HD maps: Exploiting OpenDRIVE potential for Path Planning and Map Monitoring

Alejandro Diaz-Diaz[1], Manuel Ocaña[1], Ángel Llamazares[1], Carlos Gómez-Huélamo[1], Pedro Revenga[1] and Luis M. Bergasa[1]

***Abstract*— Autonomous vehicle (AV) is one of the most challenging engineering tasks of our era. High-Definition (HD) maps are a fundamental tool in the development of AVs, being considered as pseudo sensors that provide a trusted baseline that other sensors cannot. Our approach is focused on the use of OpenDRIVE standard based HD maps in order to conduct the different mapping and planning tasks involved in Autonomous Driving (AD). In this paper we present a method for exploiting the HD map potential for two specific purposes: i) Global Path Planning and ii) Monitoring the relevant lanes and regulatory elements around the ego-vehicle to support the perception module. Mapping and planning modules are connected to the other modules of the AV stack by using ROS (Robot Operating System). Our AD architecture has been validated both in local and CARLA Autonomous Driving Leaderboard cloud, where we can appreciate a considerable improvement in the metrics by incorporating information from the HD map, not only used to conduct the Global Path Planning task but also providing prior information to the Perception module. Code is available in https://github.com/AlejandroDiazD/opendrive-mapping-planning.**

I. INTRODUCTION

AVs need to locate themselves in the environment to know what is happening close to them, in order to make decisions and execute a correct navigation like a human driver would. When we talk about localization, the first thing we need is a map where to be located and, particularly for vehicles, this is a road map. Road maps used by current navigators, e.g. Google Maps, are not valid for AVs because of data type and the fact that they are not accurate enough to be used for autonomous driving. Here is where HD maps appear to solve this problem. The concept of an HD map is a text file describing the real-world features related to the road map and its location within a 2D or 3D space and can do things that other sensors cannot [1]: First, they have an "infinite range" and, therefore, can "see" even into occluded areas. Second, maps will never fail due to environmental conditions. Lastly, maps contain highly refined data. This information can be used by different modules of an AV, (including self-localization, vehicle control, path planning, perception and system management) drastically reducing the computational load and complexity in comparison to other

*This work was supported by 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.

[1]Alejandro, Manuel, Ángel, Carlos, Pedro and Luis M. are with Department of Electronics, Polytechnic School, University of Alcalá, 28805 Alcalá de Henares, Madrid, Spain. {alejandro.diazd, manuel.ocanna, angel.llamazares, carlos.gomezh, pedro.revenga, luism.bergasa}@uah.es

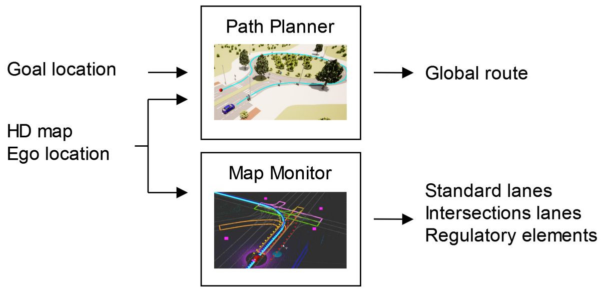

Fig. 1: Path Planning and Map Monitoring Outline

more complex methods, providing robustness and reliability to the system. There is not a standard definition about which data an HD map may include but, in the case of road maps, we can consider: roads, lanes, regulatory elements, topological information or any other information that can be considered relevant to drive in a safety way. HD maps can be used to alleviate the computational load of other modules and add confidence to the real-world model. In our case, it is used for two different purposes (see Fig. 1):

- **Planning**: Global Path Planning using as inputs the HD map, ego-vehicle and goal locations.
- **Monitoring**: Monitored Area close to the ego-vehicle using as inputs the HD map and ego-vehicle location. The relevant monitored elements are: standard lanes, intersection lanes and regulatory elements.

The combination of HD maps with onboard sensors and control systems leads to high-performance driving at the limits and is a really promising approach to enable autonomous driving [2]. One of the most challenging points of HD maps is map generation, which is a very difficult task that can be tackled using two different approaches:

- **Dynamic HD map on the cloud** that is being constantly updated by other users in real time.
- **Static offline HD map** previously generated.

