How to build and validate a safe and reliable Autonomous Driving stack? A ROS based software modular architecture baseline

Carlos Gómez-Huélamo[1], Alejandro Diaz-Diaz[1], Javier Araluce[1], Miguel E. Ortiz[1]
Rodrigo Gutiérrez[1], Felipe Arango[1], Ángel Llamazares and Luis M. Bergasa[1]

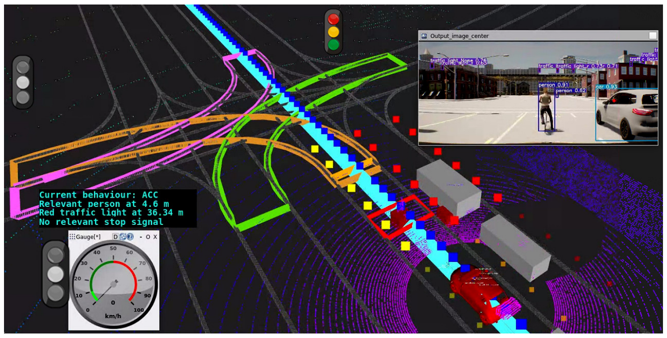
***Abstract*— The implementation of Autonomous Driving stacks (ADS) is one of the most challenging engineering tasks of our era. Autonomous Vehicles (AVs) are expected to be driven in highly dynamic environments with a reliability greater than human beings and full autonomy. Furthermore, one of the most important topics is the way to democratize and accelerate the development and research of holistic validation to ensure the robustness of the vehicle. In this paper we present a powerful ROS (Robot Operating System) based modular ADS that achieves state-of-the-art results in challenging scenarios based on the CARLA (Car Learning to Act) simulator, outperforming several strong baselines in a novel evaluation setting which involves non-trivial traffic scenarios and adverse environmental conditions (Qualitative results). Our proposal ranks in second position in the CARLA Autonomous Driving Leaderboard (Map Track) and gets the best score considering modular pipelines, as a preliminary stage before implementing it in our real-world autonomous electric car. To encourage the use research in holistic development and testing, our code is publicly available at https://github.com/RobeSafe-UAH/CARLA_Leaderboard.**

Keywords: Autonomous Driving, Modular, Simulation, CARLA, Holistic Validation, ROS

I. INTRODUCTION

In spite of all impressive efforts in the development of autonomous driving technology [1], fully-autonomous driving stacks (ADS) in arbitrarily complex scenarios are still years away. The reason for this is two-fold: Firstly, informed decision-making requires accurate perception. Most existing perception pipelines produce errors at a rate not acceptable for autonomous driving. Secondly, ADS that are operated in complex dynamic environments will require Artificial Intelligence based intelligent systems in order to generalize to unpredictable situations and reasons in a timely manner, even more considering the possible infractions committed by other vehicles on the road.

ADS combine a wide range of software components and sensors (both online and offline information) that once are processed are useful for the vehicle to make decisions and take actions [2]. In that sense, online information, also known as the traffic situation, is obtained by using the global perception system of the vehicle, which involves different on-board sensors (such as Light Detection and Ranging (LiDAR), Global Navigation Satellite System (GNSS), Camera, Inertial Measurement Unit (IMU) or actuators state) in order to create an accurate representation of the current situation to understand the surrounding environment and estimate the ego-vehicle location in real-time. On the other hand, offline information is identified as the prior knowledge of the ADS, such as the geographic and semantic information, as well as the topological relations of the environment based on high-definition maps [3], the vehicle dynamic and the traffic rules based on behavioural decision-making systems [4].



```
Current behaviour: ACC
Relevant person at 4.6 m
Red traffic light at 36.34 m
No relevant stop signal
```

Fig. 1: Simulation example of our Autonomous Driving stack using the RVIZ (ROS visualization) tool.

Furthermore, testing autonomous vehicles on public roads is an expensive and time-consuming endeavor. In order to validate an ADS the system must be tested in countless scenarios and environments, which would escalate the cost and development time exponentially with the physical approach. Considering this, the use of photo-realistic simulation and an appropriate design of the evaluation metrics and driving scenarios are the two current keys to build safe and reliable ADS. Simulation as a form of accelerating the path to market is not new to the AD industry. Tesla, Cruise, Waymo, Aurora, TuSimple and others have all touted the benefits of using simulations made from real-world data to test their AV systems, particularly against made-up scenarios that the systems have not encountered and cataloged in the real world yet.

The scope of this paper is to introduce the workflow by the RobeSafe research group (University of Alcalá, Spain) to build and validate an ADS, which achieves state-of-the-art results in challenging scenarios based on the CARLA simulator [5], outperforming several strong baselines in a new evaluation setting which involves non-trivial traffic scenarios and adverse environmental conditions , as a preliminary stage

*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), cofunded by EU Structural Funds and Scholarship for Introduction to Research activity by University of Alcalá

[1]Carlos Gómez-Huélamo, Alejandro Diaz-Diaz, Miguel E. Ortiz, Javier Araluce, Rodrigo Gutiérrez, Felipe Arango, Ángel Llamazares and Luis M. Bergasa are with the Electronics Department, University of Alcalá (UAH), Spain. carlos.gomezh, alejandro.diazd, eduardo.ortiz, rodrigo.gutierrezm, javier.araluce, angel.llamazares, luism.bergasa}@uah.es

before implementing it in our real-world autonomous electric car. 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.

