Multimedia Tools and Applications (2022) 81:4213–4240
https://doi.org/10.1007/s11042-021-11681-7

Train here, drive there: ROS based end-to-end autonomous-driving pipeline validation in CARLA simulator using the NHTSA typology

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]

Received: 1 February 2021 / Revised: 26 August 2021 / Accepted: 13 October 2021 /
Published online: 3 December 2021

Abstract
Urban complex scenarios are the most challenging situations in the field of Autonomous Driving (AD). In that sense, an AD pipeline should be tested in countless environments and scenarios, escalating the cost and development time exponentially with a physical approach. In this paper we present a validation of our fully-autonomous driving architecture using the NHTSA (National Highway Traffic Safety Administration) protocol in the CARLA simulator, focusing on the analysis of our decision-making module, based on Hierarchical Interpreted Binary Petri Nets (HIBPN). First, the paper states the importance of using hyper-realistic simulators, as a preliminary help to real test, as well as an appropriate design of the traffic scenarios as the two current keys to build safe and robust AD technology. Second, our pipeline is introduced, which exploits the concepts of standard communication in robotics using the Robot Operating System (ROS) and the Docker approach to provide the system with isolation, flexibility and portability, describing the main modules and approaches to perform the navigation. Third, the CARLA simulator is described, outlining the steps carried out to merge our architecture with the simulator and the advantages to create ad-hoc driving scenarios for use cases validation instead of just modular evaluation. Finally, the architecture is validated using some challenging driving scenarios such as Pedestrian Crossing, Stop, Adaptive Cruise Control (ACC) and Unexpected Pedestrian. Some qualitative (video files: Simulation Use Cases) and quantitative (linear velocity and trajectory splitted in the corresponding HIBPN states) results are presented for each use case, as well as an analysis of the temporal graphs associated to the Vulnerable Road Users (VRU) cases, validating our architecture in simulation as a preliminary stage before implementing it in our real autonomous electric car.

Keywords CARLA · Autonomous Driving · ROS · Simulation · Decision-Making · NHTSA typology

✉ Carlos Gómez-Huélamo
carlos.gomezh@edu.uah.es

Extended author information available on the last page of the article

1 Introduction

According to the World Health Organization, nearly one third of the world population will live in cities by 2030, leading to an overpopulation in most of them. Aware of this problem, the European Commission published the Transport White Paper in 2011 [21] indicating that new forms of mobility ought to be proposed to provide sustainable solutions for goods and people safety. In that sense, regarding safety, it set the ambitious goal of halving the overall number of road deaths in the EU between 2010 and 2020. However, only in 2014 more than 25,700 people died on the EU roads, many of them caused by an improper behaviour of the driver on the road. On top of that, the National Highway Traffic Safety Administration (NHTSA), US agency responsible of reducing vehicle-related crashes related to transportation safety, stated [59] that human beings are involved in 94 % of road accidents, in which an inadequate velocity, driver distraction or factors inherent to age, such as lack of sight and reflexes or joint mobility.

Considering this, Autonomous Driving (AD) have held the attention of technology enthusiasts and futurists for some time, evidenced by the continuous research and development in this field of study over the past decades, being one of the emerging technologies of the Fourth Industrial Revolution, and particularly of the Industry 4.0. Despite the fact these pipelines still present myriad scenarios with unsolved issues, the advent of Deep Learning [36], supported by the breakthroughs in Computer Vision, accumulated knowledge in vehicle dynamics and availability of new sensor modalities, datasets and hyper-realistic simulators have catalyzed [69] AD research and industrial implementation, representing one of the most challenging engineering tasks of our era. The launch of reliable and economically affordable autonomous vehicles will create a huge impact on society, affecting social (mitigating traffic jams, transporting the mobility impaired), demographic (preventing traffic accidents), environmental (reducing emissions) and economic (consuming Mobility as a Service (MaaS), reallocating of driving time, logistics revolution) aspects.

Despite all impressive efforts in the development of AD technology [13], fully-autonomous navigation in arbitrarily complex environments is still years away. The reason for this is twofold: Firstly, autonomous systems that are operated in complex dynamic environments require intelligent systems that generalizes to unpredictable situations in a timely manner. Secondly, informed decision-making requires accurate perception. Most of the existing perception systems produce errors at a rate not acceptable for autonomous driving [30].

Autonomous Vehicles (AVs) are expected to be driven throughout arbitrarily complex scenarios with a reliability greater than human beings and full autonomy, performing driving decisions based on the offline and online information processed by the vehicle. The offline information may be identified as the prior knowledge of the system, like the vehicle dynamic, traffic rules based on behavioural decision-making systems [41] or even the topological relations and geographic information of the environment based on HDMaps (High-Definition maps) [47, 40]. On the other hand, the online information, also known as the traffic situation, is obtained through the use of a global perception system of the vehicle, which involves different on-board sensors as: Inertial Measurement Unit (IMU), Light Detection And Ranging (LiDAR), RAdio Detection And Ranging (RADAR), Differential-Global Navigation Satellite System (D-GNSS), Wheel odometers or Cameras. For this purpose, a self-driving car must sense its environment and moving safely with little or even no human intervention. Considering a modular architecture [65, 39], which are the far more common than end-to-end architectures [10, 68] at the time of writing this article, the core functions of an AV pipeline may be summarized [49] as: vehicle control, localization and mapping, planning, perception, decision-making and V2U (Vehicle-to-User), sometimes referred as HMI (Human-Machine Interface). The main idea of these modular based architectures is to generate motor commands as a result of the whole stream, which start with feeding raw sensor inputs to the perception [12] and localization modules [64] to perceive the surrounding environment and estimate the ego-vehicle position in real-time. In terms of the perception module, for fully-autonomous driving architectures in complex scenarios, estimating the position of the objects in the 2D/2.5D/3D space with a very low error is insufficient. Then, identifying the static and dynamic objects of the environment (output of the tracking stage [24]), in addition to their heading angle and velocity, is mandatory as a preliminary stage of scene prediction. This, combined with the control and mapping/planning layer information, feeds the decision-making layer of the ego-vehicle in order to successfully predict the future trajectory and most probable behaviour of the dynamic obstacles to avoid collisions and plan the optimal route in the road in common urban use cases such as predicting the trajectory of a pedestrian in a crosswalk, predict the presence of an adversary vehicle in the next give-way or even evaluating the risk when carrying out the Adaptive Cruise Control (ACC) use case, in which the ego-vehicle adjusts the vehicle speed to maintain a safe distance with ahead vehicles.

The scope of this paper is the validation of our ROS based AD 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 [14] Autonomous Driving Challenge and based in the NHTSA protocol. This paper is an extension of our previous conference publication [23]. In this previous work, our results were limited to the analysis of the linear velocity given the corresponding states of the HBIPNs, whilst in the present work a more detailed analysis of temporal graph associated to the Vulnerable Road Users (VRU) use cases, in order to analyze in depth the performance of our architecture. 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 our architecture in our real electric car prototype.

The remaining content of this work is organized as follows. Section 2 presents some challenges associated to this research, focused on the decision-making layer and the importance of simulation in the context of AV, as well as a brief survey among different simulators for testing self-driving cars. Section 3 presents our autonomous navigation architecture. Section 4 describes the main features of the CARLA simulator, its integration with our AV architecture, the way in which the sensors perceive the environment and how this information is processed to feed the decision-making system, including both the prior knowledge and traffic situation. Sections 5 and 6 describes some interesting use cases, giving rise to some quantitative and qualitative results to illustrate the proposed architecture performance. Finally, Sect. 7 deals with the future works and concludes the paper.

2 Related works

As mentioned in Sect. 1, a fully-autonomous driving architecture (L5 in the J3016 SA [61]) is still years away, mainly due to technical challenges, but also due to social and legal ones [42]. The Society of Automotive Engineers (SAE) stated a taxonomy with five levels [61] of driving automation, in which the level zero stands for no automation and level five for full-automation. Level one (L1) includes primitive ADAS (Advanced Driver Assistance