Our work is focused on the second approach: a static offline HD map previously generated. There are two main ways for generating a prior HD map:

- **Automatically** from converted scans of the real world roads is the optimal long-term and scalable solution [3] [4], but it can become a complex task with high technical requirements depending on the map format in order to obtain a reliable method.
- **Manually** is also a simpler option but requires tons of human work, which takes a lot of time and is susceptible

to human errors. There are HD map designing tools as Java OSM Editor (JOSM) [5] for OpenStreetMap (OSM) [6] format or RoadRunner [7] for OpenDRIVE [8] format.

There are multiple formats of HD maps, both in open source formats and commercial solutions. Focused on open source solutions, in [9] is presented a comparative of the most popular formats. The two most relevant we have considered are OSM and OpenDRIVE. There are also available libraries for managing HD maps. In the case of OSM are Lanelet [10] and Lanelet2 [11], developed in C++, and in the case of OpenDRIVE is LibCarla, developed in C++ and presented with a documented PythonAPI [12].

In this project we have used OpenDRIVE format because it is the used by CARLA, the simulator we have used to validate our whole AD architecture. We have designed our own libraries, as many of the functionalities where not available in LibCarla.

The following sections will cover the technical details about how this work has been developed and the results obtained in our local tests, that reproduce an environment similar to the state-of-the-art CARLA Autonomous Driving Leaderboard [13]. Finally, we will highlight the conclusions obtained and the possibilities to continue extending this work in the future.

II. RELATED WORK

Using HD maps for autonomous driving is a very useful technique that has been used and studied since many years, both in private and open source research. In [14], the authors use the HD map concept for the iCity of the future, and the 3D mapping challenges it will present. But in the current state-of-the-art there are different ideas about how an HD map must be. There are works focused on generating the HD map from scratch using onboard sensors like in [15], where they use laser scans. Usual sensors for data acquisition are real-time-kinematic positioning, inertial measurement units, cameras and LiDARs. Other approaches start from a previously generated HD map, updating it at the same time that is used, as in the case of SLAMCU [16]. There are also other techniques that focus on using HD maps for different purposes and not on how the map is generated. The three main application using HD maps are:

- **Planning** [17], [18], [19]: A topological road graph can be generated for global planning. Also the geometries at lane level can be obtained for motion and local planning.
- **Perception** [20]: HD map can be used as prior information to overcome the limitations related to the on-board sensors, such as occlusions, poor performance under challenging weather conditions or sparse data.
- **Localization** [16], [21], [22]: Comparing static landmarks, as traffic lights, defined in the HD map with the equivalent information provided by perception systems the ego-vehicle location can be estimated.

We have identified that many works, as the ones mentioned above, make use of HD maps but without detailing which HD map format they use or how they use them. In many cases this is because a particular map format was developed for their work or for confidential issues, highlighting the lack of a standard for HD maps. The existing solutions can be classified in open source and commercial solutions, but in this work we will focus on the most popular in the case of the open sourced.

A. Open source solutions

There are many open source mapping solutions, some of them are compared in [9], in order to generate HD maps. But based on popularity and their characteristics we have considered OSM and OpenDRIVE as the best options:

- **OpenDRIVE**: It is a standard that models the road network along a reference line. Previous versions OpenDRIVE V1.4 and V1.5 were published by VIRES Simulationstechnologie GmbH, and originally was an open source standard. In 2018, the company ASAM was entrusted with the further development of OpenDRIVE, releasing their own version under the name of ASAM OpenDRIVE V1.6. Currently, the standard and all the documentation are still available free of charge on their web. OpenDRIVE V1.4 is used in CARLA simulator [23] and in our project, as well as maps generated with the application RoadRunner in its version V1.4. The file extension for maps of this format is xodr.
- **OSM**: This format allows to create maps using geographic points with different tags associated. Those tags are used to define their topology and how they are related to other elements. OSM maps can be downloaded from its web, or created using one of the multiple existing applications as JOSM. The file extension for maps of this format is osm.

B. Commercial solutions

In the case of commercial solutions is more difficult to know the technical details of their developments, but it is a fact that mapping companies are also working on HD map solutions as is the case of NVIDIA DRIVE and HERE:

- **NVIDIA**: NVIDIA DRIVE Mapping module is part of the NVIDIA End-to-End Autopilot Systems solution [24]. It is a scalable solution that combines a sensor suite, software and software APIs, with HD maps provided by mapping companies. It consists of: 1) DRIVE Localization for determining the precise 6-DOF position and orientation of an autonomous vehicle within an HD map with centimeter level accuracy. 2) Drive Mapstream for updating cloud based HD maps with road features perceived by DRIVE Perception. 3) Drive MyRoute for creating crowdsource maps of urban areas where there are not HD maps yet.
- **HERE**: HERE HD Live Map [25] is the solution by Here Technologies company. It uses machine learning to validate map data against the real world in real time. Their self-healing map analyzes data from multiple sources, such as satellite imagery and sensors from

OEM fleets in real-time, allowing the maps to always stay fresh and reliable.