II. RELATED WORKS

As mentioned in Section I, a fully ADS (L5 in the J3016 SA [6]) is still years away, not only for technical challenges, but also due to social and legal ones [7]. So far, no industry organization [4] has shown a ratified testing methodology for L4/L5 SAE 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 , there is a need to test AVs full of advanced sensors [8] using a holistic approach, where not only the individual components but also the general behaviour of the architecture is evaluated.

Regarding urban environment complexity, to validate an ADS the system must be tested in countless scenarios and environments, which would escalate the cost and development time exponentially with the physical approach, particularly against made-up scenarios that the system has not encountered and cataloged in the real world yet. Since the instrumentation of an experimental vehicle still has a high cost and conducting experiments on public road networks are highly regulated, the use of photo-realistic simulation environments and the design of challenging traffic scenarios are the two current keys to build safe and reliable ADS [8]. These simulators have evolved from merely simulating vehicle dynamics to also simulating more complex functionalities. In order to choose the right simulator [4], there is a set of criteria [9] that serve as a metric to identify which simulators are most suitable for our purposes, such as traffic infrastructure, sensors available, multi-view geometry, vehicle control, traffic scenario simulation, 2D/3D groundtruth, 3D virtual environment, open-source and scalability via a server multi-client architecture. Most private companies (Waabi, Tesla, Waymo, TuSimple, NVIDIA, etc.) use their private simulation, where free-to-use license is not available and derivative products are not allowed. For the open-sourced options [9], we can find Gazebo, CarSim, CARLA and LGSVL. At the moment of writing this paper, LGSVL and CARLA are the most suitable simulators for holistic simulation (either modular or end-to-end approaches). Most of their features are identical but LGSVL does not offer camera calibration to conduct multi-view geometry or Multi-Modal fusion for end-to-end training, which are some of our future works. Then, we decided to use the CARLA [5] simulator. Additionally, this last offers a leaderboard where teams can evaluate their approaches on secret test routes on the cloud.

Regarding AD archtectures, end-to-end and modular pipelines continue to be the dominant approaches to this day. End-to-end driving is defined as performing the entire driving task by a single neural network that takes in the raw sensor data and outputs the driving (steering, throttle and brake) commands. The main advantage of these approaches is that intermediate representations are jointly optimized for the driving task and supervision labels may be obtained cheaply. However, these methods are not very interpretable. Some end-to-end approaches are compared in Section IV with our modular architecture. On the other hand, in the module pipeline approach, the driving task is separated into individual modules that are programmed or trained individually. They are often classified as localization, mapping, planning, perception, decision-making (or executive) and perception layer respectively. One of the greatest advantages of using modular pipelines is that intermediate representations are specified by an expert and hence they are interpretable. Moreover, these different modules can be effectively developed in parallel by a large team, in such as way the modular pipeline approach has become the standard approach in industrial research. Nevertheless, these intermediate representations can led to suboptimal performance and may not be optimized for the final driving task.

Open-source modular frameworks are very useful for both researchers and the automation industry, and they can help to *democratize* AD development. Pylot [10] is an interesting modular approach, being the first modular architecture to be ranked in the CARLA Autonomous Driving Leaderboard (CADL). Autoware, Openpilot and Nvidia Drive-Works are amongst the most used software-stacks [1] capable of running an AD platform in real world. However, they do not follow a common standard and are incompatible with each other. Besides, they are not very flexible, difficult to adapt to some types of sensors/actuators and have restrictions to be transferred to real potentially marketable vehicles. Regarding this, we finally consider to develop our own modular ADS (see Section III) based on ROS [11], Docker [12] and CARLA.

III. AUTONOMOUS DRIVING STACK

To accomplish the AD development and validation, we present a workflow (Fig. 2) that uses the CARLA simulator to validate our AD approach. Traditionally, a layered (modular) architecture pattern [2] establishes a hiearchical and topological organization of the software components with a similar abstraction level. The communication among them can be classified as inter-layer and intra-layer, using the publish/subscribe paradigm to provide non-blocking communications. The former means the communication of software components (ROS nodes in this case) of different layers while the latter means the communication of nodes belonging to the same layer. ROS is featured by a well-defined communication protocol, that allows the replacement of specific components or even the entire layer, keeping the core system behaviour, being one of the main advantages of modular pipelines. Using this paradigm, in addition to the encapsulation of the ADS using Docker reinforces the scalability, reusability, maintanability and modularity of the ADS style among the different members of our team. In particular, our ADS is hierarchically broken down in the following layers: Localization, Mapping, Planning, Perception,