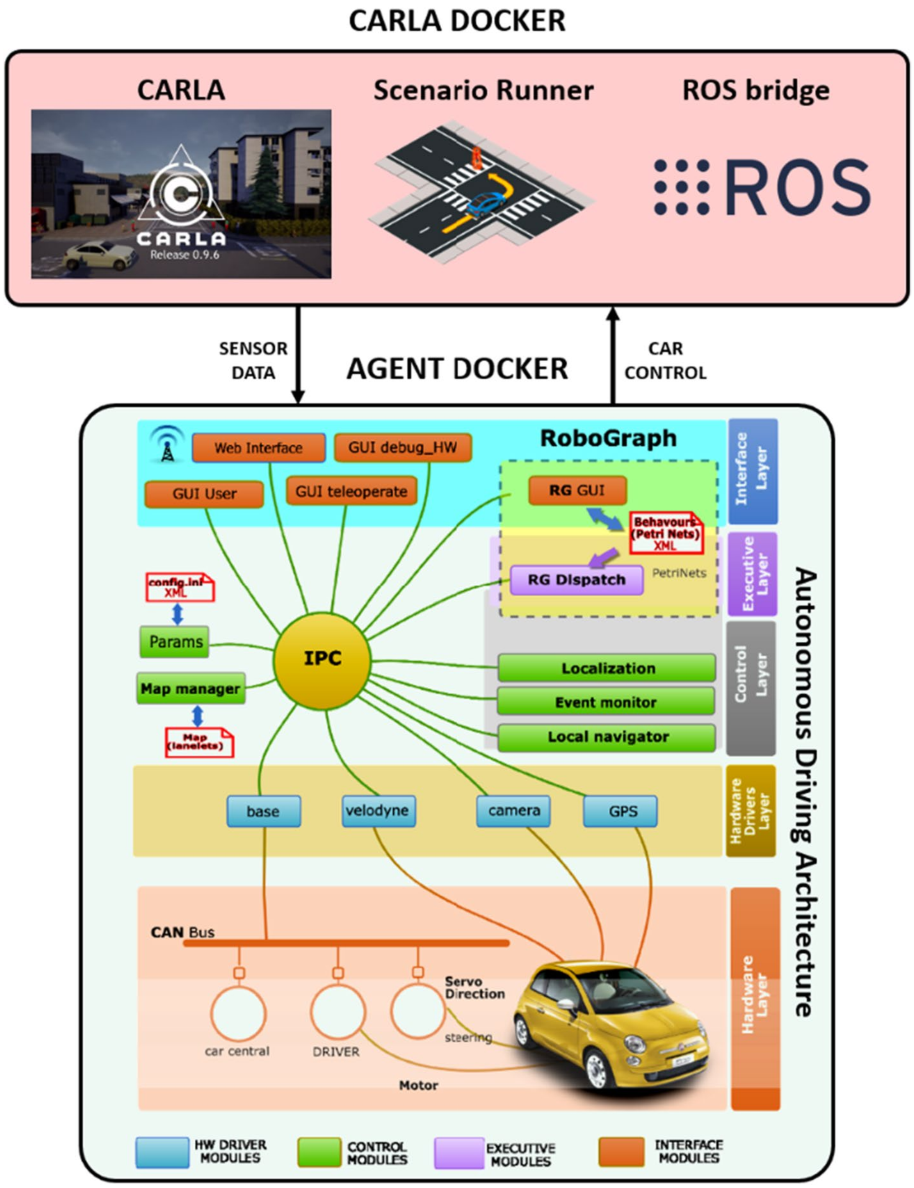


Fig. 1 Our autonomous driving architecture under CARLA simulation. GUI = Graphical User Interface; HW = Hardware; RG = RoboGraph; GPS = Global Positioning System; ROS = Robot Operating System; LiDAR = Light Detection and Ranging; IPC = Inter-Process Communication

Systems) such as Adaptive Cruise Control, stability control or anti-lock braking systems [52]. Once the system has longitudinal and lateral control in a specific use case (although the driver has to monitor the system at all times), level two (L2) becomes a feasible technology. Then, the real challenge arises above this level. Level three (L3) is conditional automation, that is, the system has longitudinal and lateral control in specific use cases, so the driver does not have to monitor the system at all times. However, the system recognizes the performance limits and the driver is requested to resume control (s/he must be in position) within a sufficient time margin. In that sense, the takeover maneuver (transition from automatic to manual mode) is an issue yet to be solved. Since recent studies [20, 43] have demonstrated how it increases the likelihood of an accident during the route, in particular if the driver is not aware of the navigation.

Besides this, we find levels four and five, in which human attention is not required anymore in specific use cases (L4) or any weather condition and road conditions (L5), which represent an open and challenging problem. The environment variables, from surrounding behaviour to weather conditions, are highly stochastic and difficult to model. In that sense, no industry organization has shown a ratified testing methodology for L4/L5 autonomous vehicles. The autonomous driving community gives a simple reason: despite the fact that some regulations have been defined for these levels and current automotive companies/research groups are very good at testing the individual components of the AD architecture using the corresponding datasets [19, 9, 70], there is a need to test intelligent vehicles full of advanced sensors [57] in an end-to-end way. In this context, artificial intelligence is increasingly being involved in processes such as detecting the most relevant objects around the car (DL 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 establish a safe navigation in such a way the different sensors and associated algorithms are integrated together, to validate the whole system and not just individual components.

Regarding urban environment complexity, in order to validate a whole AD architecture the system must be tested in countless environments and scenarios, which would escalate the cost and development time exponentially with the physical approach. Considering 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. These simulators have evolved from merely simulating vehicle dynamics to also simulating more complex functionalities. Simulators intended to be used for test self-driving technology must have requirements that extend from simulating physical car models to several sensor models, path planning, control and so forth and so on. Some state-of-the-art simulators [33] for testing self-driving vehicles are as following:

– **MATLAB/Simulink** [38] published its Automated Driving Toolbox, that provides several tools which facilitate the design, simulation and testing of automated driving systems and Advanced Driver Assistance Systems (ADAS). One of its key features is that OpenDRIVE [15] road networks can be imported into MATLAB and may be used for varios testing and design purposes. This Automated Driving toolbox also supports Hardware-in-the-Loop (HIL) testing as well as C/C++ generation, which enables faster prototyping.
– **CarSim** [6] is a vehicle simulator commonly used by academia and industry. Its newest version supports moving objects and sensors that benefit simulations involving self-driving tecnology and ADAS. These moving objects may be linked to 3D objects with their own embedded animations, such as vehicles, byciclists or pedestrians.
– **PreScan** [63] provides a simulator framework to design self-driving cars and ADAS. It presents PreScan's automatic traffic generator which enables manufacturers to validate their autonomous navigation architectures providing a variety of realistic environments

and traffic conditions. This simulator also supports HIL simulation, quite common for evaluating Electronic Control Units (ECUs) used in real-world applications.

– **CARLA (CAR Learning to Act)** [14] is an open-sources simulator which main objective is to democratize the autonomous driving research area. It is developed based on the Unreal Engine [56], consisting of a scalable client-server architecture in which the the simulation tasks are deployed at the serve, including the updates on the world-state and its sensor rendering, computation of physics, actors, etc.
– **Gazebo** [35] is an scalable, open-source, flexible and multi-robot 3D simulator. It supports the recreation of both outdoor and indoor environments in which there are two core elements that define the 3D scene, also known as world and model. The world is used to represent the 3D scene, defined in a Simulation Description File (SDF) and a model is basically any 3D object. Gazebo uses Open Dynamic Engine (ODE) as its default physic engine.
– **LGSVL (LG Electronics America R&D Center)** [55] is the most recent simulator for testing autonomous driving technology, focused on multi-robot simulation. It is based on the Unity game engine [26], providing different bridges for message passing between the simulator backbone and the autonomous driving stack. LGSVL provides a PythonAPI to control different environment entitities, such as weather conditions, the position of the adversaries, etc. in a similar way to the CARLA simulator. It also provides Functional Mockup Interface (FMI) so as to integrate vehicle dynamics platform to the external third party dynamics models.

In order to choose the right simulator, there is a set of criteria [33] that may serve as a metric to identify which simulators are most suitable for our purposes, such as perception (sensors), multi-view geometry, traffic infrastructure, vehicle control, traffic scenario simulation, 3D virtual environment, 2D/3D groundtruth, scalability via a server multi-client architecture and last but not the least, open-source. In that sense, we identify that MATLAB/Simulink is designed to simulate simple scenarios, with efficient plot functions and computation. It is usually connected to CarSim, where the user can control the vehicle models from CarSim and build their upper control algorithms in MATLAB/Simulink to do a co-simulation project, but the realism, the quality of the sensors and the complexity is limited. PreScan presents better capabilities to build realistic environments and simulate different weather conditions, unlike MATLAB and CarSim. Gazebo is quite popular as a robotic simulator, but the effort and time needed to create complex and dynamic scenes does not make it the first choice for testing self-driving technology. Then, we have two simulators as our final options: LGSVL and CARLA. At the moment of writing this paper, they are the most suited simulators for end-to-end testing of unique functionalities offered by autonomous vehicles, such as perception, mapping, vehicle control or localization. Most of their features, summarized in [33] are identical (open-source, traffic generation simulation, portability, 2D/3D groundtruth, flexible API and so forth and so on), with the only difference that LGSVL does not present camera calibration to perform multi-view geometry or Simultaneous Localization and Mapping (SLAM). In that sense, we decided to use the CARLA simulator since the performance is very similar to LGSVL and the group had previous experience in the use of this simulator. For a deeper understanding about how we integrate our architecture with this simulator, we refer the reader to Sect. 4.

Moreover, as stated in Sect. 1, this work focuses on the behavioural decision-making layer of the pipeline and its relation with the end-to-end validation. The 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 [65], that lie in 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 (POMDPs) [37] offer a great framework to manage uncertainty, they are not scalable to real-world scenarios due to 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) [16], Hierarchical Finite-State Machines (HFSM) [3] and Finite-State Machines (FSM) [46] are among the most popular approaches to design a decision making system. Several teams in the Defense Advanced Research Projects Agency (DARPA) Urban Challenge [41] used some of these approaches.

The approach used by the winning team (Tartan team) was a HFSM to implement their behaviour generation component. Then, the main task is hierarchically broken down into a set of top-level hierarchy behaviours, identified with intersection handling, lane-driving and achieving a pose [65].

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, simple discrete event systems and BTs 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) [18, 22], as shown in Fig. 2b, where a PN can start/stop another PN depending on its hierarchy.

3 Autonomous navigation architecture

To accomplish the AD task, we present an architecture (Fig. 1) that exploits the concepts of portability, flexibility and isolation in terms of software development using Docker [44] containers and standard communications in robotics using the Robot Operating System (ROS) [51]. It 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:

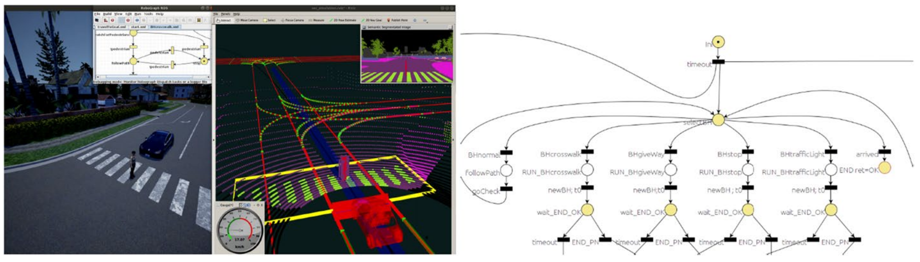


Fig. 2 (**a**) Simulation example: On the left, the CARLA simulator illustrates the urban scenario our sensors face. On the right, the RVIZ simulator shows the detection in the coloured point cloud (**b**) Start Petri Net of our behaviour decision-making system

- **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.