After considering the different HD map solutions, we have not tried to create a new HD map format to satisfy our requirements because it would be contributing to avoid a standardization of the HD maps. Instead, we have studied the different existing formats to choose the one more appropriated to our project: OpenDRIVE. It offers a huge quantity of accurate metadata, but on the other hand the structure of the format is more complex than other standards because OpenDRIVE already has a road map structure implicit in the format. The complexity has been solved developing our own library together with the library LibCarla provided by CARLA simulator. The other main reason why we have chosen this format is that in our project we work with CARLA, and native format for the simulator maps is OpenDRIVE.

III. METHOD

We propose a complete method to work with OpenDRIVE based HD maps from map generation, then parsing the map file and finally using all this information for map monitoring and path planning (see Fig. 2).

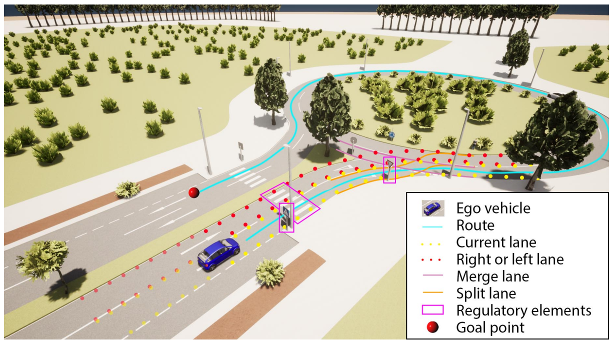

Fig. 2: Monitored Area and Path Planning in CARLA simulator

A. Map Generation

The map generation has been done using RoadRunner tool. This part includes all the process from the data acquisition until the final map file that is used in the simulator and by the mapping and planning modules. The map generation process can be summarized in three phases:

1) **Data acquisition**: The first step is to map discrete points of the real road boundaries using a Differential Global Navigation Satellite System (DGNSS). This data is saved in a gpx file, keeping a record of the geographical points.
2) **Map generation in Roadrunner**: The gpx file is imported in RoadRunner, so there is an accurate reference blueprint to define the roads with precision. In this way, a reference point is set so the map is georeferenced and Universal Transverse Mercator (UTM) system can be used.
3) **Map import in CARLA simulator**: Finally, the Roadrunner project is exported so it can be used in CARLA and in the real prototype. The files exported are fbx, xml and xodr. These three files are imported in Unreal Engine so the map can be used in CARLA. Moreover, only the xodr file is needed to be used by the map monitoring and path planning tasks.

B. Road Map Modelling

Once selected the HD map format, the first task is to model the roads in a text file (xodr in our case). In the case of OpenDRIVE, roads are generated from a sequence of segments along a reference line. The reference line is a virtual line that can be composed by a sequence of different geometries: straight, arc or spiral. All geometries that describe the road shape and other properties of the road are defined along the reference line. These properties are lanes, signals and other road elements.

C. Lanes Discretization

Lanes are generated from the reference line (ds), which is discretized as a list of n points along the total length of the line separated by a constant value d:

$$ds = [0, d, 2d, ..., nd] \quad \forall \quad n = [\frac{total\_lenght}{d}] \tag{1}$$

The different parameters can be seen in Fig. 3. The parameters needed to calculate a discrete XYZ point (red dots in Fig. 3) in the reference line are: x_0 and y_0 (coordinates xy where the reference line starts), hdg (heading at the beginning of the reference line), ds (distance along the reference line), $type$ (straight line or arc), $lane\_width$ (width of the lane of the central node) and curvature ρ (only needed in case of arcs). All these parameters are obtained from the HD map.

