Train Here, Drive There: Simulating real-world use cases with fully-autonomous driving architecture in CARLA simulator

Carlos Gómez-Huélamo[1], Javier Del Egido[1], Luis M. Bergasa[1], Rafael Barea[1], Elena López-Guillén[1], Felipe Arango[1], Javier Araluce[1], Joaquín López[2]

[1] Electronics Department, Ugniversity of Alcalá (UAH), Spain,
{luism.bergasa, rafael.barea, elena.lopezg}@uah.es,
{carlos.gomezh, javier.egido, juanfelipe.arango, javier.araluce}@edu.uah.es,
[2] Department of Systems Engineering and Automation, University of Vigo, Pontevedra, Spain,
{joaquin@uvigo.es}

Abstract. This work presents the validation of our fully-autonomous driving architecture in the CARLA open-source simulator, by using some challenging driving scenarios inspired on the CARLA Autonomous Driving Challenge (CADC), focusing on our decision-making layer, based on Hierarchical Interpreted Binary Petri Nets (HIBPN). First, our ROS (Robot Operating System) based autonomous driving architecture is introduced. Second, the CARLA simulator is described, outlining the steps conducted to merge our architecture with this simulator and the advantages to create ad-hoc driving scenarios for use cases validation. Finally, the paper validates the architecture by means of some challenging driving scenarios such as: STOP, Pedestrian Crossing, Adaptive Cruise Control (ACC) and Unexpected Pedestrian. Some qualitative (video files) and quantitative (trajectory and linear velocity segmented with its corresponding Petri Net states) results are presented for each use case, validating our architecture in simulation as a preliminary stage before implementing it in our real autonomous electric car.

Keywords: CARLA, Autonomous Vehicles, ROS, Simulation, Decision-Making, Use Cases

1 INTRODUCTION

Autonomous vehicles (AV) is one of the most challenging engineering tasks of our era. These AVs are expected to be driven throughout a highly dynamic environment with a reliability greater than human beings and full autonomy. In order to perform this task, an AV must be equipped with robust algorithms and multiple sensors in such a way the vehicle is able to perceive the surrounding environment in real-time with a high precision, estimating the position of the objects with a very low error in the 3D/BEV (Bird's Eye View) space. However, making use of the object detection alone is not good enough for an AV to navigate in such complex environments, but the vehicle must also be able to track these obstacles along the scene to estimate their velocity and predict their future positions, also known as scene prediction. In that sense, the perception layer of an intelligent vehicle can be divided into three sequential stages: Detection, Multi-Object Tracking and Scene Prediction stage. The perception layer, based on the combination of different perception sensors and tracking algorithms, must be able to provide the most probable future behaviours, velocity and position of the main objects in the scene. This, combined with the control and mapping/planning layer information, feeds the decision-making layer of the ego-vehicle in order to successfully conduct some common urban use cases such as predicting if a pedestrian will start to cross a certain pedestrian crossing, predict the presence of an adversary vehicle in the next give-way or even when the ego-vehicle is carrying out the Adaptive Cruise Control (ACC), where it must adjust the vehicle speed to maintain a safe distance from ahead vehicles.

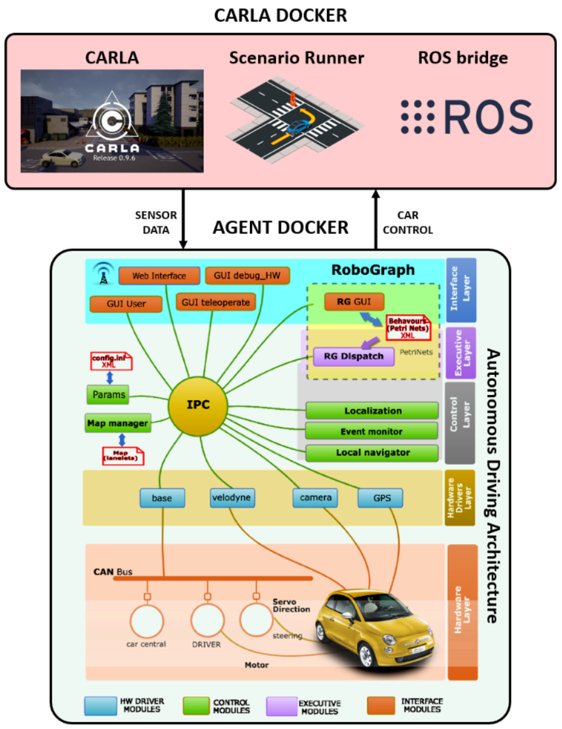


Fig. 1. Our autonomous driving architecture under CARLA simulation. GUI = Graphical User Interface; HW = Hardware; RG = RoboGraph; GPS = Global Positioning System; ROS = Robot Operating System; LiDAR = Light Detection and Ranging; IPC = Inter-Process Communication

In spite of all impressive efforts in the development of autonomous driving technology [1], fully-autonomous navigation in arbitrarily complex environments is still years away. The reason for this is two-fold: Firstly, informed decision-making requires accurate perception. Most of the existing perception systems produce errors at a rate not acceptable for autonomous driving [2]. Secondly, autonomous systems that are operated in complex dynamic environments require intelligent systems that generalizes to unpredictable situations in a timely manner.