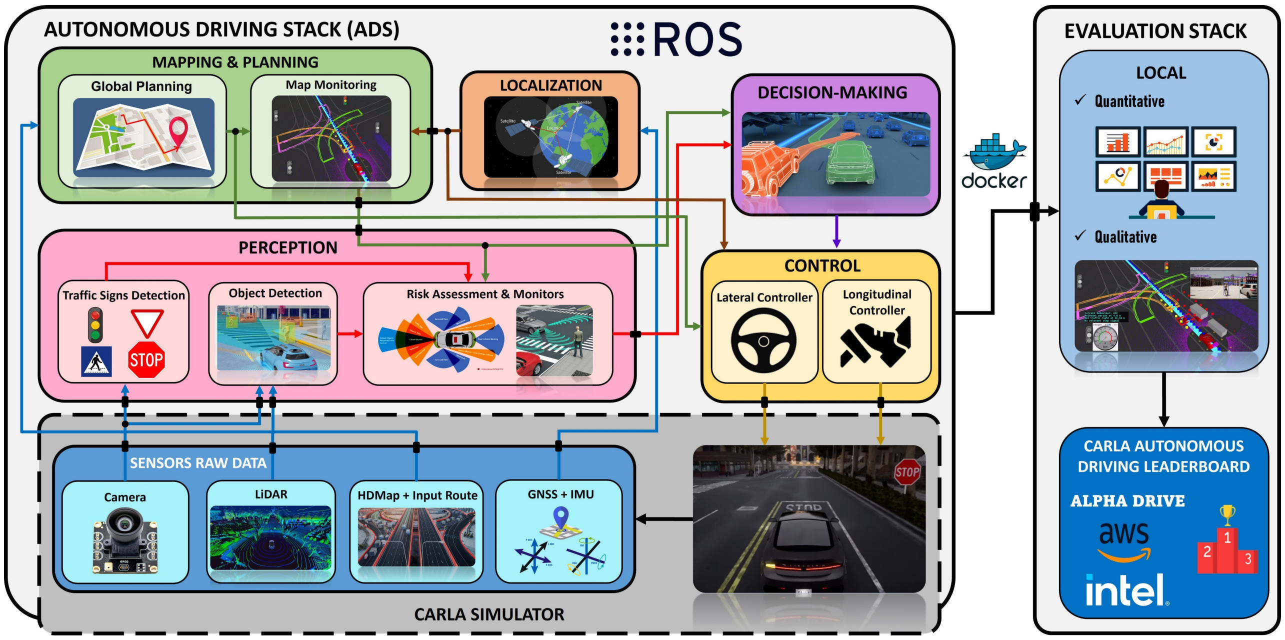


Fig. 2: Our **Autonomous Driving Stack Workflow**. Three main blocks are identified: the **CARLA simulator block**, which provides the raw data and receives the driving commands, the **ADS block**, which receives the raw data information and processes the traffic situation to commit the least number of traffic infractions , and the **Evaluation Stack block** that receives the architecture encapsulated as a Docker, and validates the ADS first in local inference as a preliminary stage before submitting it to the CARLA Autonomous Driving Leaderboard (CADL)

Decision-Making and Control.

A. Localization layer

Accurate and robust localization is a primary task for our autonomous navigation architecture. This layer is responsible for two core tasks in the architecture. Firstly, publishing the transform tree that connects all frames, and secondly positioning the ego-vehicle in the map. When locating the vehicle, it is mandatory to use the correct reference system, taking into account the ego-vehicle reference and how the different sensors or elements of interest are distributed around it. Each of these points around the vehicle has its particular frame, and it is required that all frames are connected using a transform tree, which allows ROS to compute the transforms among frames in a very efficiency way, being this one of the most powerful features of ROS.

As observed in Fig. 3, we estimate the ego-vehicle pose fusing the geographic position provided by a GNSS (Global Navigation Satellite System) and the orientation provided by an IMU. The GNSS sensor provides the measurements in World Geodetic System (WGS84) coordinates. Nevertheless, to simplify the localization problem, we use UTM (Universal Transverse Mercator) coordinates, since working with distances is easier that using a Cartesian system. In this case, we use the same conversion than proposed by the CARLA simulator.

To obtain the ego-vehicle orientation, we use the compass provided by the IMU. Nevertheless, a transformation is applied to estimate the heading orientation (θ_z) based on the Northing (N) value.

Furthermore, in order to reduce the noise associated to the GNSS and IMU measurements (which are set specifically in CARLA), we use an Extended Kalman Filter (EKF) [13], as illustrated in Fig. 3.

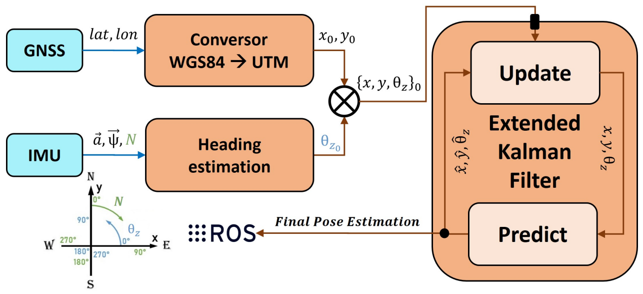


Fig. 3: **Localization pipeline** based on an Extended Kalman Filter (EKF) fed by the heading angle and UTM coordinates

B. Planning layer

In order to start the navigation, our ADS first calculates the optimal global route given the current position of the ego-vehicle and a goal. This task is done by the path planner, which calculates the global route as a list of waypoints centered in the lanes using the Dijkstra algorithm [14]. A waypoint represents a 3D oriented point with location (*x,y,z*), rotation (*roll,pitch,yaw*) and the corresponding topological information, such as the road speed or if that waypoint is inside an intersection. Considering this, the Planning layer is divided into two main modules: Lane Graph Planner (LGP) and Lane Waypoint Planner (LWP).