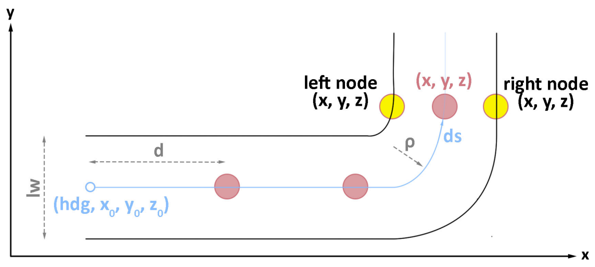

Fig. 3: Lane discretization from the reference line

In case of **straight lines**, the equations to locate a discrete position in the reference line are:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + ds \cdot \begin{bmatrix} c(hdg) \\ s(hdg) \end{bmatrix} \tag{2}$$

In Eq. 2 and the following equations, s and c are used instead of sin and cos respectively.

In case of **arc lines** the equation are:

$$radius = \frac{1}{\rho} \tag{3}$$

$$hdg_{aux} = hdg + ds \cdot \rho \tag{4}$$

$$\alpha = hdg + \frac{\pi}{2} \tag{5}$$

$$\beta = hdg_{aux} + \frac{\pi}{2} \tag{6}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + radius \cdot \begin{bmatrix} c(\alpha) & -c(\beta) \\ s(\alpha) & -s(\beta) \end{bmatrix} \tag{7}$$

The road elevation is calculated using the same polynomial function of third order in both straight and arc lines, so for the Z coordinate the polynom parameters needed are a, b, c and d:

$$z = a + b \cdot ds + c \cdot ds^2 + d \cdot ds^3 \tag{8}$$

Lanes are modelled generating points at both sides of the discretized reference line (centered at the lane), left and right nodes.

Lane boundaries can be obtained using these equations for each xy previously obtained:

$$\delta = hdg \cdot \frac{\pi}{180} \tag{9}$$

$$\begin{bmatrix} r.x & r.y \\ l.x & l.y \end{bmatrix} = \begin{bmatrix} x & y \\ x & y \end{bmatrix} + \begin{bmatrix} c(\delta) & -s(\delta) \\ -c(\delta) & s(\delta) \end{bmatrix} \cdot \frac{lw}{2} \tag{10}$$

The parameters of Eq. 10 are r (right node), l (left node), x and y (central node), lw (lane width) and α (heading in radians). The z parameter is the same in central, right and left nodes.

D. Map Parser

The Map Parser module is in charge of getting the information of the map from the OpenDRIVE file and transform it into custom classes that can be used by other modules like Planning or Perception. This module receives the OpenDRIVE file and analyzes every single text line, detecting the parameters and storing them into custom structured variables that have been previously defined. The only input is the OpenDRIVE file, and the output is a map object containing all the map parsed information.

E. Map Monitor

The Map Monitor module is the part in charge of monitoring the surrounding area of the vehicle and visualizing both the monitored area and the lanes describing the roads of the map, so we can consider two separated tasks: monitoring and visualization. The inputs of the Map Monitor are the information provided by the Map Parser and the waypoint route previously obtained by the path planner (section III-F), so it only monitors the elements around the route and the ego-vehicle location to know in which part of the route is the ego-vehicle. The outputs are the monitored elements published in ROS [26] topics using custom messages and markers published into ROS topics too so the elements can be visualized using RVIZ (3D visualization tool for ROS).

The operation of the Map Monitor is mainly organized into two callback functions: Route Callback and Location Callback.

1) Route Callback: This function is called when a route is published by the path planner. It divides the route in segments separated by a given distance and calculates in which segment of the route is the ego-vehicle, activating also a flag variable so the Map Monitor can start operating. In case it can not locate the ego-vehicle inside the route, it deactivates the Map Monitor.

2) Location Callback: Location callback is called every time that ego-vehicle position is published in its ROS topic, that in our case it happens at a frequency of 10 Hz. This callback checks some conditions like the flags generated in the Route Callback and, if every thing is correct, then calculates the monitored elements for a given distance. Finally, the monitored elements are published in their corresponding ROS topics. The threshold distance to monitor the current lane is obtained using a braking distance model that uses a linear regression with two arrays of velocity and braking distance data.

The monitored elements are:

- **Standard Lanes**: Current, back and the corresponding left and right lanes. Current lane is monitored from current position to a dynamic distance depending on the velocity of the ego-vehicle. Back lane is monitored from current position to back a proportional distance of the dynamic current lane obtained distance. Left and right lanes are monitored the same distance that current and back only if the lane marking from the HD map data allows the lane change.
- **Intersection Lanes**: Other lanes that intersect the current monitored lane are checked. Intersection lanes can have different roles (see Fig. 4): split (1 lane splits into 2 or more), merge (2 or more lanes merge into 1) and cross (a lane crosses a part of the current lane). To calculate the intersection lanes, each lane of every junction (junctions are areas where more than 2 roads meet) in the current lane is evaluated. The polygon of each lane is calculated and evaluated if is inside the polygon of the current lane. Roundabouts are considered as a set of multiple junctions.