Furthermore, our workflow is summarized in Fig. 3. In order to implement and validate new algorithms for some specific use case in our real-world autonomous electric car, we first evaluate not only the proposals of each individual layer, but also the whole architecture, which is actually the real purpose of automotive companies. We employ both Version Control (Git) and lightweight Linux containers (Docker) for consistent development to ensure the transfer learning between simulation and the real-world. Our motion control is broken down into high-level (HL) planning and low-level (LL) reactive control. First of all, as observed in Figs. 3 and 1, the map manager (control layer) loads the map made up by a sequence of lanelets [5], in which the user must specify the start and goal position of the route. Then, a lanelet path is generated using an A* algorithm [8]. Besides this, the map manager serves other queries from other modules related to the map, such as providing the most relevant lanes around the vehicle (also referred as monotorized lanes), like 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. Since the A* algorithm belongs to the goal-directed category ([2] differentiates route planner into four categories: goal-directed, separator-based, hierarchical and bounded-hop, providing the fastest route but not the most desirable one, since no road hierarchies or precomputed distances between selected vertexes are taken into account. In that sense, a local navigation module is used to safely follow this path, keeping the vehicle within the driving lane and modifying the vehicle dynamic as a function of the behaviour constraints established by the HL planning. To do that, the local navigation system calculates the curvature to guide the car from its current position to a look-at-head position

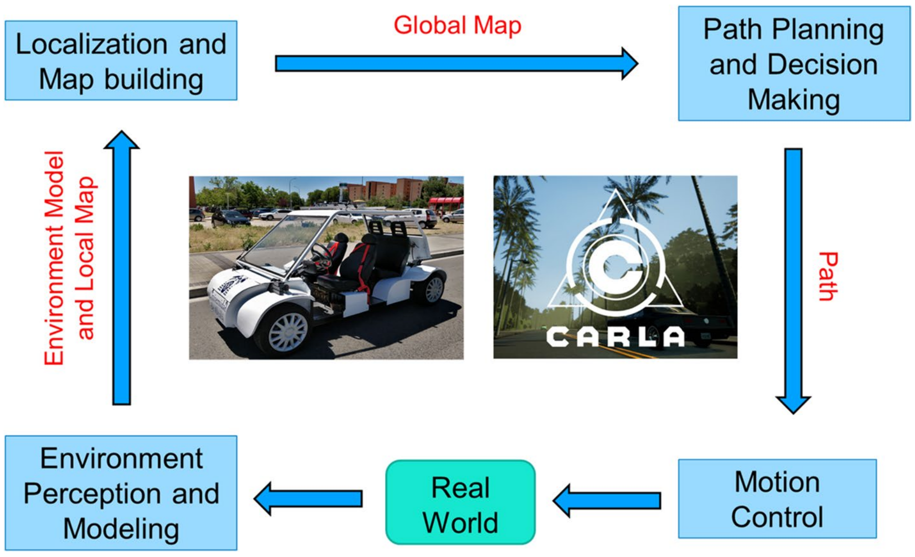

Localization and Map building
Global Map
Path Planning and Decision Making
Environment Model and Local Map
CARLA
Path
Environment Perception and Modeling
Real World
Motion Control

Fig. 3 Workflow to develop safe AD technology. Version Control (Git) and lightweight Linux containers (Docker) are employed for consistent development and deployment to ensure the transfer learning between simulation and the real-world

placed in the center of the lane using the Pure Pursuit algorithm [34]. 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) [17]. Combining both the Pure Pursuit algorithm and BCM, the local navigation system keeps the vehicle centered in the driving lane while is able to avoid unknown obstacles that can partially block the lane.

In terms of environment perception, we perform a sensor fusion [22] between LiDAR and camera in order to detect and track the most relevant objects in the scene. First, a semantic segmentation is performed in the RGB image to detect the different obstacles and the drivable area. In the present work we use the groundtruth of the semantic segmentation provided by CARLA pseudo-sensor instead of using some algorithm to process the RGB camera and obtain the segmented image. We will conduct this preprocessing step in future works. In the present work we focus on validating the autonomous navigation architecture and its relationship with the decision-making layer. We merge the 3D LiDAR point cloud information and the 2D segmented pixels so as to obtain a coloured point cloud, where the points out of the FoV (Field of View) of the camera are not coloured. Then, we carry out a coloring based clustering [28], obtaining the most relevant objects in the scene, as shown in Fig. 2a. Then, Multi-Object Tracking is performed by combining the Precision Tracker approach [28], BEV (Bird's Eye View) Kalman Filter [32] and Nearest Neighbour algorithm [4]. The output of the perception module is then published through the event monitor module (see Fig. 1).

Finally, the behavioural decision-making is implemented using HIBPNs, where the main Petri Net is fed with the inputs provided by the local perception (event monitor module), ego-vehicle dead-reckoning in the map and the map manager information. To implement these HIBPNs, we use the RoboGraph tool [41], employed by the authors in other mobile robot applications.

4 Simulation stage

Fatalities caused by immature technology (software, hardware or integration) undermine public acceptance of self-driving technology. In 2014, in the Hyundai competition one of the teams crashed due to the rain [62]. In 2016, the Google car hit a bus [11] while conducting the lane changing maneuver since it failed to estimate the speed of the bus. From L3 to L4 the vehicles goes from human centered autonomy, in which artificial intelligence is not fully responsible, to a stage in which the implemented software and hardware are fully responsible in each particular real-world situation. Regarding this, several ethical dilemmas arise. In the event of an accident, the legal responsibility would lie with the automotive company rather than the human driver. Moreover, in an inevitable accident situation, which should be the most accurate system behaviour? Then, reliability certification and risk is another task to be solved. Some promising works have been proposed such as DeepTest [62], but a workflow of *design* $\longrightarrow$ *simulation* $\longrightarrow$ *test* $\longrightarrow$ *redesign* has not been established by the automotive industry yet. Then, it can be observed that, a fully-autonomous driving architecture (L5 in this classification) is still years away, not only due to technical challenges, but also due to social and legal ones. To solve this challenge, virtual testing is becoming increasingly importance in terms of AV, even more so considering L3 or higher architectures, in order to provide an endless generation of traffic scenarios based on real-world behaviours. 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 [58], recently updated to include AV although though it was initially designed for drones, NVIDIA DRIVE PX [7], aimed at providing AV and driver assistance functionality powered by deep learning, ROS development studio [45], fully based on the cloud concept where a cluster of computers allows the parallel simulation of as many vehicles as required, V-REP [54], with an easy integration with ROS and a countless number of vehicles and dynamic parameters and CARLA [14], which is the newest open-source simulator for AV based on Unreal engine. In [22] we validated the proposed navigation architecture using the V-REP simulator due to the experience of our research group with small mobile robots. However, the paradigms of real-time and high realism could not be deeply analysed since V-REP was not designed to create high realistic urban driving scenarios, in such a way the analysis of traffic use cases is difficult. 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 to design a reliable AV pipeline. In this work we have used the 0.9.6 version of CARLA and the corresponding versions of the CARLA ROS Bridge and the Scenario Runner. Additional tools needed for the simulation, were configured according to this version. In our previous work [22], we run the simulation environment, the V-REP ROS bridge and the autonomous navigation architecture in the same host machine, running on Ubuntu 16.04. On the contrary, in the present work we exploit the concept of lightweight Linux containers using Docker [44] images, where the first image contains the CARLA world and CARLA ROS Bridge information and the second image contains our autonomous navigation architecture, as shown in Fig. 1.

4.1 Environment

CARLA is implemented as an open-source layer over Unreal Engine 4 (UE4) [56] graphic engine. 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 [15] format, whilst we use the lanelet approach based on OpenStreetMap (OSM) [27] service. We transform the OpenDrive format into OSM format by using the converter proposed by [1]. 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.

4.2 Vehicle

In order to link the vehicle in CARLA and its corresponding model in RVIZ (Fig. 1a), we modify the ROS bridge associated to the CARLA simulator. The CARLA ROS bridge is a set of algorithms 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 acceleration and speed PIDs (Proportional Integral Derivative) controllers, the sensors position, orientation and resolution, 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 that 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.

4.3 Sensors

From the sensors perspective, the agent sensor suite can be modified in a flexible way, being possible to modify both the number of sensors and their associated features, such as the position, orientation, resolution or frequency. Most common sensors in CARLA world are LiDAR, GNSS 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 CARLA ego-vehicle on-board sensors had a 800 × 600 RGB camera sensor placed at the front of the vehicle with a FoV of 100 °, a 32-channels LiDAR, a collision sensor, a GNSS 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. LiDAR information is published in PointCloud2 ROS format using the bridge 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. As mentioned above, in order to colour the 3D point cloud and perform coloring clustering, we project the semantic segmentation into the cloud. This semantic segmentation is carried out based on a 1280 × 720 RGB camera as input, which is the resolution of the ZED camera equipped in our real-world electric car. Moreover, the position of the GNSS 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 NHTSA based use cases

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 ScenarioRunner repository, associated to the CARLA GitHub. The ScenarioRunner is a module that allows the user to define and execute traffic scenarios for the CARLA simulator. In terms of crash typologies, there are several New Car Assessment Programs (NCAPs), which are a set of protocols to evaluate the safety of vehicles. Currently, evaluations are focused on the structure of vehicle and Advanced Driver Assistance Systems (ADAS), such as: Adult Occupant Protection (AOP), Child Occupant Protection (COP), Autonomous Emergency Braking (AEB), Speed Assist Systems (SAS), etc. Crash typologies provide an understanding of distinct crash types and scenarios and explain why they occur. They serve as a tool to identify intervention opportunities, set research priorities and direction in technology development, and evaluate the effectiveness of selected crash countermeasure systems. The most important NCAPs, at the moment of writing this paper, are:

- European New Car Assessment Program (Euro-NCAP): Founded in 1997, is the most widely used NCAP, within the scope of the collaboration of the countries of the European Union [66].
- China New Car Assessment Program (C-NCAP): Research and development benchmark for vehicle manufacturers in Asia, founded in 2006. Most of its program is based on the Euro-NCAP [25].
- National Highway Transportation Safety Administration (NHTSA): Agency of the United States of America federal government, part of the Department of Transportation, founded in 1970. They have published research reports, guidance documents, and regulations on vehicles equipped with ADAS [60].
- Autoreview Car Assessment Program (ARCAP): is the first independent evaluation program for cars in Russia. They publish their studies in a newspaper called Autoreview [29].