1) Lane Graph Planner: The LGP generates a directed graph (DiGraph) of roads and lanes. The LGP builds the graph using edges, that represent the connection between two nodes. In this case a node is a tuple of 2 parameters: *road_id* and *lane_id*. Each edge is defined as a Python set containing the input node, output node and weight. The weight represents the cost that will be used by the graph planner for going from the input node to the output node. In this case the weights used are road lengths in meters and time to travel the road in seconds calculated using the maximum

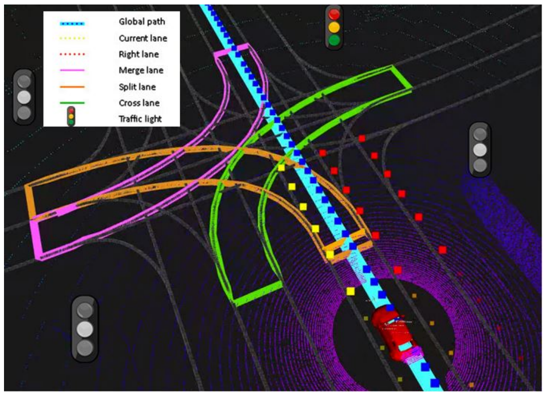


Fig. 4: Monitored lanes, global route and regulatory elements are represented in RVIZ. Non-relevant traffic lights are painted in grayscale

speed allowed for each segment. To generate the edges and build the graph, the LGP evaluates connections for every road/lane of the map object parsed from the HD map. Lane change is also considered adding a cost value for lane change when it is allowed. Once we have the road graph, the route is calculated using the Dijkstra algorithm. The route returned by the LGP is a topological route as a list of tuples: (road, lane, action).

2) Lane Waypoint Planner: Given the topological global route calculated by the LGP, the LWP module is in charge of calculating a list of waypoints separated by a given distance specified by the user for every road-lane calculated by the LGP. For doing that, it uses the road-lane geometries derived from the HD map and applies mathematical discretization in such a way the output of this module is set of waypoints separated by a distance *d*.

For a deeper explanation we remit the reader to [15].

C. Mapping layer

This layer analyzes the prior knowledge of the vehicle. It is divided in two main modules: map parser and map monitor. The map parser is in charge of parsing and processing the HD map input and providing useful information (such as the monitored lanes or the current traffic light) to the other layers of the vehicle, like the Planning or Perception. On the other hand, the map monitor module is the module responsible for monitoring the surrounding area of the vehicle and visualizing in RVIZ (ROS visualization tool) both the lanes describing the roads of the map and the monitored area The map monitor inputs are the map parser output, the global route calculated by the path planner and the ego-vehicle location in such a way it only monitors the road infrastructure around the ego-vehicle. As mentioned above, the monitored area (Fig. 4) is divided into three main groups:

- *Standard Lanes*: Lanes available around the ego-vehicle in all traffic situations. Current and back lane are monitored from the current position, frontwards and backwards respectively, to a dynamic distance proportional to the velocity of the ego-vehicle.
- *Intersection Lanes*: Lanes that intersect with the current lane of the vehicle. They may have three different roles: split (lane derived from the current lane), merge (lane that merges with the current lane) or cross (lane that crosses the current lane but neither its start not its end are connected with the current lane).
- *Regulatory Elements*: Topological information from the HD map, represented by the traffic signals and road information, such as the traffic lights, stops, crosswalks or speed signals. Regulatory elements are only monitored for the next intersection affecting the route.

More information can be found in [15].

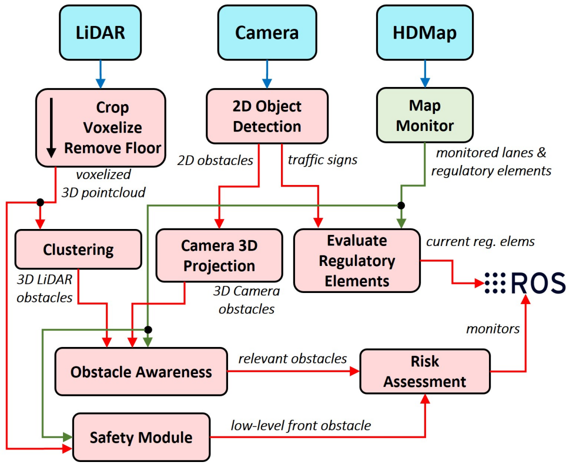


Fig. 5: **Perception pipeline** based on camera 2D object detection, LIDAR 3D clustering and obstacle awareness and risk assessment evaluating 3D obstacles in the HD map

D. Perception layer

In order to understand the environment around the ego-vehicle, our ADS employs two perception systems to capture the traffic situation, camera and LiDAR, as well as a map monitor module, which represents prior knowledge and is made up by the monitored lanes (both standard and intersection) and regulatory elements, as shown in Fig. 5. In the present work we focus on the fusion of 3D object detection and prior knowledge in order to study all the benefits provided by the HD map information, giving rise to a fast yet powerful perception node that processes the raw data to perform obstacle awareness and risk assessment, as well as a LiDAR based safety module, to dump this information into the ROS communications, mainly used by the decision-making layer to conduct the corresponding traffic behaviour.

Despite the fact there are quite interesting studies that focus on enhancing the quality of simulated raw data, the quality of the LiDAR pointcloud of the current CARLA version of the CADL (0.9.10.1) is far away from the quality offered by real-world datasets, like KITTI [16] or NuScenes [17]. Considering our modular architecture, a Deep Learning based 3D object detector is not suitable for our purposes. Then, we use a simple Machine Learning based algorithm to get the 3D clusters from the voxelized 3D pointcloud. First, the original LiDAR pointcloud is cropped (front, back and both sides) a distance proportional to the ego-vehicle velocity, and voxelized with a resolution of *0.3 m*, giving rise to a tradeoff between information lose and redundancy, enough for this purpose. Then, we remove the floor using the well-established RANSAC-3D algorithm [18] for efficient plane extraction using a tolerance of *0.2 m* to include inliers. After that, we conduct fast nearest neighbour searches by using the KDTree (k-dimensional tree) [19] clustering algorithm for 3D data and FLANN (Fast Library for Approximate Nearest Neighbors), allowing for fast nearest neighbour searches to find correspondences between groups of points.