An AV must be able to carry out driving decisions based on the offline and online information processed by the vehicle. In that sense, the offline information can be identified as the prior knowledge of the system, such as the topological relations and geographic information of the environment based on high-definition maps [3], the vehicle dynamic and the traffic rules based on behavioural decision-making systems [4]. On the other hand, the online information, also known as the traffic situation, is obtained through the global perception system of the vehicle, which involves different on-board sensors (like Inertial Measurement Unit (IMU)), Light Detection and Ranging (LiDAR), Global Positioning System (GPS), Steering wheel angle and Cameras). Traditionally, autonomous driving systems break down hierarchically into four main components [5]: Route Planning, Executive Layer, Motion Planning and Vehicle Control.

The scope of this paper is the evaluation of our ROS based fully-autonomous driving architecture (Agent Docker in Fig. 1), focusing on the behavioural decision-making layer, in the context of some challenging driving scenarios inspired in the CARLA [6] Autonomous Driving Challenge. This work is the continuation of our previous validation carried out in a more limited simulator (V-REP [7]). The validation with CARLA, a novel open-source autonomous driving simulator, featured by its flexibility, hyper-realism and real-time working, reinforces the simulation design stage. We hope that our distributed system can serve as a solid baseline on which future research can build on to advance the state-of-the-art in validating fully-autonomous driving architectures using hyper-realistic simulation, as a preliminary stage before implementing the architecture in our real electric car prototype.

The remaining content of this work is organized as follows. The next section presents the work related to this research, focused on the decision-making layer and the importance of simulation in the context of AV. Section 3 presents our autonomous navigation architecture. Section 4 describes the main features of the CARLA simulator, its integration with out AV architecture, regarding the way in which the sensors perceive the environment and how this information is processed to feed the decision-making system, including both the traffic situation and prior knowledge. Section 5 describes some interesting use cases, including a table with their main features, giving rise to some quantitative and qualitative results to illustrate the proposed architecture performance. Finally, section 6 deals with the future works and concludes the work.

2 RELATED WORKS

As mentioned in the previous section, a fully-autonomous driving architecture (L5 in the J3016 SA [8]) is still years away, mainly due to technical challenges, but also due to social and legal ones [9].

No industry organization has shown a ratified testing methodology for L4/L5 autonomous vehicles. The autonomous driving community gives a simple reason: although some regulations have been defined for these L4/L5 levels, simulation is critical to build safe AV. However, despite the fact that current automotive companies are very good at testing the individual components of the autonomous driving architecture, there is a need to test intelligent vehicles full of advanced sensors [10] and sharing information among them. In this context, artificial intelligence is increasingly being involved in processes such as detecting the most relevant objects around the car (deep learning based multi-object tracking systems), or evaluating the current situation of the vehicle to conduct the safest decision (e.g. deep reinforcement learning applied to behavioural systems). Moreover, it is important to consider the presence of sensor redundancy in order to reach safe navigation in such a way that the different sensors and associated algorithms must be integrated together, required to validate the whole system, not just the individual components.

Regarding urban environment complexity, the system must be tested in countless environments and scenarios, which would escalate the cost and development time exponentially with the physical approach. Regarding this, the use of photo-realistic simulation (virtual development and validation testing) and an appropriate design of the driving scenarios are the current keys to build safe and robust AV. We propose the use of CARLA as the best open-source tool to reach these goals.

On the other hand, behavioural decision-making layer must provide tools to model the sequence of events and action, based on some predefined traffic rules, that can take place in the different traffic scenarios. In terms of AV, different approaches have been proposed to design the decision-making layer, including different heuristic solutions [11] based on identifying a set of driving scenarios or driving contexts (e.g. intersection handling or lane driving), reducing the number of environmental features to which the vehicle must be focused, according to each driving context.

The design of the decision-making layer for AV is challenging due to uncertainty in the knowledge about the driving situation and the state of the vehicle. This uncertainty comes from different sources, such as the estimation of the continuous state of nearby external agents, like other vehicles or pedestrians, whose behaviour is usually unpredictable. Hence, in order to design an optimal decision system uncertainty must be considered. Despite the fact that Partially Observable Markov Decision Processes (POMDP) [12] offer a framework to manage uncertainty, they are not scalable to real-world scenarios because of the associated complexity. Other approaches tackle this layer using simple discrete events systems, which are not enough complex to model real-world driving scenarios. Decision Trees (DT) [13], Hierarchical Finite-State Machines (HFSM) [14] and Finite-State Machines (FSM) [15] are among the most popular approaches to design a decision making system. Moreover, several teams [4] in the Defense Advanced Research Projects Agency (DARPA) Urban Challenge [16] used some of these approaches.

Nevertheless, for complex problems in which common urban driving scenarios are included, the difficulty in representing the system as FSM lies in dealing with the need to implement the potentially high number of transitions due to the state explosion problem. Behaviours Trees (BT) can get rid of these drawbacks since the transitions among the states are implicit to their control structure formulated as a tree, giving rise to a higher flexibility, maintainability and extensibility with respect to FSMs.

However, BTs and simple discrete event systems share a common problem: A naive implementation usually gives rise to blocking behaviour, which make them unsuitable for autonomous driving. In that sense, concurrency and parallel activities can be easily programmed using Petri Nets (PN). PNs are a powerful tool to design, model and analyze concurrent, sequential and distributed systems. While in a FSM there is always a single current state, in PNs there may be several states that can change the state of the PN. In particular, in this work we model every behaviour as Hierarchical Interpreted Binary Petri Net (HIBPN) [17] [7], as shown in Fig. 2(b), where a PN can start/stop another PN depending on its hierarchy.