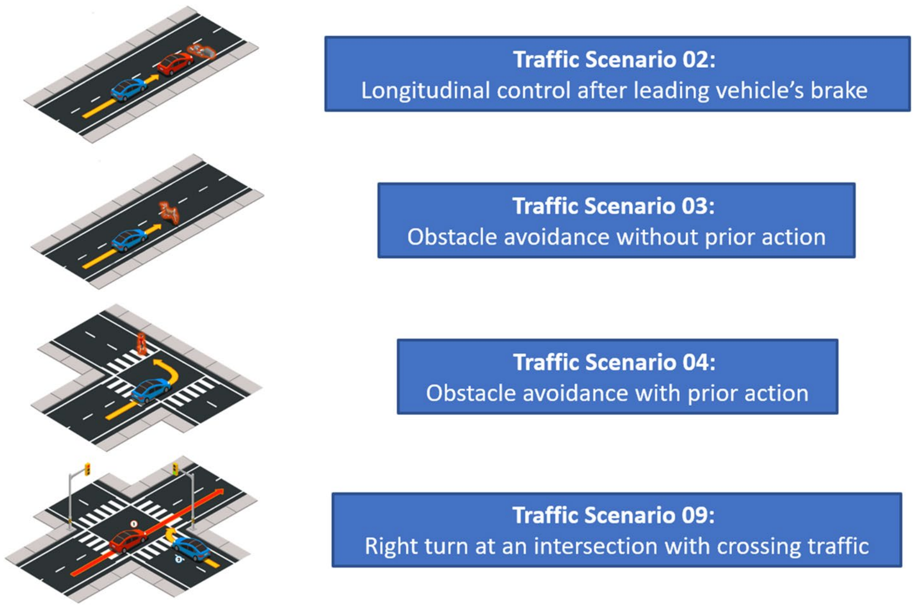

Traffic Scenario 02:
Longitudinal control after leading vehicle's brake
Traffic Scenario 03:
Obstacle avoidance without prior action
Traffic Scenario 04:
Obstacle avoidance with prior action
Traffic Scenario 09:
Right turn at an intersection with crossing traffic

Fig. 4 Our selected CARLA Autonomous Driving challenge traffic scenarios, based on the pre-crash scenario typology for crash avoidance defined by the NTHSA organization

In the present case, the scenarios are inspired in the CARLA Autonomous Driving Challenge (CADC) 4, selected from the NHTSA pre-crash typology [48], which can be defined using a Python interface or the OpenSCENARIO [31] standard. Then, the user can prepare its AD pipeline (also referred as the agent) for evaluation by creating complex routes and traffic scenarios with 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 conditions and so on. 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). Below the particular HIBPNs covered in this paper are explained from the behavioural decision-making layer perspective.

5.1 Stop behaviour

Intersections represent some of the most common danger areas on the road, since they pose many challenges as vehicles must deal with different maneuvers such as traffic signals, turning lanes, merging lanes, and other vehicles in the same area. Intersections are highly varied, and some require the car to stop, while others only require a give way.

In our case, this behaviour (Fig. 5) is triggered if the distance between the Stop regulatory element and our ego-vehicle is equal or lower than a certain threshold D_{re} (that stands for distance to regulatory element, usually 30 m). Then, the perception system will try to check if there is some car in the lanelets that intersect with our current trajectory. If no car is detected, the car resumes the path, not modifying the current trajectory. Nevertheless, even if no car is detected and the distance to the reference line (position of the regulatory element) is lower than a more restrictive threshold D_{rl} (that stands for

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; **N** = Nodes; **T** = Transitions

Petri Net	Inputs	Input modules	Outputs	Output modules	N / T
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			

distance to reference line, usually 10 m), due to the traffic rules of the stop behaviour, the ego-car must always stop, so the ego-vehicle sends a stop command to the local navigation module. At this point, while waiting for the car to stop, the messages regarding the car velocity from the local navigation module will be received until the speed reaches zero (stopped transition). After waiting a few seconds for safety, it will proceed to follow the path if the event monitor publishes a message indicating that it is safe to merge. While merging, when the car reaches the intersection, it considers that the safest maneuver to do is to continue and it ends the behavior.

On the other hand, if a vehicle is detected before reaching the intersection, a corresponding message is received from the event monitor and the *safeMerge* transition is set to 0. In this case, the car will stop (stop place) sending the corresponding stop message to the local navigator and waits until the event monitor module reports a *safeMerge* message. After stopping in front of the reference line, and waiting for the detected car to take out of our perception system, the car will proceed to keep on the path and finish the behaviour.

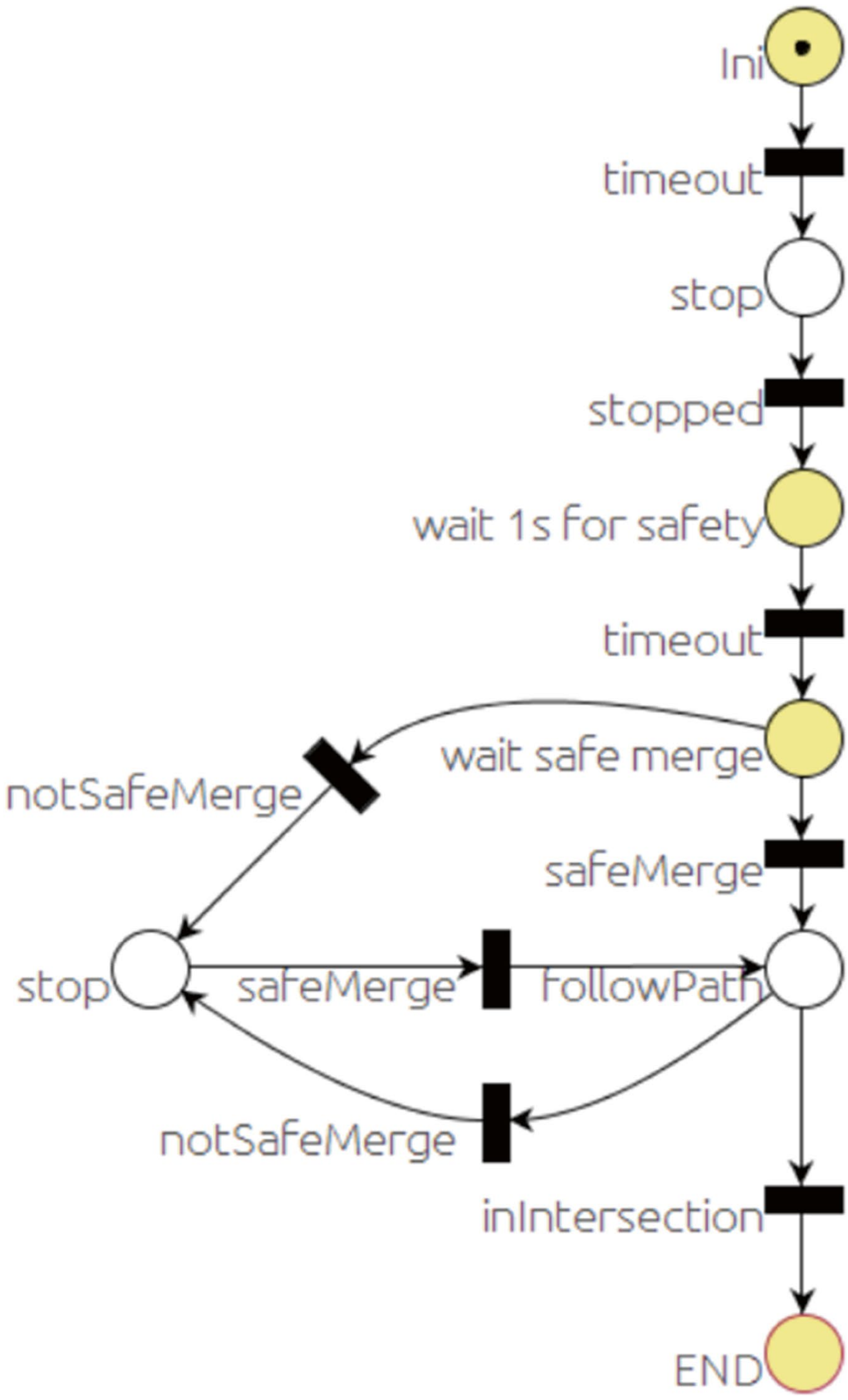


Fig. 5 Stop Petri Net

5.2 Pedestrian crossing behaviour

The pedestrian crossing use case (Fig. 6), also referred as crosswalk use case in the literature, is the most basic regulatory element that helps people to cross a road. It is started when the ego-vehicle is approaching an area with the corresponding regulatory element and the distance to the regulatory element is lower than a certain threshold D_{re} (usually 30 m), in a similar way to the Stop behaviour. Then, the node *watchForPedestrians* sends a message to the event monitor module to watch for pedestrians crossing or trying to cross on the crosswalk. The *watchForPedestrians* output transition includes three possibilities:

- **Non-pedestrian detected** A confirmation from the event monitor notifying that it received the message and so far no pedestrian has been detected (*!pedestrian* transition) in the pedestrian crossing area.
- **Pedestrian detected** A pedestrian has been detected (*pedestrian* transition) in the pedestrian crossing area.