In terms of camera information, we employ a Convolutional Neural Network (CNN) known as YOLOv5 [20] (L architecture) that computes the most relevant 2D objects for our purposes. Traffic signs detections are used, in addition to the output of the map monitor module, to compute the semantic information of the current regulatory elements . Moreover, to project the 2D obstacles into the 3D space, as a prelimininary stage before implementing a DL based 3D detector from monocular information, we simplify the projection stage assuming flat floor. Then, by using the camera height with respect to the floor and the 2D detection coordinates, the corresponding 3D projection is obtained, which must be transformed into a common reference frame, in this case the *map* frame. Once the 3D clusters are computed, we determine whether they are relevant or not by studying their presence in the monitored lanes (standard and intersection lanes) to conduct the corresponding executive behaviour, such as Adaptive Cruise Control (ACC), Stop or Crosswalk. Studying only the most relevant obstacles helps us to increase the reliability and robustness of our ADS since evaluating all the objects would escalate the computational cost in an arbitrarily complex urban scenario. Nevertheless, the ADS may face sometimes unexpected VRUs (Vulnerable Road Users) that may not be in a particular monitored lane and suddenly jump into the road in non-signalized venues. In that sense, if the obstacle is identified as VRU (cyclist or pedestrian), we enlarge the lane a certain threshold t to the sidewalk in order to be prepared before a possible emergency braking, running in the decision-making layer background.

With this strategy and after conducting many experiments, we realized that some special vehicles like the CARLA Cola or the CyberTruck sometimes are not detected either by the KDTree clustering or by YOLOv5-L due to its particular shape and appearance, which represents an interesting challenge for our ADS in order to avoid the corresponding collisions. Then, we develop a safety module, always running in the background of the perception node, which is fed by the voxelized 3D pointcloud and the monitored lanes in order to study the presence of a relevant group of points which has not been previously computed.

E. Decision-Making layer

The decision-making layer must provide tools to model the sequence of actions and events, based on some predefined traffic rules, that can take place in the different traffic scenarios. Different approaches have been proposed in the literature: Partially Observable Markov Decision Processes (POMDPs) [21], Decision Trees (DT) [22] and Hierarchical Finite-State Machines (HFSM) [23].

Petri Nets (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 that sense, we make use of Hierarchical Interpreted Binary Petri Nets (HIBPNs) [4] to model the behaviours for each traffic scenario, where a PN can start/stop another PN depending on its hierarchy. We run a main PN that defines the conditions to switch between the different behaviours (ACC, Stop, Traffic Light, Crosswalk, etc.), modeled as a sequence of discrete events, as well as an emergency break behaviour, which is always running in the background. The resulting pipeline is noticeable flexible since the user can easily modify the traffic rules with minor changes, such as the minimum distance to be stopped in front of an obstacle, in the corresponding PNs. In our case, as stated in Fig. 2, the decision-making layer is fed the risk assessment and monitors information from the perception layer and the monitored area from the map monitor module, which are jointly considered to conduct the top priority behavior. Finally, the decision-making layer feeds the control layer with the corresponding action to be carried out.

For more information about this module we remit the readers to [4].

F. Control layer

The Control layer usually represents the final part of every autonomous driving software stack, either end-to-end or modular approaches. It generates the steering, throttle and brake commands in order to keep the agent in the planned trajectory, which are directly sent to the vehicle, either the low-level hardware in real-world operation, or the simulator in this case. As observed in Fig. 2, this planned is calculated using the global planner, though it may be recalculated if the decision-making layers requires a local-planning action, as occur in the obstacle avoidance or overtake situations. This layer is divided into three modules: Waypoint interpolator, which calculate intermediate points trough two-dimensional cubic splines to ensure continuity. Longitudinal control, responsible for calculating the velocity profile throughout the route as a function of the current curvature of the road. Lateral control, module that generates the steering signal using a Linear-Quadratic Regulator (LQR) Optimal Controller. Deeper information can be found in [24].

IV. 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. In the present case, the scenarios both in local and CADL experiments are selected from the NHTSA (National Highway Transportation Safety Administration) pre-crash typology [25], one of the most famous NCAPs (New Car Assessment Programs). In that sense, the CADL proposes a benchmark for the evaluation of ADS that relies on different sets of software architectures approaches and set of sensors, being limited by the CARLA team . In this leaderboard, a particular ADS is run in an arbitrarily complex environment (urban, highway, residential district, unsigned intersections and so forth and

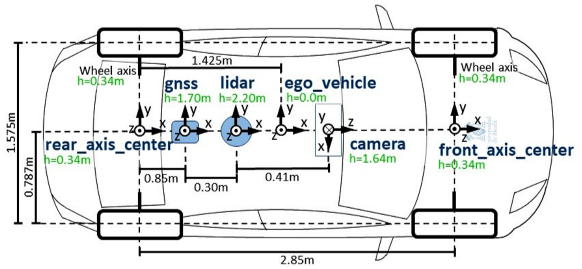


Fig. 6: Vehicle layout illustrating vehicle dimensions and sensor suite

so on), where the number of traffic participants, weather conditions and route lengths are specified beforehand. In order to measure the driving proficiency of an agent, CARLA uses the Driving Score (*DS*) metric. This metric can be broken down into two components, the Route Completion (*RC*) and the Infraction Penalty (*IP*):

