Simulating use cases for the UAH Autonomous Electric Car

Carlos Gómez-Huelamo[1], Luis M. Bergasa[1], Rafael Barea[1], Elena López-Guillén[1], Felipe Arango[1], Pablo Sánchez[2]

***Abstract*— This paper presents the simulation use cases for the UAH Autonomous Electric Car, related with typical driving scenarios in urban environments, focusing on the use of hierarchical interpreted binary Petri nets in order to implement the decision making framework of an autonomous electric vehicle. First, we describe our proposal of autonomous system architecture, which is based on the open source Robot Operating System (ROS) framework that allows the fusion of multiple sensors and the real-time processing and communication of multiple processes in different embedded processors. Then, the paper focuses on the study of some of the most interesting driving scenarios such as: stop, pedestrian crossing, Adaptive Cruise Control (ACC) and overtaking, illustrating both the executive module that carries out each behaviour based on Petri nets and the trajectory and linear velocity that allows to quantify the accuracy and robustness of the architecture proposal for environment perception, navigation and planning on a university Campus.**

keywords: simulation, behaviours, autonomous vehicles, ROS, V-REP

I. Introduction

Autonomous vehicles are one of the greatest engineering challenges of our era. It must be able to navigate without making mistakes, consequently it has to understand the environment. Since the first projects in the 80s (PROMETHEUS in Europe [1] and Navlab in US [2]) many initiatives were launched by universities, research centers and automobile companies. DARPA Urban Challenge provided a breakthrough in self-driving technology [3].

Despite all the impressive efforts in the development of autonomous systems, fully autonomous navigation in arbitrarily complex environments is still decades away. The reason for this is two-fold: Firstly, autonomous systems, which operate in complex dynamic environments, require artificial intelligence that generalizes to unpredictable situations and reasons in a timely manner. Secondly, informed decisions require accurate perception, but so far, most of the existing computer vision systems produce errors at a rate not acceptable for autonomous navigation [4].

Autonomous car control systems must be able to take driving decisions based on the prior knowledge of the systems (road maps, traffic rules, sensor models and vehicle dynamics) and observations about the particular traffic situation. The observations come from the perception system that can include different on-board sensors such as odometry, cameras, LiDAR or GPS. At the same time, the decisions are variables that control the vehicle motion. Most of the implementations of these decision-making systems break down hierarchically into four components: route planning, executive layer, motion planning and vehicle control.

Our architecture is based on the open source Robot Operating System (ROS) that was launched to make easier the implementation of perception, mapping, localization and planning methods for robots autonomous navigation, providing a dynamic middleware with publisher/subscriber communication and a remote-procedure call mechanism. ROS-based systems provide an operating system-like services to operate robots with the fusion of multiple sensors data and time stamp of different devices [5].

In that sense, this paper presents and validates a novel ROS-based autonomous navigation architecture Fig. 1 based on hierarchical interpreted binary Petri nets to implement the decision-making framework, developed by the authors, in V-REP Fig. 2(a) [6], one of the most used 3D simulators in the field of robotics, describing some very interesting driving behaviours inspired in the CARLA Autonomous Driving Challenge [7], and finally showing both qualitative and quantitative results. This validation represents the preliminary stage for implementing these use cases in a real environment using our UAH electric car prototype.

II. Related Works

Behavioral decision-making should provide tools to define the traffic rules and model the sequence of actions and events that can take place in the different driving scenarios (overtaking, pedestrian crossing, stop, give way, ACC, etc.). Different approaches have been proposed for decision-making systems with the aim of self-driving. These approaches include different heuristic solutions [3] based on the concept of identifying a set of driving contexts or driving scenarios (lane driving, intersection handling, etc.), each of which requires the vehicle to focus on a reduced set of environmental features.