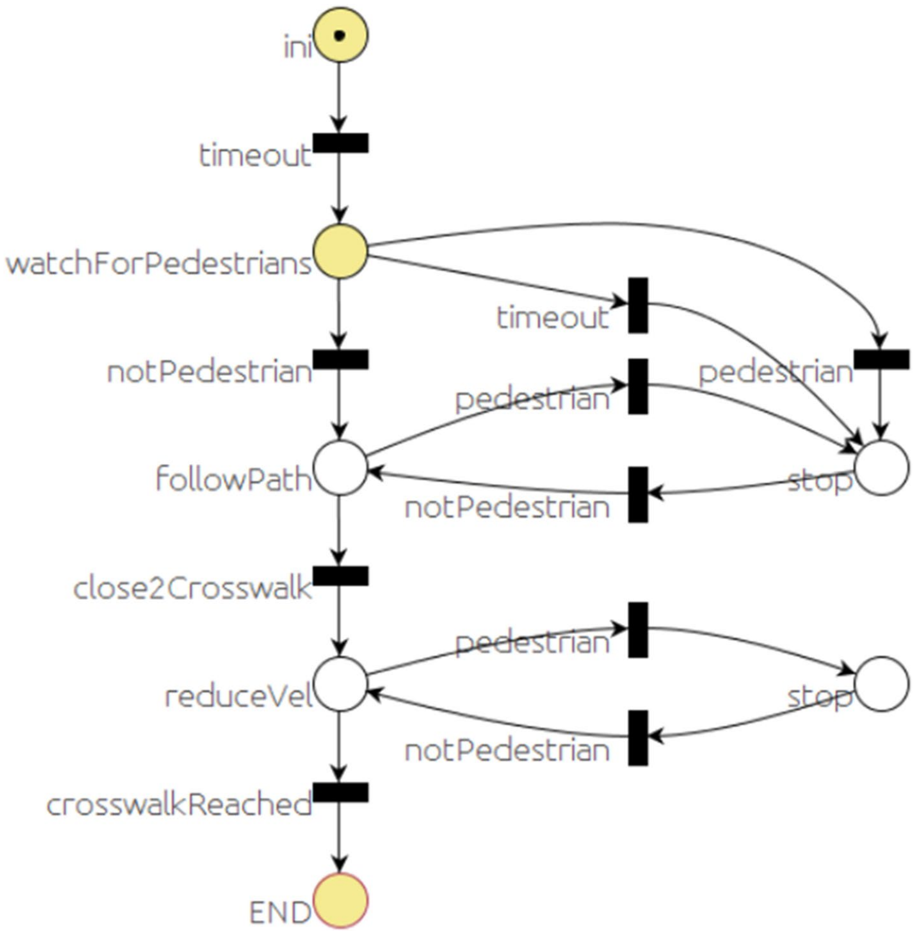


Fig. 6 Pedestrian Crossing Petri Net

- **Non-confirmation** There has been some problem and no confirmation has been received from the event monitor, so a timeout message is received and the car must stop its trajectory.

While the current state is *followPath*, the local navigation module will keep following the lane provided by the map manager module. At this moment, the system is receptive to two events: *pedestrian* and *distToRegElem* $< D$.

The event *pedestrian* corresponds to a message published by the event monitor module: If this message is received, the pedestrian transition will be fired, switching the PN from *followPath* node to *stop* node. The transition *distToRegElem* $< D_{re}$ takes place when the car is closer than a threshold distance D_{re} (usually 30 m) to the crosswalk. In this case, every time the localization module publishes a new position message, the distance between the car position and the pedestrian crossing (in particular its reference line) is calculated and compared with this threshold D_{re}.

When the car gets close to the pedestrian crossing, it reduces its velocity, even if there is no pedestrian presence, and keeps following the path (*reduceVel* node) until the reference line of the pedestrian crossing is reached (*distToRegElem* $< D_{rl}$ transition). Meanwhile, if a pedestrian is detected by the event monitor module, the message issued by this module will fire the transition labeled *pedestrian*. The Petri net will switch the token to the second *stop* node, where a stop message will be sent to the local navigation module, forcing the vehicle to stop in front of the reference line of the pedestrian crossing. The Petri net will leave the stop mode when a notPedestrian message is received (*!pedestrian* transition), meaning that the car can resume its path, ending the Petri net and allowing the Selector PN to enable another behaviour in the trajectory, if required.

Furthermore, as observed in Fig. 6b, no high-level behaviours are launched with the Adaptive Cruise Control (ACC) or Unexpected Pedestrian use cases since they do not depend of the presence of a regulatory element, but they are always running in the background of the Selector PN, that is, always in execution. The Adaptive Cruise Control (ACC) is a behaviour that represents an available cruise control system for vehicles on the road, automatically adjusting the vehicle velocity to keep a safe distance to ahead vehicles. As mentioned in Sect. 1, it is a well mature technology, widely regarded as a key component of any future generations of autonomous vehicles. It improves the driver safety as well as increasing the capacity of roads by maintaining optimal separation between vehicles an reducing driver errors. On the other hand, the Unexpected Pedestrian is a special use case consisting on a pedestrian, bicycle, etc. jumps into the lane without the presence of a pedestrian crossing. Then, an emergency break must be conducted until the car is stopped in front of the obstacle to ensure safety, and resumes the navigation once the obstacle leaves the driving lane.

6 Experimental results

In our previous work [22] we study the linear velocity and the odometry of the ego-vehicle throughout the following ScenarioRunner use cases: Stop with car detection, Stop with no car detection, Pedestrian Crossing, Adaptive Cruise Control (ACC) and Unexpected Pedestrian, inspired in the Traffic Scenario 09 (Right turn at an intersection with crossing traffic), Traffic Scenario 04 (Obstacle avoidance with prior action), Traffic Scenario 02 (Longitudinal control after leading vehicle's brake) and Traffic Scenario 03 (Obstacle avoidance

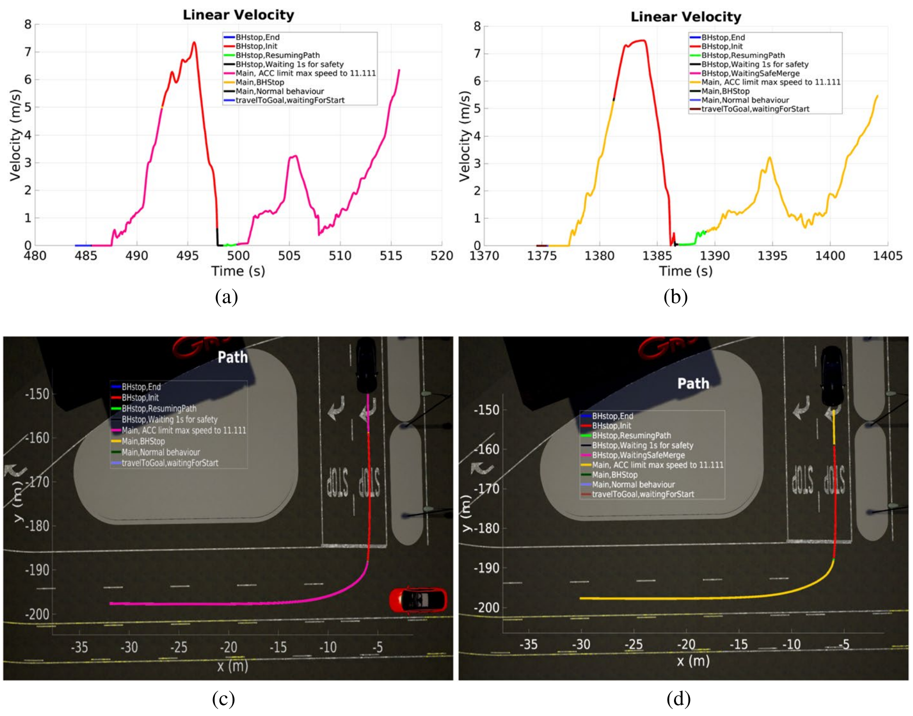


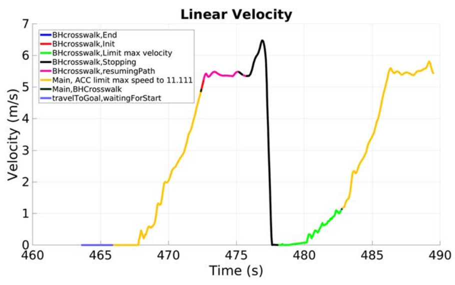
Fig. 7 First and second row represent, respectively, the linear velocities and described trajectory projected onto the corresponding CARLA scenario for Stop with car detection (**a**, **c**) and Stop with no car detection (**b**, **d**) behaviours

without prior action) of the CADC respectively. Furthermore, in this work we focus on the analysis of Vulnerable Road Users (VRU), in particular the pedestrian crossing and unexpected pedestrian, through the analysis of their corresponding temporal graphs. A video demonstration for each use cases can be found in the following play list Simulation Use Cases[1].

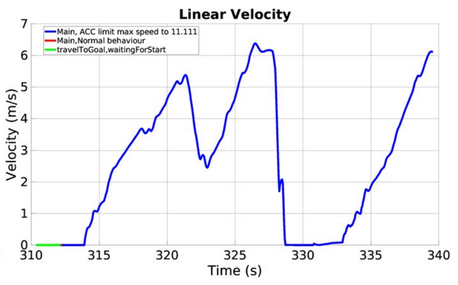
6.1 Stop use case

The resulting graphics are segmented using different colours, corresponding to the current node of the associated PN. Figure 7 shows the two different situations regarding the Stop use case: with and without the presence of an adversary vehicle. The vehicle carries out the Stop Petri net (*BHstop, init*) once the distance to the regulatory element is lower than 30 m. Then, the car reduces its velocity from 11 m/s (most common urban speed limitation) to 0 m/s, and wait for a safe situation (*BHstop*). In that sense, 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. 7(b) and (d)), which is quite similar to time that the ego-vehicle waits with the presence of an adversary vehicle (Fig. 7(a) and (c)). This behaviour is coherent, since the

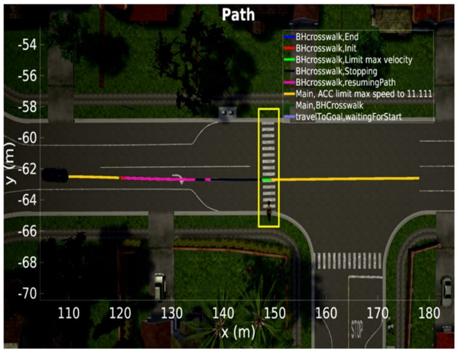
[1] Simulation Use Cases link: https://cutt.ly/prUzQLi