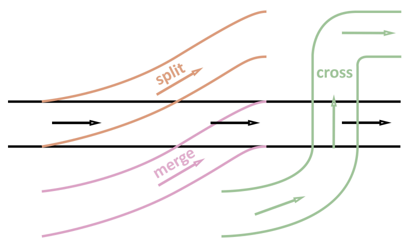

Fig. 4: Intersection Lane Roles

- **Regulatory Elements**: The monitored elements are stops, giveaways, traffic lights, speed limits and crosswalks. The regulatory elements are only monitored for the next intersection affecting the route.

F. Path Planner

The path planner is the module that receives ego-vehicle's position and goal position and calculates the route between them as a list of waypoints centered in the lanes. A waypoint is a structured object that represents a 3D point with location (x, y, z), rotation (pitch, yaw, roll) and some other topological information obtained from the HD map: road and lane ID, lane width, lane marking and speed limit of the road. To do that, it applies a Dijkstra algorithm to a road-lane graph previously generated from the HD map to obtain a topological (road-lane) route. Then, it generates waypoints centered in every lane of the route separated by a given distance. It also considers lane change when the order is published by the decision making module. So there is a fist topological route generated by the Lane Graph Planner (LGP) and then a waypoint route is generated by the Lane Waypoint Planner (LWP).

1) Lane Graph Planner: LGP generates a directed graph (DiGraph) of roads and lanes from the HD map using the Python module Networkx [27]. 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 speed allowed for each segment.

To generate 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 when it is allowed. Once we have the road graph, the route is calculated using an Dijkstra algorithm already implemented in the Networkx module as a method. The route returned by the LGP is a topological route as a list of tuples: (road, lane, action). Action can be lane follow, lane change right or lane change left. It could be used as an input to a perception based controller, but in our case a list of waypoints must be provided [28].

2) Lane Waypoint Planner: The LWP is in charge of calculating a list of waypoints separated by a given distance for every road-lane calculated by the LGP. For doing that, it uses the road-lane geometries obtained from the HD map and the mathematical discretization explained in section III-C.

IV. EXPERIMENTAL RESULTS

In this section we present the results obtained, both qualitative and quantitative. Finally, we discuss the results obtained in the local tests reproducing the CARLA Autonomous Driving Leaderboard.

A. Map Generation

The Map Generation method has been used to replicate an area of our university campus. Fig. 5 shows how the map has been generated from real data (Fig. 5a), imported into CARLA (Fig. 5b) and then visualized using RVIZ (Fig. 5c). The real AV prototype developed in our project currently uses this map to navigate and it works exactly the same as in the simulator for the path planning and map monitoring described in this paper.

Town03 (Number of graph nodes: 397)		
Algorithm	Execution Time [s]	Path Length [m]
A*	0.0268	823.50
Dijkstra	0.0259	823.50
Bellman Ford	0.0278	1083.98
Town10 (Number of graph nodes: 168)		
Algorithm	Execution Time [s]	Path Length [m]
A*	0.0116	350.41
Dijkstra	0.0111	350.41
Bellman Ford	0.0118	350.41
CampusUAH (Number of graph nodes: 276)		
Algorithm	Execution Time [s]	Path Length [m]
A*	0.0196	1280.84
Dijkstra	0.0183	1280.84
Bellman Ford	0.0195	1280.84

TABLE I: Results for different shortest path algorithms

B. Map Monitor

In Fig. 6 we can observe a qualitative result of the Map Monitor in an intersection of Town03 from CARLA. The monitored elements in the image are: global path with discretized waypoints, current and right standard lanes, merge, split and cross intersection lanes and traffic lights of the intersection.

The Map Monitor has been tested successfully in different maps and multiple use cases including complex intersections.

C. Path Planner

The path planner developed in this work has been tested for different routes and maps, and it has succeed reaching the goal in all the cases, regardless of the map used or the route length. Table I shows the results obtained for three routes in three different maps, applying the most popular algorithms for path planning: A*, Dijkstra and Bellman Ford.

