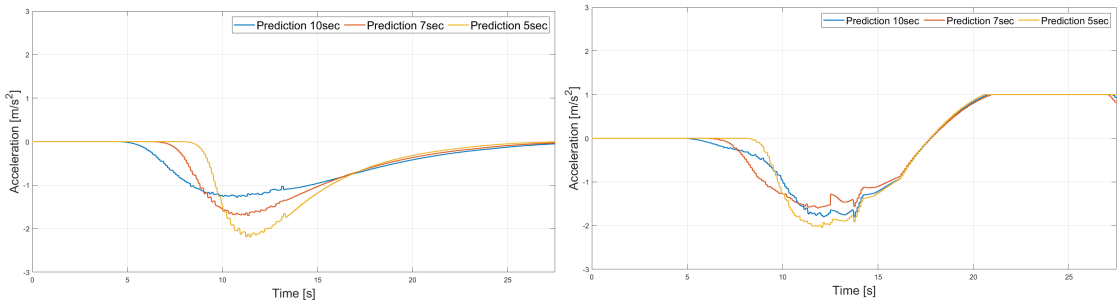

Fig. 6. Acceleration variation for different prediction time instances: Scenario1 (left) and Scenario 2 (right).

6. Conclusion

We proposed a flexible and efficient framework that focuses on integrating the functionalities of multiple simulators such as CarMaker, ns3, Simulink, and SUMO, to study the impact of connected cars on safety and traffic-efficiency applications. CarMaker is used to analyse the dynamics of the vehicles and simulate the on-board sensors, while SUMO represents the vehicle interactions along the simulated road scenarios, and ns3 the behaviour of the communication infrastructure and the connected cars. To combine the strengths of these simulators, we leveraged Simulink and a Python engine as interfaces. Within the Python engine, we also developed the control strategies for cooperative adaptive cruise control, exploiting both ADAS and CAMs, and for collision avoidance, exploiting CAMs. Our results show the prominent benefits of wireless connectivity vehicles in terms of both traffic efficiency and safety. In particular, connected cars were able to predict a collision and avoid it with gradual acceleration/deceleration, whereas vehicles equipped with on-board sensors only ended up activating Automatic Emergency Braking (AEB) to reduce the impact of the collision.

Future work will focus on the implementation and use of the vehicle dynamic and the ADAS sensors within Simulink, as well as extensive experiments including multiple scenarios. Among these, complex traffic conditions will be considered, where lateral movements can also be performed to avoid a collision. Finally, we will identify suitable metrics and assess other important aspects such as the vehicles' eco-footprint.

References

[1] F. Arena, G. Pau, "An overview of vehicular communications," *Future Internet*, Vol. 11, No. 2, 2019.
[2] O. Urra, S. Ilarri, "MAVSIM: Testing VANET applications based on mobile agents," in Cognitive Vehicular Networks, 2016.
[3] C. Lei, E. Martijn van Eenennaam, W. Klein Wolterink, Jeroen Ploeg, Georgios Karagiannis and Geert Heijenk, "Evaluation of CACC string stability using SUMO, Simulink, and OMNeT++," *EURASIP Journal on Wireless Communications and Networking*, Vol. 116, 2012.
[4] A. Ibrahim, C. B. Math, D. Goswami, T. Basten, H. Li, "Co-simulation framework for control, communication and traffic for vehicle platoons," *Euromicro DSD,* 2018.
[5] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, S. Scarpina, "Performance analysis of C-V2I-based automotive collision avoidance," *IEEE WoWMoM*, Chania, 2018.
[6] J. Kaths, S. Krause, "Integrated simulation of microscopic traffic flow and vehicle dynamics," *IPG Apply and Innovate*, Karlsruhe, 2017.
[7] S. Stević, M. Krunić, M. Dragojević, N. Kaprocki, "Development and validation of ADAS perception application in ROS environment Integrated with CARLA simulator," *TELFOR*, Belgrade, Serbia, 2019.
[8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, "Carla: An open urban driving simulator," *arXiv:1711.03938*, 2017.
[9] C. Pilz, G. Steinbauer, M. Schratter, D. Watzenig, "Development of a scenario simulation platform to support autonomous driving verification," *IEEE ICCVE*, 2019.
[10] K. Tong, Z. Ajanovic, G. Stettinger, "Overview of tools supporting planning for automated driving," *arXiv:2003.04081*.
[11] A. Choudhury, et al., "An integrated V2X simulator with applications in vehicle platooning," *IEEE ITSC*, 2016.
[12] IPG CarMaker, "Programmer's Guide Version 8.0," Karlsruhe, 2019.
[13] ETSI, "EN 302 637-2, EN 302 637-3," 2014.
[14] Y. Li, H. Wang, W. Wang, L. Xing, S. Liu, X. Wei, "Evaluation of the impacts of cooperative adaptive cruise control on reducing rear-end collision risks on freeways," *Accident Analysis & Prevention,* Vol. 98, pp. 87–95, 2017.
[15] ACI (Italian Automobile Club), "Art. 149. of new road codex - safety distance between vehicles," *Decree of the the Republic*, No. 495, 1992.
[16] A. Arcidiacono, "ADAS virtual validation: ACC and AEB case study with IPG CarMaker," *Master Thesis*, Politecnico di Torino, 2018.