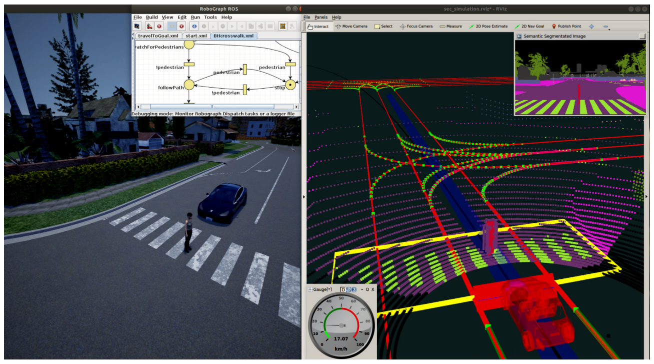
3 AUTONOMOUS NAVIGATION ARCHITECTURE

Our autonomous navigation framework (Fig. 1) is featured by a modular architecture where individual modules process the information in an asynchronous way. These modules are standalone processes that communicate each other using the ROS Inter-Process Communication (IPC) system. The publish/subscribe concept is used to provide non-blocking communications. These software modules are organized in four layers as following:

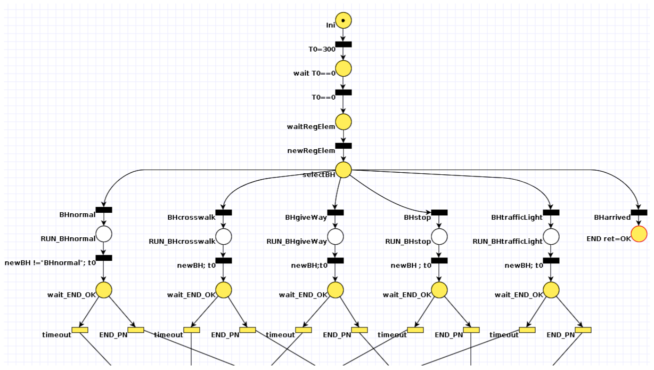
- **Hardware drivers layer** Set of programs that implements different hardware devices, such as sensors and actuators.
- **Control layer** Set of programs that implements the vehicle control and navigation functionality. These programs include the path planning (map manager), localization, reactive control (local navigator) and a program to process most of the perception sensors to detect and track relevant events (event monitor).
- **Executive layer** Set of programs that coordinates the sequence of actions required to be executed to perform the current behaviour.
- **Interface layer** Set of processes that interact with the user and enable the communication to other processes for multi-robot applications.

The motion control is broken down into lower-level (LL) reactive control and high-level (HL) planning. First of all, the map manager (control layer) loads the map made up by a sequence of lanelets [18], in which the user must specify the start and goal position of the route. Then, the lanelet path is generated using an A* algorithm [19]. Besides this, the map manager serves other queries from other modules related to the map. For example, it must provide the most relevant lanes around the vehicle, such as intersection lanes or contiguous lanes for stop and overtaking use cases respectively, or providing the position of the regulatory elements in the route. Finally, a global path planner calculates a suitable path to be followed by the car. The goal of the local navigation module is to safely follow this path, keeping the vehicle within the driving lane and modifying the vehicle dynamic when was required by the behaviour constraints established by the HL planning. In order to do that, the local navigation system calculates the curvature to guide the car from its current position to a look-at-head position placed in the center of the lane using the Pure Pursuit algorithm [20]. This curvature is used as the reference for the reactive control, where an obstacle avoidance method is implemented based on the Beam Curvature Method (BCM) [21]. Combining both the Pure Pursuit algorithm and BCM, the local navigation system keep the vehicle centered in the driving lane while is able to avoid unknown obstacles which can partially block the lane.

In terms of environment perception, we perform a sensor fusion [7] between LiDAR and camera in order to detect and track the most relevant objects in the scene. First, a



(a)



(b)

Fig. 2. (a) Simulation example: On the left, the CARLA simulator illustrates the urban scenario our sensors face. On the right, the RVIZ simulator shows the detection in the coloured point cloud. (b) Start Petri Net of our behaviour decision-making system .

semantic segmentation is performed in the RGB image taken from the camera to detect the different obstacles and the driving area. In our real-world vehicle, we use our Efficient Residual Factorized ConvNet (ERFNet) [22] for real-time semantic segmentation, while in simulation, CARLA already provides a sensor including the semantic segmentation of the scene. We merge the 3D LiDAR point cloud and the 2D segmented pixels so as to obtain a coloured point cloud, where the points out of the FoV of the camera are not coloured. Then, we carry out a coloring clustering, obtaining the most relevant objects in the scene, as shown in Fig. 2(a). Multi-Object Tracking is performed by combining the Precision Tracker approach [23], BEV Kalman Filter [24] and Nearest Neighbour algorithm [25]. All this local perception information is published through the event monitor module.

The behavioural decision-making has been implemented using HIBPNs, where the main Petri Net is fed with the inputs provided by the local perception (event monitor module), the vehicle odometry in the map and the map manager information. To implement these HIBPNs, we use the RoboGraph tool [4], employed by the authors in other mobile robot applications.