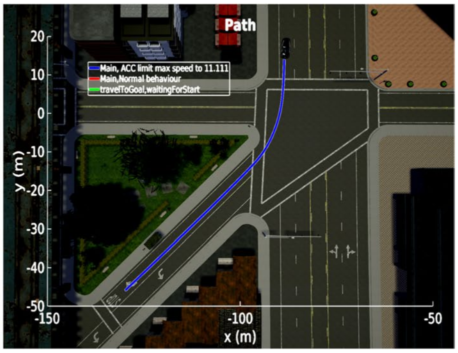
(a)



(b)



(c)


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

(d)

Fig. 8 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

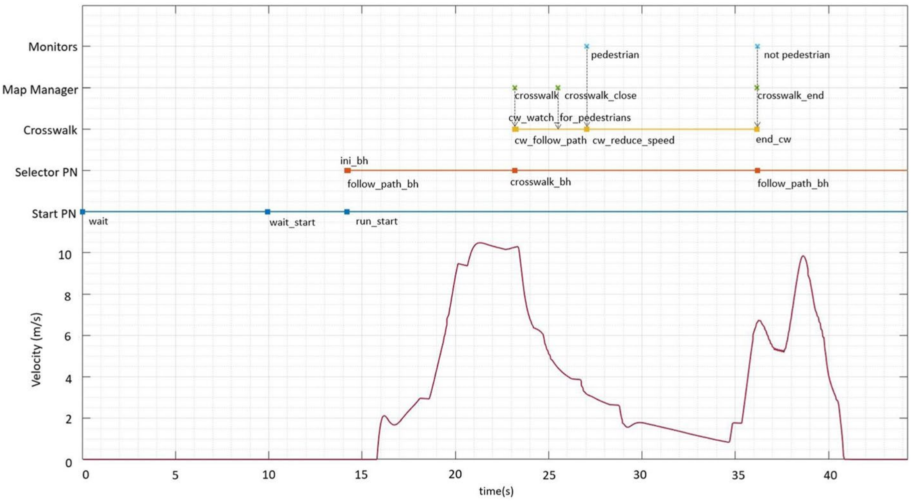


Fig. 9 Pedestrian Crossing Temporal diagram. At the top, the events produced by the monitors and map manager modules. In the middle, the evolution of the pedestrian crossing use case, selector, and start (background) PNs. At the bottom, the velocity of the car throughout the route

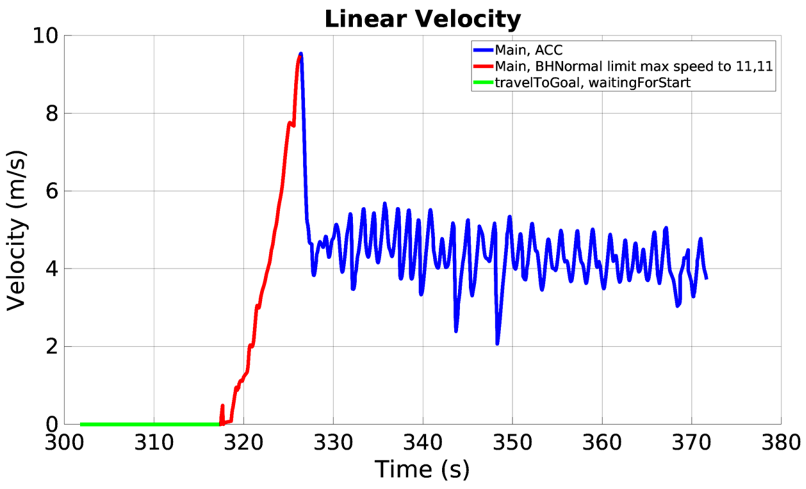
Fig. 10 Analysis of the ACC use case with variable adversary vehicle velocity. (**a**) represents the ego-vehicle ▶ linear velocity, (**b**) analysis of the road distance to the adversary throughout the use case, (**c**) the ego-vehicle odometry projected onto the CARLA world

Stop PN presents a transition that requests 0 m/s for the ego-vehicle in addition to *n* extra seconds to wait for safety, in the same way that a real-world car would do, which is usually enough time for the adversary to leave the intersection lane and enable again the navigation (*BHstop, ResumingPath*) until the car faces another regulatory element.

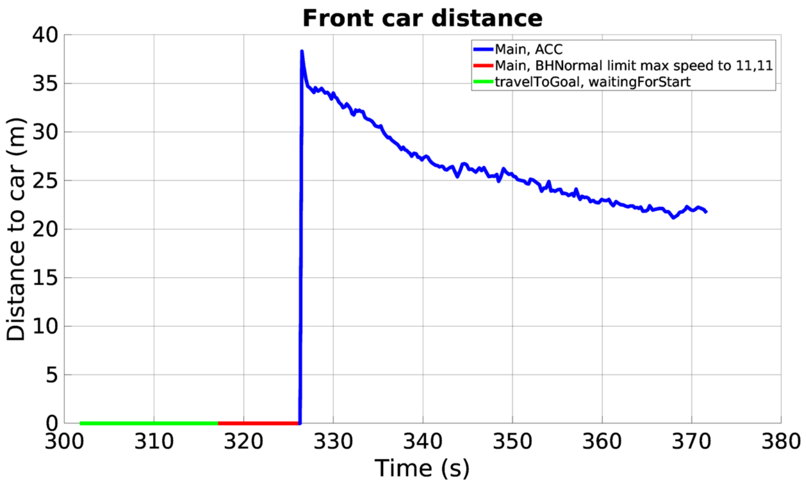
6.2 Pedestrian crossing use case

Figure 8(a) and (c) show the linear velocity and odometry projected onto the CARLA simulator associated to the pedestrian crossing behaviour as it was presented in our previous publication [22]. Figure 9 represents a temporal diagram with the sequence of events, evolution of the PNs states and the actual velocity of the car throughout the same scenario for a new experiment. Note that this diagram does not exactly match with their corresponding linear velocities and ego-vehicle odometry shown in Fig. 8 since corresponds to a different experiment with different stochastic processes involved on it.

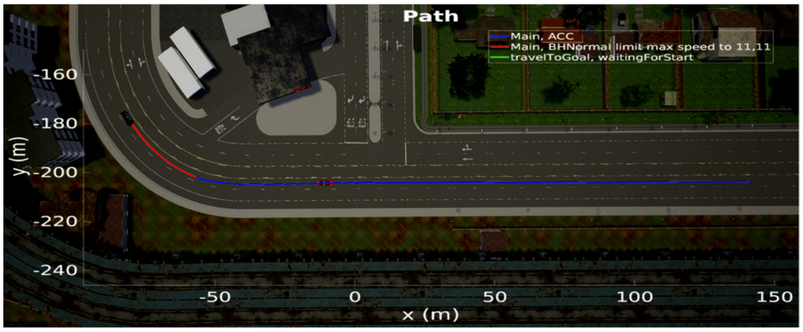
The incorporation of this temporal diagram, also presented in [41], is a powerful manner to validate the architecture in an end-to-end way, since we can observe how the car behaves considering the different actions and events provided by the executive layer, which is actually the output of the whole architecture before sending commands to the motor. The scenario starts with the vehicle stopped 40 meters away from the regulatory element. Since this distance exceeds the previously defined threshold D_{re}, the velocity is set at maximum at the beginning (*Main, ACC limit max speed to 11,111*). As observed in Fig. 8(c), an additional safety area (yellow rectangle) is defined around the pedestrian crossing, in order to activate the pedestrian detection, not only considering the lane but also a small area of the sidewalk. Note that the shown velocity is not the velocity commanded by the control layer, which is usually more homogeneous, but it is the actual velocity of the ego-vehicle, considering the physic constraints of the car. About second 18, a message from the map manager module indicates the proximity of the pedestrian crossing regulatory element which triggers the pedestrian crossing behaviour ($cw_{followpath}$). Then, as illustrated above, once the distance is lower than the more restrictive threshold D_{rl}, represented as the transition *close2CrossWalk*, the executive layer indicates the reactive control to gradually stop in front of the pedestrian crossing reference line. The reason for this is simple: A good practice when driving is to reduce the velocity of the vehicle in the proximity of a regulatory element, furthermore if it is a pedestrian crossing, to safely conduct the maneuver, even without the presence of a pedestrian in the monitorized area. In our case, above second 22 a pedestrian is detected, so a *stop* signal is sent to the reactive control, which actually does not need to abruptally change the velocity of the car since it is already moving slowly. It is important to note that the vehicle must stop in front of the reference line if there is one obstacle in the monitorized area Fig. 2(a). In this case, the pedestrian leaves the relevant lane (second 30) before the vehicle reaches the line, so it does not reduce its velocity at all. Once the reference line is over (second 31), the pedestrian crossing behaviour is finished and the vehicle continuous the route.


Linear Velocity
Main, ACC
Main, BHNormal limit max speed to 11,11
travelToGoal, waitingForStart
Velocity (m/s)
Time (s)
0
2
4
6
8
10
300
310
320
330
340
350
360
370
380

(a)


Front car distance
Main, ACC
Main, BHNormal limit max speed to 11,11
travelToGoal, waitingForStart
Distance to car (m)
Time (s)
0
5
10
15
20
25
30
35
40
300
310
320
330
340
350
360
370
380

(b)


Path
Main, ACC
Main, BHNormal limit max speed to 11,11
travelToGoal, waitingForStart
y (m)
x (m)
-160
-180
-200
-220
-240
-50
0
50
100
150

(c)

6.3 Adaptive Cruise Control (ACC) use case

Regarding the Adaptive Cruise Control (ACC) use case Fig. 10, we faced several problems simulating this behaviour in V-REP since this environment 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. In the present case, we take advantage of the possibilities offered by CARLA, in particular ScenarioRunner, to configure the spatiotemporal variables of the adversary in this use case. In that sense, Fig. 10(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. It can be observed that the ACC behaviour starts at the exact moment in which the adversary vehicle is detected. Moreover, Fig. 10(c) shows how the road distance to the vehicle is decreased till 22/23 m, where the ACC is kept until the traffic scenario is concluded. It is important to note that we do not calculate the distance Fig. 10(c) to the adversary as a front distance (X-frame considering LiDAR frame) or even the Euclidean distance, since it could lie in the false hypothesis of always carrying out this scenario in a straight lane, but we calculate the distance to the obstacle calculating the nearest node of the lane, and then the accumulated position from our global position to that node taking into account the intermediate nodes of the lane.