In addition, the presence of reliable simulation use cases take even more importance in order to check the behaviours that a car can face in complex driving situations which must be managed by using this decision-making layer. Decision making for autonomous driving is challenging due to uncertainty in the knowledge about the state of the vehicle

*This work has been funded in part from the Spanish MINECO/FEDER through the SmartElderlyCar project (TRA2015-70501-C2-1-R, TRA2015-70501-C2-2-R)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]Carlos Gómez-Huelamo, Luis M. Bergasa, Rafael Barea, Elena López-Guillén and Felipe Arango are with the Electronics Department, University of Alcalá (UAH), Spain. cram3r95@gmail.com, {luism.bergasa, rafael.barea, elena.lopezg}@uah.es, juanfelipe.arango@edu.uah.es

[2]Pablo Sánchez is with the Department of Systems Engineering and Automation, University of Vigo, Pontevedra, Spain, pasanchez@uvigo.es

and particularly the driving situation. This uncertainty comes from different sources, such as that introduced by observers. Especially challenging is to estimate the continuous state of nearby external agents such as other vehicles or pedestrians, since their behavior is, in general, unpredictable.

To design an optimal decision system that takes into account uncertainty, Partially Observable Markov Decision Processes (POMDP) offer a theoretically grounded framework to capture the fact that the decision system does not know the underlying state of the system [8]. Even though POMDP appears to be a good solution, in practice, due to the complexity of these systems, they cannot scale to real-world scenarios.

Finite-state machines FSM [9], hierarchical finite-state machines [10] and decision trees are among the most popular solutions. The winning team (Tartan Team) in the DARPA Urban Challenge [11] used a hierarchical finite-state machine to implement their behavior generation component. For complex problems, the difficulty in representing the system as FSM is to deal with the state explosion problem and the need to explicitly implement the potentially high number of transitions. Some researchers have considered Behaviors Trees (BT) because the transitions between states are implicit to the structure of the BT. According to [12], one of the major benefits attributed to BTs over FSMs is superior maintainability and extensibility.

However, one of the main problems of BTs is that a naive implementation exhibits blocking behavior and, in order to make them suitable for autonomous driving, this limitation needs to be addressed. Parallel activities or concurrency can be easily expressed in terms of Petri nets. In an FSM there is always a single current state. Petri nets are a powerful tool to model, design and analyze distributed, sequential and concurrent systems. In Petri nets there may be several states any one of which may evolve by changing the state of the Petri net. In the approach presented here, every behavior is modeled as a Petri Net. The main decision-making system that decides what behavior to execute is also modeled as a Petri Net, as shown in 2(b). In particular, we use hierarchical interpreted binary Petri nets [13] where a Petri net can start another Petri net or stop any of the Petri nets that have started.

On the other hand, most used 3D simulators in the field of robotics are V-REP and Gazebo, because of their ease integration in ROS. Other simulation environments are Microsfot Airsim [14], initially designed for drones but recently updated so as to include autonomous vehicles, ROS development studio [15] (based 100 % on Cloud, so a system of computers allows the parallel training of as many robots as required) and CARLA [7], which is one of the newest open source simulator for autonomous vehicles based on Unreal engine, with a recent release of its ROSbridge. Despite of the fact that our simulation use cases are based on CARLA Challenge and future works will try to adapt our system architecture to CARLA, for this paper we decided to use V-REP simulator since the performance was very similar to Gazebo and the group had previous experience in the use of this simulator.

III. Autonomous Navigation Architecture

The navigation framework is a modular architecture where individual modules asynchronously process information. These modules are independent processes that communicate with each other using the ROS inter-process communication system (PCS). In particular, the publish/subscribe paradigm is used in order to provide non-blocking communications. Each module corresponds to an independent Linux process running on different ECUs (Electronic Circuit Unit). Software modules are organized in four sets:

1) The hardware drivers layer: includes a set of programs that control different hardware devices that comprises sensors and actuators.

2) The control layer: implements the basic control and navigation functionality. It also contains the reactive control (local navigator), localization (localization), path planning (map manager) and a program that processes most of the exteroceptive sensors to detect relevant events (event monitor).

3) The executive layer: coordinates the sequence of actions that need to be executed by other modules to carry out the current behavior.

4) The interface layer: consists of a set of processes to interact with the users and to connect to other processes for multi-robot applications.