- **Route Completion (RC)**: Percentage of route completed by the agent in a every route i, averaged across N routes.
$$RC = \frac{1}{N}\sum_{i}^{N} R_i \tag{1}$$
- **Infraction Penalty (IP)**: Geometric series of infraction penalty coefficients, p_i^j for every instance i of infraction j incurred by the agent during the route. Each agent starts with an ideal 1.0 base score for each route, which is reduced by a penalty coefficient for every infraction.
$$P_i = \prod_{j}^{ped,\ldots,stop} (p_i^j)^{infractions(j)} \tag{2}$$
- **Driving Score (DS)**: Weighted average of the infraction penalty with route completion for each route. This is the main metric used to evaluate the ADS both in local experiments and for ranking models on the CADL.
$$DS = \frac{1}{N}\sum_{i}^{N} R_i P_i \tag{3}$$

For these previous metrics, higher value is better.

All local test were carried out in a PC desktop (AMD Ryzen 9 5900X, 32GB RAM with CUDA-based NVIDIA GeForce RTX 3090 24GB VRAM), using the ROS Noetic version (Ubuntu 20.04) as the communication middleware. To conduct our experiments, we make use of the following sensor suite: Monocular RGB camera (1920x1080, 80 FoV), a 64-channels LiDAR, GNSS returning the geo location data, a 6-axis IMU and a speedometer that provides an approximation of the ego-vehicle linear velocity. Fig. 6 shows the vehicle dimensions and frames positions used in our experiments (both local and in the leaderboard).

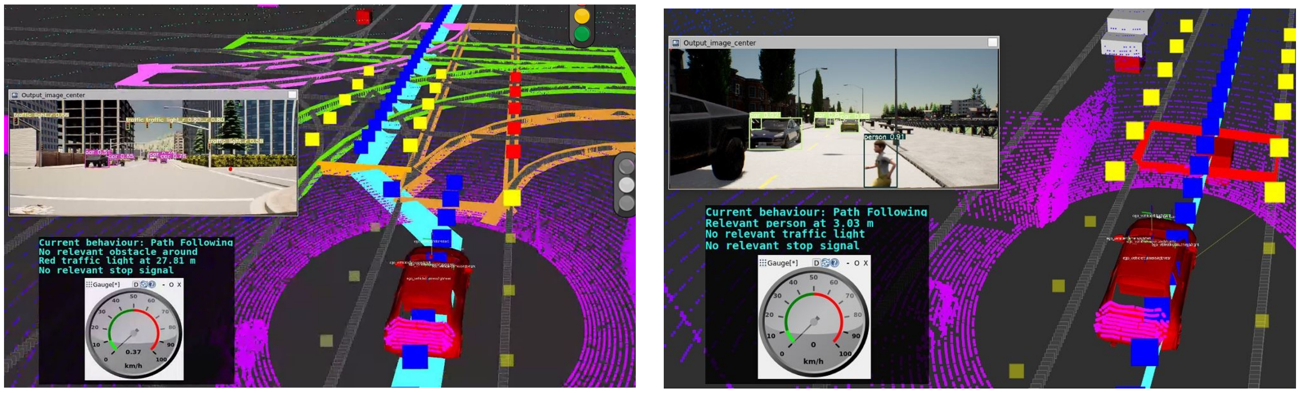
Note that the Docker image submitted to the AlphaDrive cloud platform, which hosts the CADL, presents exactly the same sensor suite and software architecture than in our local experiments. As observed in the general workflow (Fig. 2), we first develop our architecture using training routes. Then, validation and testing routes with non-observed scenarios are run. In terms of local testing, we conduct two local experiments (A and B respectively). Firstly, we observe the evolution our ADS and how the different improvements help to commit the fewer number of traffic infractions each time. Secondly, a local test proposed by [26] is run in order to compare our architecture with other state-of-the-art ADS as a preliminary local validation before submitting the Docker image to the cloud.

Regarding the first experiment, we select 5 different routes in 6 different CARLA towns (01 to 06). Along these routes we may find traffic situations which are based on the above mentioned NTHSA pre-crash typology, in addition to the own challenging features of each map (roundabouts, unevenness, tunnels, 5-lane junction, T-junctions, bridge, etc.). Table I and II summarize the local experiment A. In Table I we may observe the results of running a first version of our ADS with a basic version of the map monitor, without stop signals in the regulatory elements and the decision-making layer only computing the Adaptive Cruise Control (ACC) behaviour. On the other hand, Table II illustrates the results with our current configuration, in which we refactorize the ego-vehicle ROS node, which is the parent node in our ADS, to ensure sensor synchronization avoiding and prevent data loss. A full version of the map monitor is used in this case, computing both the standard and intersection lanes around the ego-vehicle as well as the most relevant regulatory elements, in addition to other improvements of the perception and decision-making layers according to the monitored elements provided, such as the LiDAR based safety module for low-level detection or the implementation of different behavioural use cases using HBIPNs. We may observe in both tables that DS, RC and IP are the average values for each of the towns. As expected, infraction penalties have been reduced incrementing the complexity of our ADS. By using an enhanced prior knowledge information, which derives in more robust reliable perception and decision-making monitors, we are able to minimize the agent blockouts, helping to improve the RC metric. Regarding collisions infractions, pedestrian collision is reduced to a half of the original value and collisions with vehicles are reduced to a third of the original value, contributing to improve IP parameter. In conclusion, the improvements of the local experiment A.2 with respect the local experiment A.1 are as following: +137.67% DS, +23.72% RC and +63.88% IP. Fig. 7 illustrates some interesting traffic scenarios of this first experiment.