The parameters evaluated in Table I are: 1) Execution time, 2) Path length and 3) Number of nodes that form the graph map. We observe that the path generated by the three algorithms is the same, except for Bellman Ford in one of the cases that generates a longer path. The execution time increases proportionally to the number of nodes that form the map graph, being Dijkstra always the fastest. The execution time is valid for our system, considering that global route is only calculated once at the beginning of a new route or re-planning. Dijkstra is chosen due to obtain the best results in all the cases.

D. CARLA Autonomous Driving Leaderboard

The leaderboard faces the agent to multiple traffic situations based on the NHTSA typology [29], characterizing its driving proficiency by multiple metrics. The modality chosen in this work has been the *Map Track*, where an HD map is given as one of the inputs.

In order to validate the system, it has been tested previously in local tests and then submitted to the CARLA Leaderboard cloud, due to the cloud tests only provide the metrics but not the whole process feed-back for debugging purpose. We have reproduced the leaderboard simulation conditions in our local environment, generating exactly the same metrics that are defined in it:

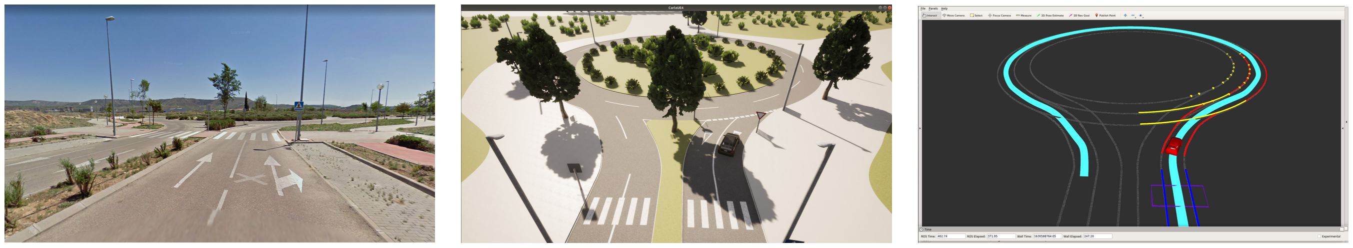
(a) Real environment (b) CARLA Simulator (c) RVIZ

Fig. 5: Different phases of the map generation process of our University Campus

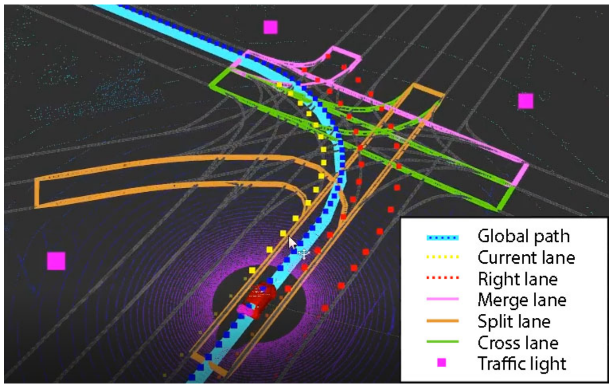

Fig. 6: Monitored Area in RVIZ

Driving Score (DS) is the most representative metric of the global result, being a product between *Route Completion* and *Infraction Penalty*. *Route Completion (RC)* is the percentage of the route completed. *Infraction Penalty (IP)* is a metric that combines all the other specific infractions: The agent starts with an ideal 1.0 IP, which is reduced whenever an infraction is committed. For these three metrics, higher value is better. The infractions considered are: *Collisions with Pedestrians (CP)*, *Collisions with other Vehicles (CP)*, *Collision Layouts (CL)*, which means collisions with static elements, running a *Red Light Infraction (RLI)* and running a *Stop Sign Infraction (SSI)*. There are other events considered that can interrupt the simulation, preventing the agent to continue the route. These events can be triggered because of a *Route Deviation (RD)*, a *Route Timeout (RT)* if the simulation is taking too long to finish the route and *Agent Blocked (AB)* if the agent is blocked for too long without taking any action. For the infractions and the events described, lower value is better.

After running 5 different scenarios for each town in 6 different CARLA towns, we have obtained the following results for two incremental experiments of our AV architecture. Experiment 1 has a basic version of the Map Monitor that only considers the current lane. Experiment 2 makes use of the full Map Monitor system, in addition to other improvements of the perception and decision making modules according to the monitored elements provided. The path planner is the same for both experiments, and has been used for calculating the route in all the cases.