6.4 Unexpected pedestrian use case

The last use case we validate in this paper is inspired in the Traffic Scenario 03 (Obstacle avoidance without prior action) of the CADC, which is one of the key scenario to validate VRU based behaviours. In this traffic scenario, the 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 on the road, being initially totally occluded by a bus marquee. As expected, Fig. 8(b) and (d) show

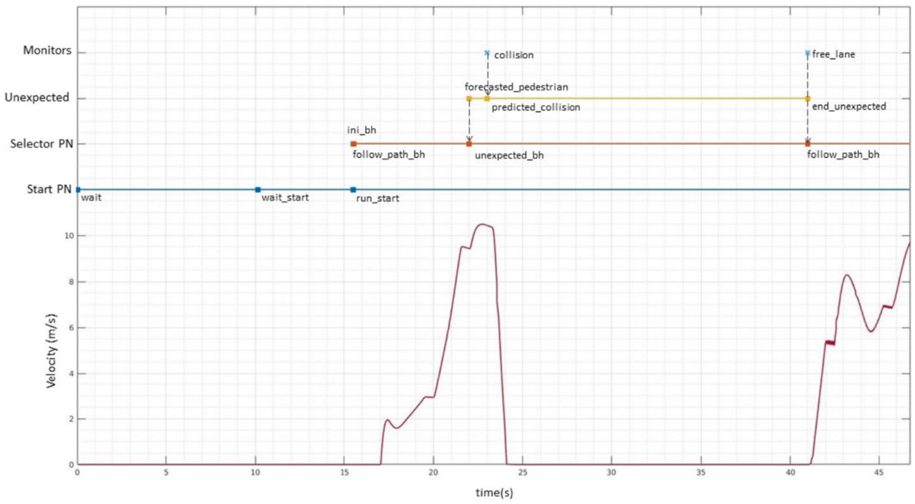


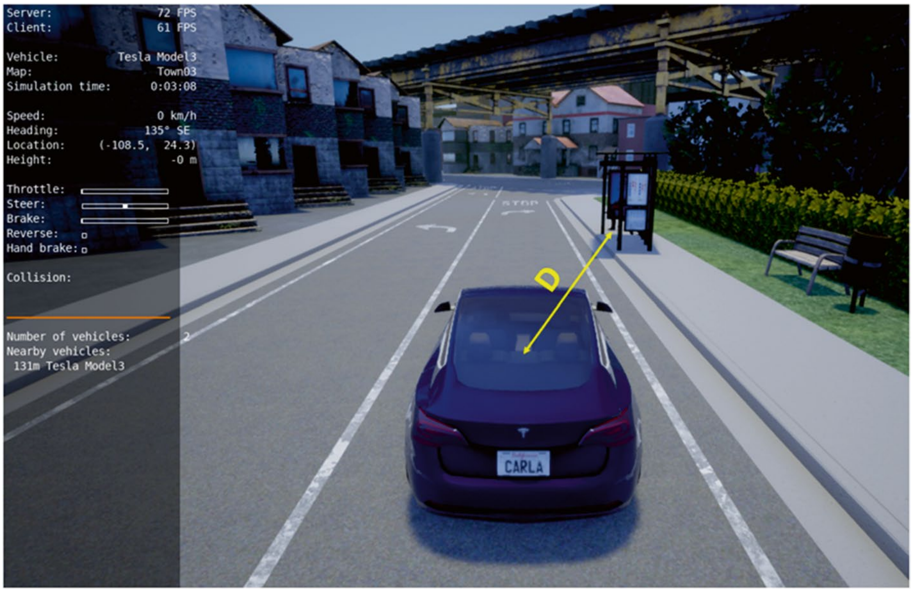
Fig. 11 Unexpected Pedestrian Temporal diagram. At the top, the events produced by the monitors and map manager modules. In the middle, the selector, and start (background) PNs. At the bottom, the velocity of the car throughout the route

Table 2 Reactive control analysis on CARLA simulator. NV: Nominal Velocity of the ego-vehicle (km/h). D: Minimum distance between the adversary and the ego-vehicle to start its trajectory. Blue and Gray cells indicate that a collision has not occurred and there has been, respectively