4 SIMULATION STAGE

As commented in previous sections, virtual testing is becoming increasingly importance in terms of AV. Since the urban environment is highly complex, the navigation architecture must be tested in countless environments and scenarios, which would escalate the cost and development time exponentially with the physical approach. Some of the most used simulators in the field of AV are Microsoft Airsim [26], recently updated to include AV although it was initially designed for drones, NVIDIA DRIVE PX [27], aimed at providing AV and driver assistance functionality powered by deep learning, ROS development studio [28], fully based on the Cloud concept where a cluster of computers allows the parallel simulation of as many vehicles as required, V-REP [29], with an easy integration with ROS and a countless number of vehicles and dynamic parameters and CARLA [6], which is the newest open-source simulator for AV based on Unreal engine. In [30], [7] we validated the proposed navigation architecture using the V-REP simulator cause our previous experience in this simulator. However, the paradigms of real-time and high realism could not be deeply analysed due to V-REP is not designed to create high realistic urban driving scenarios, so the analysis of traffic use cases is hard due to the unrealism and slowness of the simulator environment. Since our project is characterized by being open-source, we decided to integrate our autonomous driving architecture with the open-source CARLA simulator. This offers a much more interesting environment in terms of traffic scenarios, perception, real-time and flexibility, which are key concepts for our algorithms. In this work we have used the 0.9.6 version of CARLA, so the CARLA ROS bridge and the Scenario Runner. Additional tools needed for the simulation, were configured according to this version.

In [7] we used the simulation environment, the V-REP ROS bridge and the autonomous navigation architecture in the same host machine, running on Ubuntu 16.04. For this work, we decided to split them into two Ubuntu 18.04 Docker [31] images, the CARLA world and the CARLA ROS bridge in one image and the autonomous navigation architecture in the other one, so as improve the portability, flexibility and isolation of our work.

4.1 Environment

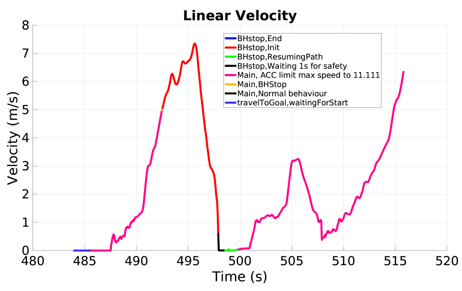
CARLA is implemented as an open-source layer over Unreal Engine 4 (UE4) [32]. This simulation engine provides CARLA an hyper-realistic physics and an ecosystem of interoperable plugins. In that sense, CARLA simulates a dynamic world and provides a simple interface between an agent that interacts with the world and itself. In order to support this functionality, CARLA was designed as a server-client system, where the simulation is run and rendered by the server. CARLA environment is made up by 3D models of static objects like infrastructure, buildings or vegetation, as well as dynamic objects such as vehicles, cyclists or pedestrians. They are designed using low-weight geometric models and textures but maintaining visual realism due to a variable level of detail and carefully crafting the materials. On the other hand, the maps provided by CARLA are in OpenDrive [33] format, whilst we use the lanelet approach based on OpenStreetMap (OSM) [34] service. We transform the OpenDrive format into OSM format by using the converter proposed by [35]. Then, based on this lanelet map, we manually include the regulatory traffic information to generate an enriched topological map useful for navigation. Furthermore, the OSM map uses WGS84 coordinates (latitude, longitude and height) whilst the simulator provides Cartesian coordinates (UTM) relative to an origin (usually the center of the map). Then, geometric transformations between both systems are calculated using the libraries implemented in the ROS geodesy package.

Table 1. Summary of the main features, including inputs, outputs and involved modules, for the main Petri Nets of our work. **MM** = Map Manager; **EM** = Event Monitor; **LN** = Local Navigator; **RGD** = RoboGraph Dispatch; **PC** = Pedestrian Crossing

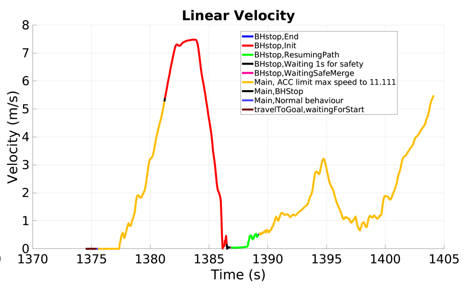
Petri Net	Inputs	Input modules	Outputs	Output modules	Nodes / Transitions
Background	Man/auto	GUI User	Run Selector	RGD	8/9
	goToPoint	GUI User	Stop selector	RGD	
Selector PN	Reg. Element	MM	Run PC	RGD	21/29
	Dist. Reg Element	MM	Stop PC	RGD	
	End Reg. Element	MM	Run GiveWay	RGD	
	Reg. Element	EM	Stop GiveWay	RGD	
	End Reg. Element	EM	Run STOP	RGD	
	FrontCarVel	EM	...	...	
	Odom	Base			
Pedestrian Crossing	NoPedestrian	EM	WatchforPedetrians	EM	10/13
	Pedestrian	EM	SetMaxVel	LN	
	Dist. To PC	MM	StopAtPoint	LN	
	PC Over	MM			
	Force End	RGD			
	Stopped	LN			
STOP	SafetoMerge	EM	CheckSafeMerge	EM	9/12
	NotSafetoMerge	EM	SetMaxVel	LN	
	DistToStop	MM	StopAtPoint	LN	
	StopOver	MM			
	Force End	RGD			
	Stopped	LN			
Adaptive Cruise Control (ACC)	Current Velocity	MM	SetMaxVel	LN	4/6
	FrontCarVel	EM			
	DistToFrontCar	MM			