Table II and Table III show the results for the experiments 1 and 2 respectively. The Route Length (RL) average has been also included in both tables. The metrics of DS, RC and IP are the average values for each of the towns. The other metrics have been normalized per kilometer.

As we can observe, comparing both tables, the results have improved in most of the cases for the experiment 2 that includes the full Map Monitor. Infraction penalties have been reduced thanks to the security module that makes use of the Map Monitor, providing a prior information about where to look for obstacles and regulatory elements. The RC has been improved because the number of events that interrupted the simulation has been reduced. These events are mainly AB events, that mostly occurred at intersections, and have been reduced due to monitored intersection lanes. If we focus on the global metrics, the improvements are: **+137.67% DS, +23.72% RC and +63.88% IP**. The low values of the RD validate the proficiency of the path planner working together with the control module.

Finally, our full Map Monitor has been evaluated within our full AV stack in the CARLA Autonomous Driving Leaderboard on the cloud [30] obtaining the second place in the *Map Track*.

V. CONCLUSIONS AND FUTURE WORKS

We have presented a method that exploits the potential of HD maps, specifically OpenDRIVE format, for path planning and map monitoring. The developed path planner applies a shortest path graph based algorithm to a graph previously generated from the OpenDRIVE map. The Map Monitor proposed is a novel technique of using HD maps to provide a priori information of relevant lanes and regulatory elements to other modules of the AV, adding confidence to the system. The method has been validated within a full AV stack in a local tests of the CARLA Autonomous Driving Leaderboard and in the cloud, proving the proficiency of the path planner and showing how the different metrics improve after adding the Map Monitor that is used by the perception module to create a security module. The system has been also tested in a real prototype, showing how the method works both for simulation and real cases. For future works, we want to go deeper in perception systems to detect static new elements of the map, generating a perception based map monitor updater for real time dynamic maps that reduce the impact caused by positioning errors. In addition, more complex scenarios will be consider to make a more reliable validation of the method. Code is available in GitHub.

Town	RL [m]	DS [%]	RC [%]	IP [0,1]	CP [n/km]	CV [n/km]	CL [n/km]	RLI [n/km]	SSI [n/km]	RD [n/km]	RT [n/km]	AB [n/km]
Town01	680.04	55.12	75.12	0.80	0.00	0.29	0.00	0.59	0.00	0.00	0.00	0.59
Town02	899.57	53.12	78.62	0.65	0.00	1.11	0.00	0.22	0.00	0.00	0.22	0.67
Town03	1520.76	2.36	77.02	0.04	0.13	2.50	0.39	1.45	1.45	0.13	0.00	0.13
Town04	2066.07	4.18	92.25	0.05	0.00	0.77	0.00	2.81	0.97	0.00	0.00	0.19
Town05	1043.06	9.81	38.60	0.48	0.00	1.53	0.19	0.77	0.19	0.38	0.00	0.38
Town06	1655.34	11.25	90.7	0.13	0.60	1.57	0.00	0.85	0.12	0.00	0.00	0.12
Global Metrics	1310.81	**22.64**	**75.39**	**0.36**	0.09	0.99	0.07	0.85	0.35	0.06	0.03	0.26

TABLE II: Experiment 1: Local results with basic version of Map Monitor

Town	RL [m]	DS [%]	RC [%]	IP [0,1]	CP [n/km]	CV [n/km]	CL [n/km]	RLI [n/km]	SSI [n/km]	RD [n/km]	RT [n/km]	AB [n/km]
Town01	680.04	86.0	100.00	0.86	0.00	0.29	0.0	0.29	0.00	0.00	0.00	0.00
Town02	899.57	45.17	88.09	0.54	0.00	0.67	0.0	1.56	0.00	0.00	0.67	0.00
Town03	1520.76	56.44	86.56	0.7	0.13	0.13	0.0	0.39	0.00	0.00	0.00	0.13
Town04	2066.07	52.60	98.27	0.53	0.00	0.29	0.0	0.68	0.00	0.00	0.00	0.10
Town05	1043.06	38.42	86.76	0.46	0.00	0.96	0.0	1.15	0.19	0.19	0.00	0.00
Town06	1655.34	44.24	100.00	0.44	0.24	0.24	0.0	0.72	0.12	0.00	0.00	0.00
Global Metrics	1310.81	**53.81**	**93.28**	**0.59**	0.05	0.33	0.0	0.61	0.04	0.02	0.09	0.03

TABLE III: Experiment 2: Local results with full version of Map Monitor

REFERENCES

[1] K. Wong, Y. Gu, and S. Kamijo, "Mapping for autonomous driving: Opportunities and challenges," *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 1, pp. 91–106, 2021.