TABLE I: Local experiment A.1.: Basic version of map monitor, DM layer only conducts Adaptive Cruise Control (ACC) behaviour.

Town	**RL [m]**	**DS ↑ [%]**	**RC ↑ [%]**	**IP ↑ [0,1]**
Town01	680.04	55.12	75.12	0.80
Town02	899.57	53.12	78.62	0.65
Town03	1520.76	2.36	77.02	0.04
Town04	2066.07	4.18	92.25	0.05
Town05	1043.06	9.81	38.60	0.48
Town06	1655.34	11.25	90.7	0.13
Global Metrics	1310.81	**22.64**	**75.39**	**0.36**

Once we have observed increasing complexity of our ADS is directly related to improvement of the driving proficiency, represented by the DS metric, we conduct a second internal evaluation proposed by [26], comparing the driving proficiency of our architecture against some state-of-the-art end-to-end approaches. The evaluation consists of 42 routes


Current behaviour: Path Following
No relevant obstacle around
Red traffic light at 27.81 m
No relevant stop signal

Current behaviour: Path Following
Relevant person at 3.03 m
No relevant traffic light
No relevant stop signal

(a) Red Traffic Light and lane change use case

(b) Unexpected pedestrian use case

Fig. 7: Qualitative results of our local experiment A. (a) The ego-vehicle is waiting in front of a red traffic light, monitoring the intersection lanes, before carrying out a lane change. (b) An Unexpected VRU jumps into the road giving rise to an emergency break. As commented in Sec. III-D, the CyberTruck (on the left lane) is not detected either by Yolov5-L or KD-Tree, representing an interesting challenge during navigation

from 6 different CARLA towns (01 to 06) where each route presents a unique environment combining one of 7 weather conditions (Cloudy, Wet, MidRain, WetCloudy, HardRain, SoftRain, Clear) with one of 6 daylight conditions (Morning, Noon, Sunset, Dawn, Twilight, Night).

TABLE II: Local experiment A.2. : Full version of map monitor, Safety Module is implemented, Full version of DM layer, Blockouts are minimized

Town	**RL [m]**	**DS ↑ [%]**	**RC ↑ [%]**	**IP ↑ [0,1]**
Town01	680.04	86.0	100.00	0.86
Town02	899.57	45.17	88.09	0.54
Town03	1520.76	56.44	86.56	0.7
Town04	2066.07	52.60	98.27	0.53
Town05	1043.06	38.42	86.76	0.46
Town06	1655.34	44.24	100.00	0.44
Global Metrics	1310.81	**53.81**	**93.28**	**0.59**

Regarding the different approaches, CIRLS [27] learns to directly predict the vehicle control from visual features while being conditioned on a discrete navigational command. Learning By Cheating (LBC) [28] models the navigation task as a teacher model, with access to the Bird's Eye View (BEV) semantic segmentation groundtruth maps, and finally an image-based student model trained using supervision from the teacher. AIM [29] represents a Multi-Modal fusion transformer as an improved version of CIRLS, where a GRU (Gated Recurrent Unit) decoder regresses control waypoints. NEAT [26] presents a continous function which maps locations in BEV scene coordinates to waypoints and semantics, using intermediate attention maps to iteratively compress highdimensional 2D image features into a compact representation. Table III illustrates our comparison. The evaluation (42 routes) is performed three times for each model in order to report the mean and standard deviation for the main metrics. Our architecture obtains the best RC and the third best DS of the different configurations, with an Infraction Penalty on par with these solid end-to-end baselines. Qualitative results of this internal evaluation can be found **here**.

After validating the architecture with local experiments, we upload our Docker image to the CADL where teams can evaluate their approaches on secret test routes on provided third party servers. In particular, this secret test set consist of 10 routes evaluated on 2 unknown weather conditions with 5 repetitions (100 routes) in two secret CARLA towns, leading to a total of 173 km of driving experiences. In our case, we participate in the MAP modality, where the input high-level route is a list of tuples with the first element of the tuple expressing the waypoints in world coordinates, and the second element of the tuple contains a high-level command (lane follow, change lane left, etc.).

TABLE III: Local experiment B: We compare our ADS against different End-to-End architectures, showing the μ and σ over 3 evaluations for each model. We bold the best results in **black** and the second best in **blue** for each metric

Method	**Aux. Sup.**	**DS ↑ [%]**	**RC ↑ [%]**	**IP ↑ [0,1]**
CIRLS [27]	Velocity	22.97±0.90	35.46±0.41	0.66±0.02
LBC [28]	BEV Sem	29.07±0.67	61.35±2.26	0.57±0.02
AIM [29]	None	51.25±0.17	70.04±2.31	0.73±0.03
	2D Sem	57.95±2.76	80.21±3.55	0.74±0.02
AIM-MT	BEV Sem	60.62±2.33	77.93±3.06	0.78±0.01
	Dth+2D Sem	**64.86±2.52**	**80.81±2.47**	**0.80±0.01**
AIM-VA	2D Sem	60.94±0.79	75.40±1.53	0.79±0.02
NEAT [26]	BEV Sem	**65.10±1.75**	79.17±3.25	**0.82±0.01**
Ours	Modular	62.91±1.96	**92.11±1.84**	0.69±0.01