4.2 Vehicle

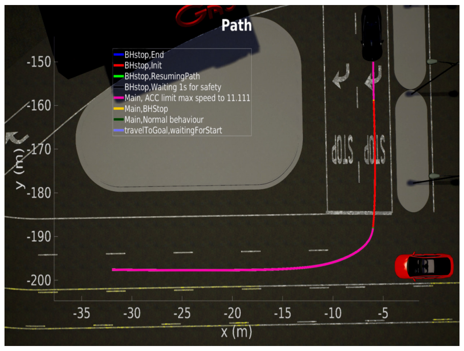
In order to link the vehicle in CARLA and its corresponding model in RVIZ (Fig. 2(a)), we modify the ROS bridge associated to the CARLA simulator. The CARLA ROS bridge is a ROS package that aims at providing a bridge between CARLA and ROS, being able to send the data captured by the on-board sensors and other variables associated to the vehicle in the form of topics and parameters understood by ROS. In that sense, we modify some parameters of the speed and acceleration PID, the sensors position and orientation, the Ackermann control node and the waypoint publisher to adjust the CARLA ego-vehicle parameters to our project. The reference system of the vehicle is centered on the rear axle, and the orientation of the frame is based on the LiDAR frame, that is, X-axis pointing to the front, Y-axis to the left and Z-axis above. The angular velocity at Z is positive according to the right hand rule, so the right turns are considered negative. The speed commands generated by the simulator are interpreted by the local navigation system (in particular, the low-level controller), giving rise the corresponding accelerations and turning angle.



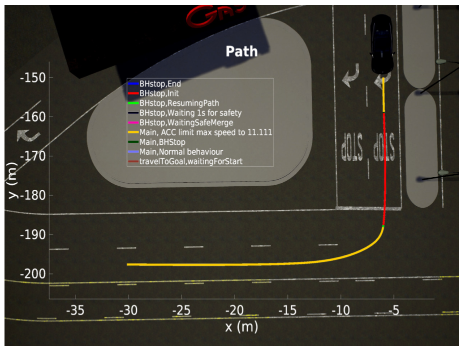
(a)



(b)



(c)



(d)

Fig. 3. First and second row represent, respectively, the linear velocities and described trajectory projected onto the corresponding CARLA scenario for Stop with car detection (a,c) and Stop with no car detection (b,d) behaviours

4.3 Sensors

From the sensors perspective, the agent sensor suite can be modified in a flexible way. Most common sensors in CARLA world are LiDAR, GPS and RGB cameras as well as their corresponding pseudo-sensors that provide semantic segmentation and groundtruth depth. Moreover, camera parameters include 3D orientation and position with respect to the car coordinate system, field-of-view and depth of field. The original configuration of the on-board sensors of the ego-vehicle had a 800x600 RGB camera sensor placed at the front of the vehicle with a FoV of 100 ^{o}, a 32-channels LiDAR, a

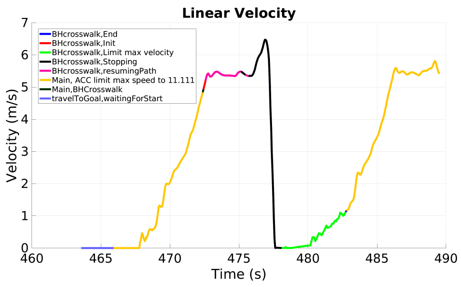
GPS sensor, a collision sensor and a lane invasion sensor. In our case, the position of the LiDAR and camera sensors are manually configured in order to obtain the same frames distribution that in our real-world vehicle. Based on the bridge, the LiDAR information can be published in PointCloud2 ROS format with the Z axe pointing up, Y left and the X inwards. Image messages are published with the Z axe inwards, the Y down and the X pointing right in the image plane. We take advantage of the semantic segmentation provided by CARLA so as to colour the LiDAR point cloud and perform the coloring clustering, placing the pseudo-sensor at the same position and orientation that the RGB camera sensor, both with a resolution of 1280x720, since it is the resolution of the ZED camera equipped in our real-world electric car. Moreover, the position of the GPS sensor is displaced to the center of the rear axle, required by the local navigation system. The remaining sensors keep unmodified. As observed, CARLA provides a straightforward way to add or remove sensors from the vehicle or even modify their parameters, to adjust the simulation to the real-world as best as possible.