[2] S. Song, "Towards autonomous driving at the limit of friction," Master's thesis, University of Waterloo, 2015.

[3] F.-A. Moreno, J. Gonzalez-Jimenez, J.-L. Blanco, and A. Esteban, "An instrumented vehicle for efficient and accurate 3d mapping of roads," *Computer-Aided Civil and Infrastructure Engineering*, vol. 28, no. 6, pp. 403–419, 2013.

[4] M. Elhousni, Y. Lyu, Z. Zhang, and X. Huang, "Automatic building and labeling of hd maps with deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13255–13260, 2020.

[5] "JOSM (Java OpenStreetMap Editor)." `https://josm.openstreetmap.de/`.

[6] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ." `https://www.openstreetmap.org`, 2017.

[7] "Roadrunner." `https://es.mathworks.com/products/roadrunner.html`.

[8] "Asam opendrive." `https://www.asam.net/standards/detail/opendrive/`.

[9] J. Godoy, A. Artuñedo, and J. Villagra, "Self-generated osm-based driving corridors," *IEEE Access*, vol. 7, pp. 20113–20125, 2019.

[10] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 420–425, IEEE, 2014.

[11] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *Proc. IEEE Intell. Trans. Syst. Conf.*, (Hawaii, USA), November 2018.

[12] "PythonAPI." `https://carla.readthedocs.io/en/latest/python\_api/`.

[13] "Carla autonomous driving challenge." `https://leaderboard.carla.org/challenge/`.

[14] H. G. Seif and X. Hu, "Autonomous driving in the icity—hd maps as a key challenge of the automotive industry," *Engineering*, vol. 2, no. 2, pp. 159–162, 2016.

[15] A. Barsi, V. Poto, A. Somogyi, T. Lovas, V. Tihanyi, and Z. Szalay, "Supporting autonomous vehicles by creating hd maps," *Production Engineering Archives*, vol. 16, 2017.

[16] K. Jo, C. Kim, and M. Sunwoo, "Simultaneous localization and map change update for the high definition map-based autonomous driving car," *Sensors*, vol. 18, no. 9, p. 3145, 2018.

[17] K. Zhang, S. Wang, L. Ji, and C. Wang, "High definition map based motion plan and control of autonomous vehicle on structured road," in *IOP Conference Series: Materials Science and Engineering*, vol. 825, p. 012018, IOP Publishing, 2020.

[18] A. Sadat, S. Casas, M. Ren, X. Wu, P. Dhawan, and R. Urtasun, "Perceive, predict, and plan: Safe motion planning through interpretable semantic representations," in *European Conference on Computer Vision*, pp. 414–430, Springer, 2020.

[19] R. Liu, J. Wang, and B. Zhang, "High definition map for automated driving: Overview and analysis," *The Journal of Navigation*, vol. 73, no. 2, pp. 324–341, 2020.

[20] B. Yang, M. Liang, and R. Urtasun, "Hdnet: Exploiting hd maps for 3d object detection," in *Conference on Robot Learning*, pp. 146–155, PMLR, 2018.

[21] H. Cai, Z. Hu, G. Huang, D. Zhu, and X. Su, "Integration of gps, monocular vision, and high definition (hd) map for accurate vehicle localization," *Sensors*, vol. 18, no. 10, p. 3270, 2018.

[22] F. Ghallabi, F. Nashashibi, G. El-Haj-Shhade, and M.-A. Mittet, "Lidar-based lane marking detection for vehicle positioning in an hd map," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2209–2214, IEEE, 2018.

[23] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[24] "Nvidia drive mapping." `https://developer.nvidia.com/drive/drive-mapping`.

[25] "Here hd live map." `https://www.here.com/platform/automotive-services/hd-maps`.

[26] 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.

[27] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[28] 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.

[29] "Nhtsa typology." `https://www.nhtsa.gov/`.

[30] C. Gómez-Huélamo, D.-D. Alejandro, M. E. Ortiz, R. Gutiérrez, F. Arango, J. Araluce, A. Llamazares, and L. M. Bergasa, "How to build and validate safe and reliable autonomous driving stacks? a ros based software modular baseline," in *2022 IEEE Intelligent Vehicles Symposium (IV): In submission*, IEEE, 2022.