Table IV summarizes the CADL results in the Map track. As observed, we get both the best RC with our current architecture and second best DS, being the best modular pipeline in the CADL. GRI-based DRL [30] combines benefits from exploration and expert data and is straightforward to implement over any off-policy RL algorithm. Pylot [10], CaRINA [2] and SmartElderlyCar [4] are modular platforms with several state-of-the-art reference implementations for the various components of the ADS, similar to our architecture. Nevertheless, despite the fact we beat our previous submission considering the DS (12.63 → 18.75%) and RC (61.59 → 75.11%) metrics and our ADS ranks in second position, the IP metric (0.28) is not on a par with the results obtained in the local experiments (0.59 from Table II and 62.91±1.96 from Table III), which indicates that at the moment of writing this work, our method is not able to safely drive on the novel conditions of the leaderboard.

TABLE IV: CARLA Autonomous Driving Leaderboard results (MAP track). We bold the best results in **black** and the second best in **blue** for each metric

Team	Submission	**DS** ↑ [%]	**RC** ↑ [%]	**IP** ↑ [0,1]
Anonymous	GRI-based DRL [30]	**33.78**	57.44	**0.57**
RobeSafe	Techs4AgeCar (Ours)	**18.75**	**75.11**	0.28
ERDOS	Pylot [10]	16.70	48.63	**0.50**
LRM-B	CaRINA [2]	15.55	40.63	0.47
RobeSafe	SmartElderlyCar [4]	12.63	**61.59**	0.33

V. CONCLUSIONS AND FUTURE WORKS

This work presents the implementation and validation of a powerful ROS-based modular software ADS using CARLA and Docker as a baseline to democratize and accelerate the development and research of holistic validation of AVs. The modular architecture is made up by several state-of-the-art reference implementations for the different layers of the vehicle. The validation has consisted on two local experiments, where we have compared our ADS both against a previous version of the ADS as well as against solid end-to-end baselines in a novel evaluation setting which involves non-trivial traffic scenarios (based on the NHTSA pre-crash typology) and adverse environmental conditions. The final objective has been the validation of our ADS in the secret test routes of the CADL, where even beating other solid modular baselines in the MAP track in terms of driving proficiency, the results in the cloud are not on pair with the results obtained throughout internal evaluation. We plan to reduce this gap, as a preliminary stage before implementing the ADS in our real-world electric prototype.

REFERENCES

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58443–58469, 2020.

[2] L. A. Rosero, I. P. Gomes, J. A. R. da Silva, T. C. d. Santos, A. T. M. Nakamura, J. Amaro, D. F. Wolf, and F. S. Osório, "A software architecture for autonomous vehicles: Team lrm-b entry in the first carla autonomous driving challenge," *arXiv preprint arXiv:2010.12598*, 2020.

[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] C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, J. Araluce, and J. López, "Train here, drive there: Ros based end-to-end autonomous-driving pipeline validation in carla simulator using the nhtsa typology," *Multimedia Tools and Applications*, pp. 1–28, 2021.

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[6] 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.

[7] R. Matthaeia, A. Reschkaa, J. Riekena, F. Dierkesa, S. Ulbricha, T. Winkleb, and M. Maurera, "Autonomous driving: Technical, legal and social aspects," 2015.

[8] H. Schöner, "The role of simulation in development and testing of autonomous vehicles," in *Driving Simulation Conference, Stuttgart*, 2017.

[9] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, "A survey on simulators for testing self-driving cars," *arXiv preprint arXiv:2101.05337*, 2021.

[10] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8806–8813, IEEE, 2021.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[12] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[13] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, p. 46, 2004.

[14] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[15] D.-D. Alejandro, M. Ocaña, Llamazares, C. Gómez-Huélamo, L. M. Bergasa, and P. Revenga, "Hd maps: Exploiting opendrive potential for path planning and map monitoring," in *2022 IEEE Intelligent Vehicles Symposium (IV): In submission*, IEEE, 2022.

[16] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.

[17] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631, 2020.

[18] H. Cantzler, "Random sample consensus (ransac)," *Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh*, 1981.

[19] Z. Lari, A. Habib, and E. Kwak, "An adaptive approach for segmentation of 3d laser point cloud," in *ISPRS Workshop laser scanning*, vol. 38, pp. 29–31, 2011.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[21] 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.

[22] 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.

[23] P. Beeson, J. O'Quin, B. Gillan, T. Nimmagadda, M. Ristroph, D. Li, and P. Stone, "Multiagent interactions in urban driving," 2008.

[24] R. Gutiérrez, E. López-Guillén, L. M. Bergasa, R. Barea, Ó. Pérez, C. Gómez-Huélamo, F. Arango, J. Del Egido, and J. López-Fernández, "A waypoint tracking controller for autonomous road vehicles using ros framework," *Sensors*, vol. 20, no. 14, p. 4062, 2020.

[25] 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.

[26] K. Chitta, A. Prakash, and A. Geiger, "Neat: Neural attention fields for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15793–15803, 2021.

[27] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, 2019.

[28] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*, pp. 66–75, PMLR, 2020.

[29] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7077–7087, 2021.

[30] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde, "Gri: General reinforced imitation and its application to vision-based autonomous driving," *arXiv preprint arXiv:2111.08575*, 2021.