5 EXPERIMENTAL RESULTS

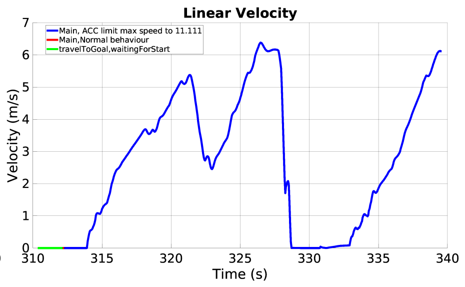
One of the best advantages of CARLA is the possibility to create ad-hoc urban layouts, useful to validate the navigation architecture in challenging driving scenarios. This code can be downloaded from the Scenario Runner repository, associated to the CARLA GitHub. This repository offers an execution engine for CARLA and traffic scenario definition. The scenarios are inspired in the CARLA Autonomous Driving Challenge (CADC), selected from the NHTSA pre-crash typology [36]. The features of these scenarios can be defined through a Python interface, such as the presence of additional obstacles, start condition for the adversary (e.g. pedestrian or vehicle), position and orientation of the dynamic obstacles in the environment, weather, etc. In this work we study the linear velocity and the odometry of the ego-vehicle throughout the following use cases: Stop with car detection, Stop with no car detection, Pedestrian Crossing, Adaptive Cruise Control (ACC) and Unexpected Pedestrian. A video demo for each of them can be found in the following play list Simulation Use Cases [3]. Due to size constraints, we refer the readers to [7] for a deeper explanation of the behaviour that the ego-vehicle must carry facing to the corresponding traffic scenario. Nevertheless, Table 1 shows the main HIBPNs features used to manage the logic of these use cases(inputs, input modules, outputs, output modules, number of nodes and number of transitions).

The resulting graphics are segmented using different colours, corresponding to the current node of the associated PN. Both Stop use cases are inspired in the Traffic Scenario 09 (Right turn at an intersection with crossing traffic) of the CADC, demonstrating a similar behaviour since the logic performed by the PNs is the same. It is appreciated that the ego-vehicle waits 2.3 s in front of the stop line in the use case of Stop with no detection (Fig. 3(b), Fig. 3(d)). This time is approximately the same that the ego-vehicle waits with the presence of an adversary vehicle (Fig. 3(a), Fig. 3(c)). This behaviour is coherent, since the Stop PN presents a transition that requests 0 m/s for the ego-vehicle in addition to 1 extra s to wait for safety, in the same way that a real-world car would

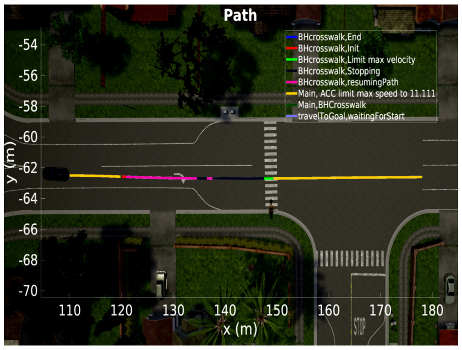
[3] Simulation Use Cases link: https://cutt.ly/prUzQLi



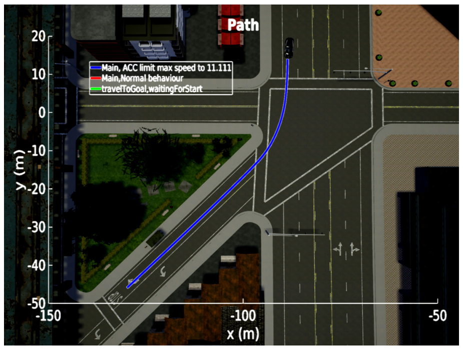
(a)



(b)



(c)


Path
Main, ACC limit max speed to 11.111
Main,Normal behaviour
travelToGoal,waitingForStart
y (m)
x (m)

(d)

Fig. 4. First and second row represent, respectively, the linear velocities and described trajectory projected onto the corresponding CARLA scenario for Pedestrian Crossing (a,c) and Unexpected Pedestrian (b,d) behaviours

do. Regarding the Pedestrian Crossing use case (Fig. 4(a), Fig. 4(c)), it is inspired in the Traffic Scenario 04 (Obstacle avoidance with prior action) of the CADC. As observed in Fig. 2(a), in this work we define an additional safety area (Yellow rectangle in RVIZ simulator) around the pedestrian crossing, in order to activate the pedestrian detection, carried out by the event monitor, in all this area, including the sidewalks.

While V-REP does not offer the possibility to configure a variable velocity for the adversary, being the ACC limited to decrease the ego-vehicle velocity till the adversary fixed velocity [7], we take advantage of the possibilities offered by CARLA to configure the behaviour of the adversary in this use case. In that sense, Fig. 6(a) shows the linear velocity under the effects of the ACC in blue, being the linear velocity of the ego-vehicle adjusted to the variable adversary linear velocity. Moreover, Fig. 6(c) shows how the distance is decreased till 22/23 m, where the ACC is kept until the traffic scenario is concluded. It can be observed that the ACC behaviour starts at the exact moment in which the adversary vehicle is detected. This new version of our ACC analysis is inspired in the Traffic Scenario 02 (Longitudinal control after leading vehicle's brake) of the CADC.