$NV(km/h)\backslash D(m)$	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

that no high-level behaviour is launched because this situation is not included as a specific use case (with its corresponding HBIPN), but it is always running in the background similar to the ACC behaviour. Figure 11 represents a temporal diagram with the sequence of events and the actual velocity of the car throughout the scenario in a similar way that the shown in Fig. 9. The ego-vehicle starts far away from the adversary and starts its journey. At second 22 a pedestrian that is in the sidewalk is detected, so s/he is tracked and forecasted in the short-term. After that, our prediction module intersects the ego-vehicle forecasted trajectory and the pedestrian forecasted trajectory. If the Intersection over Union (IoU) is greater than a threshold (in this case, 0.01), a *predicted collision* flag is activated and the low-level (reactive) control, which is always running in the background, performs an emergency break until the car is stopped in front of the obstacle. Navigation is resumed once the obstacle leaves the driving lane. This temporal graph shows a good performance 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 a combination of different "jumping distances" D in meters and different nominal speed of the ego


```
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. 12 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

vehicle *NV* in km/h. Blue cells indicate no collision has taken place. The *D* parameter represents the initial Euclidean distance between the adversary pedestrian and the ego-vehicle Bird's Eye View (BEV) centroid, as illustrated in Fig. 12. This distance corresponds with the initial condition of the pedestrian, placed at the sidewalk, 1.5 m away from the road. In a similar way to the pedestrian crossing scenario, we always monitorize an additional safety area (the sidewalk) to detect the presence of VRUs and forecast them to ensure a safety navigation. In this case, since the bus marquee is totally occluding the VRU, s/he is tracked and forecasted once jumps on the lane, since we still do not present a system to forecast the trajectories of dynamic obstacles in occluded obstacles [67, 50], which is a quite interesting state-of-the-art topic in the field of AD. 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.

7 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, responsible 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 GNSS, camera (including semantic segmentation) and LiDAR. The validation has consisted on the study of some traffic scenarios inspired on the National Highway Safety Traffic Administration (NHTSA) typology, and in particular in the CARLA Autonomous Driving Challenge (CADC), such as Stop, Pedestrian Crossing, Adaptive Cruise Control and Unexpected Pedestrian. In particular, the work was extended from our previous publication with several interesting temporal graphs to analyze in a holistic way the sequence of events and its impact on the physical behavior of the vehicle. 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, we will study a comparison among different autonomous driving architectures using our ongoing Autonomous Driving Benchmark Development Kit which will help us to compare in terms of score and quantitative results different navigation pipelines, such as Baidu Apollo or Autoware [53]. Moreover, sensor errors (such as localization and perception raw data) and associated uncertainty will be thoroughly studied in future works as an ablation study to observe the influence of these modifications in the final validation of the architecture. Regarding these pseudo-sensors, we plan to use traditional semantic segmentation algorithms, or even panoptic segmentation networks, instead of our traditional Precision-Tracking approach, since using panoptic directly provides an identifier for each object in every class, which would facilitate the process of data association and creation of trackers. In that sense, a Deep Learning based 3D Multi-Object Tracking and Motion Prediction algorithm, an enhanced version of our reactive control, an state-of-the-art Human-Machine Interface (HMI) and new behaviours in the behavioural decision-making layer will be implemented as well as the validation of the architecture in more challenging situations to improve the reliability, effectiveness and robustness of our system as a preliminary stage before implementing it in our real autonomous electric car.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. 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. Althoff M, Urban S, Koschi M (2018) Automatic conversion of road networks from opendrive to lanelets. In: 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI). IEEE, pp 157–162
2. Bast H, Delling D, Goldberg A, Müller-Hannemann M, Pajor T, Sanders P, Wagner D, Werneck RF (2016) Route planning in transportation networks. In: Algorithm engineering. Springer, pp 19–80
3. Beeson P, O'Quin J, Gillan B, Nimmagadda T, Ristroph M, Li D, Stone P (2008) Multiagent interactions in urban driving. Journal of Physical Agents 2(1):15–29
4. Beis JS, Lowe DG (1997) Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition. IEEE, pp 1000–1006
5. Bender P, Ziegler J, Stiller C (2014) Lanelets: Efficient map representation for autonomous driving. In: Intelligent Vehicles Symposium Proceedings, 2014 IEEE. IEEE, pp 420–425
6. Benekohal RF, Treiterer J (1988) Carsim: Car-following model for simulation of traffic in normal and stop-and-go conditions. Transp Res Rec 1194:99–111
7. Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J et al (2016) End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316
8. Brandes U (2001) A faster algorithm for betweenness centrality. J Math Sociol 25(2):163–177
9. Caesar H, Bankiti V, Lang AH, Vora S, Liong VE, Xu Q, Krishnan A, Pan Y, Baldan G, Beijbom O (2020) nuscenes: A multimodal dataset for autonomous driving. In : Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp 11621–11631
10. Chen C, Seff A, Kornhauser A, Xiao J (2015) Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision. pp 2722–2730
11. Davies A (2016) Google's self-driving car caused its first crash. Wired
12. Del Egido J, Gómez-Huélamo C, Bergasa LM, Barea R, López-Guillén E, Araluce J, Gutiérrez R, Antunes M (2020) 360 real-time 3d multi-object detection and tracking for autonomous vehicle navigation. In: Workshop of Physical Agents. Springer, pp 241–255
13. Dickmanns ED, Mysliwetz B, Christians T (1990) An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles. IEEE Trans Syst Man Cybern 20(6):1273–1284
14. Dosovitskiy A, Ros G, Codevilla F, López A, Koltun V (2017) Carla: An open urban driving simulator. arXiv preprint arXiv:1711.03938
15. Dupuis M, Strobl M, Grezlikowski H (2010) 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

16. Fernandez C, Izquierdo R, Llorca DF, Sotelo MA (2015) A comparative analysis of decision trees based classifiers for road detection in urban environments. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems. IEEE, pp 719–724
17. Fernández JL, Sanz R, Benayas J, Diéguez AR (2004) Improving collision avoidance for mobile robots in partially known environments: the beam curvature method. Robot Auton Syst 46(4):205–219
18. Fernández JL, Sanz R, Paz E, Alonso C (2008) Using hierarchical binary petri nets to build robust mobile robot applications: Robograph. In: 2008 IEEE International Conference on Robotics and Automation. IEEE, pp 1372–1377
19. Geiger A, Lenz P, Urtasun R (2012) Are we ready for autonomous driving? The Kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, pp 3354–3361
20. Gold C, Körber M, Lechner D, Bengler K (2016) Taking over control from highly automated vehicles in complex traffic situations: the role of traffic density. Hum Factors 58(4):642–652
21. Golinska P, Hajdul M (2012) European union policy for sustainable transport system: Challenges and limitations. In: Sustainable transport. Springer, pp 3–19
22. Gómez-Huelamo C, Bergasa LM, Barea R, López-Guillén E, Arango F, Sánchez P (2019) Simulating use cases for the UAH autonomous electric car. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, pp 2305–2311
23. Gómez-Huélamo C, Del Egido J, Bergasa LM, Barea R, López-Guillén E, Arango F, Araluce J, López J (2020) Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in CARLA simulator. In: Workshop of Physical Agents. Springer, pp 44–59
24. Gómez-Huélamo C, Del Egido J, Bergasa LM, Barea R, Ocana M, Arango F, Gutiérrez-Moreno R (2020) Real-time bird's eye view multi-object tracking system based on fast encoders for object detection. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, pp 1–6
25. Guo K, Yan Y, Shi J, Guo R, Liu Y (2017) An investigation into C-NCAP AEB system assessment protocol. In: SAE Technical Paper. SAE International. https://doi.org/10.4271/2017-01-2009
26. Haas J (2014) A history of the unity game engine. Diss, Worcester Polytechnic Institute
27. Haklay M, Weber P (2008) Openstreetmap: User-generated street maps. IEEE Pervasive Comput 7(4):12–18
28. Held D, Levinson J, Thrun S (2013) Precision tracking with sparse 3d and dense color 2d data. In: 2013 IEEE International Conference on Robotics and Automation. IEEE, pp 1138–1145
29. Ivanov A, Kristalniy S, Popov N (2021) Russian national non-commercial vehicle safety rating system runcap. In: IOP Conference Series: Materials Science and Engineering, vol. 1159. IOP Publishing, p 012088
30. Janai J, Guney F, Wulff J, Black MJ, Geiger A (2017) 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
31. Jullien JM, Martel C, Vignollet L, Wentland M (2009) Openscenario: a flexible integrated environment to develop educational activities based on pedagogical scenarios. In: 2009 Ninth IEEE International Conference on Advanced Learning Technologies. IEEE, pp 509–513
32. Kalman RE et al (1960) A new approach to linear filtering and prediction problems. J Basic Eng 82(1):35–45
33. Kaur P, Taghavi S, Tian Z, Shi W (2021) A survey on simulators for testing self-driving cars. arXiv preprint arXiv:2101.05337
34. Ko NY, Simmons RG (1998) 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. IEEE, pp 1615–1621
35. Koenig N, Howard A (2004) Design and use paradigms for gazebo, an open-source multi-robot simulator. In Intelligent Robots and Systems, 2004 (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, vol. 3. IEEE, pp 2149–2154
36. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Adv Neural Inf Proces Syst 25:1097–1105
37. Kurniawati H, Hsu D, Lee WS (2008) Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and systems , vol. 2008. Zurich, Switzerland
38. Lattarulo R, Pérez J, Dendaluce M (2017) A complete framework for developing and testing automated driving controllers. IFAC-PapersOnLine 50(1):258–263
39. Levinson J, Askeland J, Becker J, Dolson J, Held D, Kammel S, Kolter JZ, Langer D, Pink O, Pratt V et al (2011) Towards fully autonomous driving: Systems and algorithms. In: 2011 IEEE Intelligent Vehicles Symposium (IV). IEEE, pp 163–168

40. Liu R, Wang J, Zhang B (2020) High definition map for automated driving: Overview and analysis. J Navig 73(2):324–341
41. López J, Sánchez-Vilariño P, Sanz R, Paz E (2020) Implementing autonomous driving behaviors using a message driven petri net framework. Sensors 20(2):449
42. Matthaeia R, Reschkaa A, Riekena J, Dierkesa F, Ulbricha S, Winkleb T, Maurera M (2015) Autonomous driving: Technical, legal and social aspects
43. Merat N, Jamson AH, Lai FC, Daly M, Carsten OM (2014) Transition to manual: Driver behaviour when resuming control from a highly automated vehicle. Transportation Research Part F: Traffic Psychology and Behaviour 27:274–282
44. Merkel D (2014) Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal 2014(239):2
45. Michal DS, Etzkorn L (2011) A comparison of player/stage/gazebo and Microsoft robotics developer studio. In: Proceedings of the 49th Annual Southeast Regional Conference. ACM, pp 60–66
46. Montemerlo M, Roy N, Thrun S (2003) 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. IEEE, pp 2436–2441
47. Murciego E, Huélamo CG, Barea R, Bergasa LM, Romera E, Arango JF, Tradacete M, Sáez Á (2018) Topological road mapping for autonomous driving applications. In: Workshop of Physical Agents. Springer, pp 257–270
48. Najm WG, Smith JD, Yanagisawa M et al (2007) Pre-crash scenario typology for crash avoidance research. Tech. rep., United States. National Highway Traffic Safety Administration
49. Paden B, Čáp M, Yong SZ, Yershov D, Frazzoli E (2016) A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on Intelligent Vehicles 1(1):33–55
50. Park JS, Manocha, D (2020) HMPO: Human motion prediction in occluded environments for safe motion planning. arXiv preprint arXiv:2006.00424
51. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) Ros: An open-source robot operating system. In: ICRA workshop on open source software, vol. 3. Kobe, Japan, p 5
52. Rajamani R (2011) Vehicle dynamics and control. Springer Science & Business Media
53. Raju VM, Gupta V, Lomate S (2019) Performance of open autonomous vehicle platforms: Autoware and Apollo. In: 2019 IEEE 5th International Conference for Convergence in Technology (I2CT). IEEE, pp 1–5
54. Robotics C (2015) V-rep user manual. http://www.coppeliarobotics.com/helpFiles/. Ultimo acesso 13, 04
55. Rong G, Shin BH, Tabatabaee H, Lu Q, Lemke S, Možeiko M, Boise E, Uhm G, Gerow M, Mehta S et al (2020) LGSVL simulator: A high fidelity simulator for autonomous driving. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, pp 1–6
56. Sanders A (2016) An introduction to unreal engine 4. AK Peters/CRC Press
57. Schöner H (2017) The role of simulation in development and testing of autonomous vehicles. In: Driving Simulation Conference, Stuttgart
58. Shah S, Dey D, Lovett C, Kapoor A (2018) Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and service robotics. Springer, pp 621–635
59. Singh S (2015) Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Tech. rep.
60. Takács Á, Drexler DA, Galambos P, Rudas IJ, Haidegger T (2018) Assessment and standardization of autonomous vehicles. In: 2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES). pp 000185–000192
61. Taxonomy S (2016) Definitions for terms related to driving automation systems for on-road motor vehicles (j3016). Technical report, Society for Automotive Engineering, Tech. rep.
62. Tian Y, Pei K, Jana S, Ray B (2018) Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering. pp 303–314
63. Tideman M, Van Noort M (2013) A simulation tool suite for developing connected vehicle systems. In: 2013 IEEE Intelligent Vehicles Symposium (IV). IEEE, pp 713–718
64. Tradacete M, Sáez Á, Arango JF, Huélamo CG, Revenga P, Barea R, López-Guillén E, Bergasa LM (2018) Positioning system for an electric autonomous vehicle based on the fusion of multi-GNSS RTK and odometry by using an extented Kalman filter. In: Workshop of Physical Agents. Springer, pp 16–30

65. Urmson C, Anhalt J, Bagnell D, Baker C, Bittner R, Clark M, Dolan J, Duggins D, Galatali T, Geyer C et al (2008) Autonomous driving in urban environments: Boss and the urban challenge. J Field Rob 25(8):425–466
66. van Ratingen M, Williams A, Lie A, Seeck A, Castaing P, Kolke R, Adriaenssens G, Miller A (2016) The european new car assessment programme: A historical review. Chin J Traumatol 19(2):63–69. https://doi.org/10.1016/j.cjtee.2015.11.016. https://www.sciencedirect.com/science/article/pii/S1008127516000110
67. Wang L, Ye H, Wang Q, Gao Y, Xu C, Gao F (2020) Learning-based 3D occupancy prediction for autonomous navigation in occluded environments. arXiv preprint arXiv:2011.03981
68. Xu H, Gao Y, Yu F, Darrell T (2017) End-to-end learning of driving models from large-scale video datasets. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp 2174–2182
69. Yurtsever E, Lambert J, Carballo A, Takeda K (2020) A survey of autonomous driving: Common practices and emerging technologies. IEEE Access 8:58443–58469
70. Zhan W, Sun L, Wang D, Shi H, Clausse A, Naumann M, Kummerle J, Konigshof H, Stiller C, de La Fortelle A et al (2019) Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps. arXiv preprint arXiv:1910.03088

Authors and Affiliations

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]

Javier Del Egido
javier.egido@edu.uah.es

Luis M. Bergasa
luism.bergasa@uah.es

Rafael Barea
rafael.barea@uah.es

Elena López-Guillén
elena.lopezg@uah.es

Felipe Arango
juanfelipe.arango@edu.uah.es

Javier Araluce
javier.araluce@edu.uah.es

Joaquín López
joaquin@uvigo.es

1 Electronics Department, University of Alcalá (UAH), Madrid, Spain

2 Department of Systems Engineering and Automation, University of Vigo (UVIGO), Pontevedra, Spain