The last use case that we validate in this paper is inspired in the Traffic Scenario 03 (Obstacle avoidance without prior action) of the CADC. In this traffic scenario, the

Table 2. Reactive control analysis on CARLA simulator

NV (km/h) / D (m)	**12**	**16**	**20**	**24**
14.4	🟩 6.75	🟩 11.41	🟩 15.20	🟩 18.73
18	🟥 6.17	🟩 10.46	🟩 13.75	🟩 17.16
21.6	🟥 5.31	🟩 9.07	🟩 12.89	🟩 16.58
25.2	🟥 3.48	🟩 8.72	🟩 12.15	🟩 15.24
28.8	🟥 2.66	🟥 8.39	🟥 11.78	🟩 13.12

NV: Nominal Velocity of the ego-vehicle (km/h). D: Minimum distance between the pedestrian and vehicle BEV centroid to allow the pedestrian start its trajectory

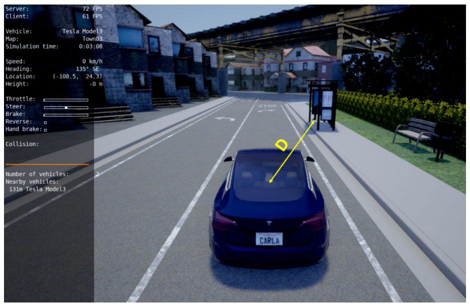
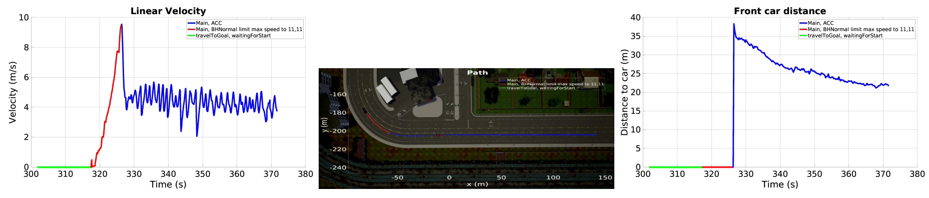

```
Server:                72 FPS
Client:                61 FPS

Vehicle:          Tesla Model3
Map:                    Town03
Simulation time:       0:03:08

Speed:                  0 km/h
Heading:               135° SE
Location:       (-108.5,  24.3)
Height:                   -0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:

Collision:

Number of vehicles:          2
Nearby vehicles:
 131m Tesla Model3
```

Fig. 5. Unexpected Pedestrian scenario. The pedestrian can be found behind the bus stop, so the perception systems detects it at the moment it is entering the road

ego-vehicle suddenly finds an unexpected obstacle on the road and must perform an avoidance maneuver or an emergency break. In our case, we design our own scenario in such a way an unexpected pedestrian jumps to the road, being totally occluded by a bus stop kiosk. As expected, Fig. 4(b) and Fig. 4(d) show that no high-level behaviour is launched because this situation is not included as an use case in the PN. However, our low-level (reactive) control performs an emergency break until the car is stopped in front of the obstacle, and resumes the navigation once the obstacle leaves the driving lane. In this way, we validate the reactive control of our architecture, always in background execution, in those cases in which none of the specific use cases PN are launched. This strategy shows a good working in unexpected situations due to the high frequent execution of our reactive control.

A robustness analysis of our reactive control is depicted in Table 2. We show some parameters of the obstacle avoidance maneuver for different jumping distances (D(m)) of the unexpected pedestrian and different nominal speeds of the ego-vehicle (NV(Km/h)). For each combination, the distance at which the reactive control detects the obstacle in the road is calculated (m). Green cells indicate no collision has taken place and read that the pedestrian has been overwhelmed. The *D* parameter represents the initial Euclidean distance between the adversary pedestrian and the ego-vehicle BEV centroid, as illustrated in Fig. 5. This distance corresponds with the initial condition of the pedestrian, placed at the sidewalk, 1.5 m away from the road. Whilst in the Pedestrian Crossing use case an additional safety area is considered to detect the presence of pedestrians, in this case the reactive control detects the pedestrian once it is inside the lane. After the



(a) (b) (c)

Fig. 6. Analysis of the ACC use case with variable adversary vehicle velocity. (a) represents the ego-vehicle linear velocity, (b) the ego-vehicle odometry projected onto the CARLA world, (c) analysis of the euclidean distance between the ego-vehicle and adversary throughout the use case

detection a braking maneuver is performed. As expected, the faster the vehicle goes, the lower the distance at which the reactive control detects for the first time the pedestrian inside the lane. This is due to the ego-vehicle has travelled a greater distance, increasing the likelihood of colliding with the pedestrian, and the ego-vehicle must perform the emergency break in a shorter distance.

6 CONCLUSIONS AND FUTURE WORKS

This work presents the validation of our ROS-based fully-autonomous driving architecture, focusing in the decision-making layer, with CARLA, a hyper-realistic, real-time, flexible and open-source simulator for autonomous vehicles. The simulator and its bridge, in charge of communicating the CARLA environment with our ROS-based architecture, on the one hand, and the navigation architecture, on the other hand, have been integrated in two Docker images, in order to gain flexibility, portability and isolation. The decision-making is based on Hierarchical Interpreted Binary Petri Nets (HIBPN), and our perception is based on the fusion of GPS, camera (including semantic segmentation) and LiDAR. The validation has consisted on the study of some traffic scenarios inspired on the CARLA Autonomous Driving Challenge, such as Stop, Pedestrian Crossing, Adaptive Cruise Control and Unexpected Pedestrian, in addition to an analysis of our reactive control in terms of emergency brake without prior action. We hope that our distributed system can serve as a solid baseline on which others can build on to advance the state-of-the-art in validating fully-autonomous driving architectures using virtual testing. As future works, a deep learning based 3D Multi-Object Tracking will be implemented and multiple adversaries will be included in order to get more challenging situations to improve the reliability, effectiveness and robustness of our system, so as to validate the architecture and test it in our real-world electric vehicle.

ACKNOWLEDGMENT

This work has been funded in part from the Spanish MICINN/FEDER through the Techs4AgeCar project (RTI2018-099263-B-C21) and from the RoboCity2030-DIH-CM project (P2018/NMT- 4331), funded by Programas de actividades I+D (CAM) and cofunded by EU Structural Funds.

References

1. E. D. Dickmanns, B. Mysliwetz, and T. Christians, "An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1273–1284, 1990.
2. J. Janai, F. Guney, J. Wulff, M. J. Black, and A. Geiger, "Slow flow: Exploiting high-speed cameras for accurate and diverse optical flow reference data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3597–3607, 2017.
3. E. Murciego, C. G. Huélamo, R. Barea, L. M. Bergasa, E. Romera, J. F. Arango, M. Tradacete, and Á. Sáez, "Topological road mapping for autonomous driving applications," in *Workshop of Physical Agents*, pp. 257–270, Springer, 2018.
4. J. López, P. Sánchez-Vilariño, R. Sanz, and E. Paz, "Implementing autonomous driving behaviors using a message driven petri net framework," *Sensors*, vol. 20, no. 2, p. 449, 2020.
5. B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.
6. A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.
7. C. Gómez-Huelamo, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, and P. Sánchez, "Simulating use cases for the uah autonomous electric car," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2305–2311, IEEE, 2019.
8. S. Taxonomy, "Definitions for terms related to driving automation systems for on-road motor vehicles (j3016)," tech. rep., Technical report, Society for Automotive Engineering, 2016.
9. R. Matthaeia, A. Reschkaa, J. Riekena, F. Dierkesa, S. Ulbricha, T. Winkleb, and M. Maurera, "Autonomous driving: Technical, legal and social aspects," 2015.
10. H. Schöner, "The role of simulation in development and testing of autonomous vehicles," in *Driving Simulation Conference, Stuttgart*, 2017.
11. C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.
12. H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces.," in *Robotics: Science and systems*, vol. 2008, Zurich, Switzerland., 2008.
13. C. Fernandez, R. Izquierdo, D. F. Llorca, and M. A. Sotelo, "A comparative analysis of decision trees based classifiers for road detection in urban environments," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 719–724, IEEE, 2015.
14. P. Beeson, J. O'Quin, B. Gillan, T. Nimmagadda, M. Ristroph, D. Li, and P. Stone, "Multiagent interactions in urban driving," 2008.
15. M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3, pp. 2436–2441, IEEE, 2003.
16. M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*, vol. 56. springer, 2009.
17. J. L. Fernández, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary petri nets to build robust mobile robot applications: Robograph," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1372–1377, IEEE, 2008.
18. P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pp. 420–425, IEEE, 2014.

19. U. Brandes, “A faster algorithm for betweenness centrality,” *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.
20. N. Y. Ko and R. G. Simmons, “The lane-curvature method for local obstacle avoidance,” in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*, vol. 3, pp. 1615–1621, IEEE, 1998.
21. J. L. Fernández, R. Sanz, J. Benayas, and A. R. Diéguez, “Improving collision avoidance for mobile robots in partially known environments: the beam curvature method,” *Robotics and Autonomous Systems*, vol. 46, no. 4, pp. 205–219, 2004.
22. E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “Erfnet: Efficient residual factorized convnet for real-time semantic segmentation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.
23. D. Held, J. Levinson, and S. Thrun, “Precision tracking with sparse 3d and dense color 2d data,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1138–1145, IEEE, 2013.
24. R. E. Kalman, “A new approach to linear filtering and prediction problems,” 1960.
25. J. S. Beis and D. G. Lowe, “Shape indexing using approximate nearest-neighbour search in high-dimensional spaces,” in *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pp. 1000–1006, IEEE, 1997.
26. S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and service robotics*, pp. 621–635, Springer, 2018.
27. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
28. D. S. Michal and L. Etzkorn, “A comparison of player/stage/gazebo and microsoft robotics developer studio,” in *Proceedings of the 49th Annual Southeast Regional Conference*, pp. 60–66, ACM, 2011.
29. C. Robotics, “V-rep user manual,” *URL http://www. coppeliarobotics. com/helpFiles/. Ultimo acesso*, vol. 13, no. 04, 2015.
30. C. Otero, E. Paz, R. Sanz, J. López, R. Barea, E. Romera, E. Molinos, R. Arroyo, L. Bergasa, and E. López, “Simulación de vehículos autónomos usando v-rep bajo ros,” 2017.
31. D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.
32. A. Sanders, *An introduction to unreal engine 4*. AK Peters/CRC Press, 2016.
33. M. Dupuis, M. Strobl, and H. Grezlikowski, “Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks,” in *Proc. of the Driving Simulation Conference Europe*, pp. 231–242, 2010.
34. M. Haklay and P. Weber, “Openstreetmap: User-generated street maps,” *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
35. M. Althoff, S. Urban, and M. Koschi, “Automatic conversion of road networks from opendrive to lanelets,” in *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pp. 157–162, IEEE, 2018.
36. W. G. Najm, J. D. Smith, M. Yanagisawa, *et al.*, “Pre-crash scenario typology for crash avoidance research,” tech. rep., United States. National Highway Traffic Safety Administration, 2007.