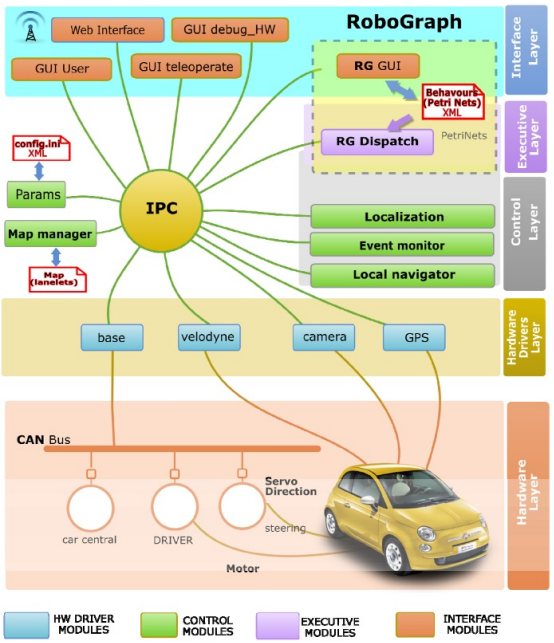
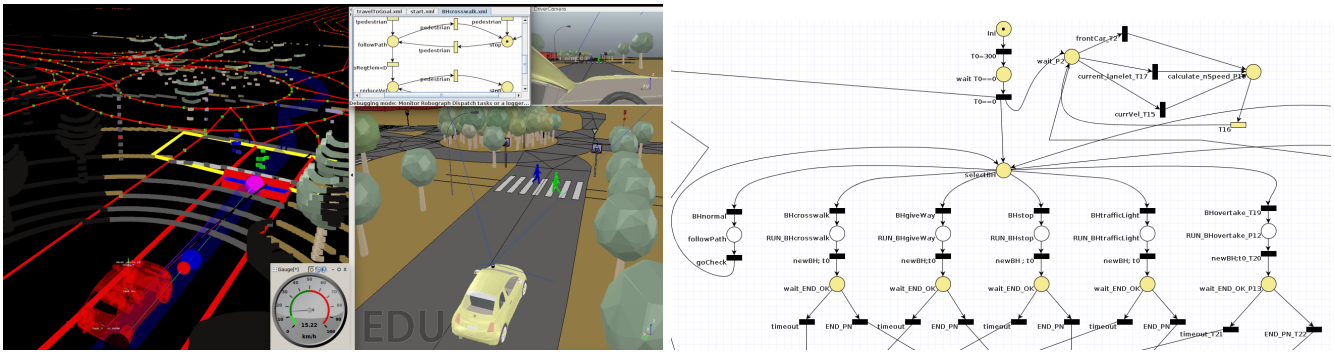


Fig. 1. Proposed Autonomous navigation architecture

Environment Perception is based on the fusion of LiDAR and camera information. Our motion control is divided into high-level planning and lower-level reactive control. First, the high-level planning calculates a path consisting of a sequence of lanelets [16] that can be modified depending on the performed behaviours by the executive layer. The goal



(a) (b)

Fig. 2. (a) Simulation example: On the left the RVIZ simulator illustrates the point cloud detection. On the right, V-REP simulator shows the environment which our sensors face. (b) Main Petri Net of our decision-making system.

of the local navigation system is to safely follow the path, keeping the car within the driving lane, and following the behaviours constraints established by the high-level planning. To do that, it obtains the curvature to guide the car from the current position to a look-at-head position placed in the center of the lane using the Pure Pursuit approach [17]. This curvature is used as the reference for an obstacle avoidance method based on the Beam Curvature Method (BCM) [18]. This approach allows to maintain the vehicle centered in the lane while is able to avoid unknown obstacles that can partially block the lane.

The decision making is implemented through a Petri net that takes as inputs the local perception (provided through the event monitor module), the map manager information and the vehicle localization in the map. In concrete, we use hierarchical interpreted binary Petri nets, where a net can start another net or stop any of the already started ones. To implement them we use the same tool (RoboGraph) employed by the authors in other mobile robot applications [19], but not in a real prototype of autonomous electric vehicle, as presented in this paper.

IV. Simulation Stage

V-REP is a multiplatform simulation software developed by Coppelia Robotics GmbH [20]. A fully functional free version is available for researchers.

A. Environment

The environment is modeled both geographic and topologically by using the lanelet approach presented in [16] and OpenStreetMap (OSM) service. Lanes and connections among them are manually delimited, including regulatory traffic information, to generate an enriched map useful for navigation.

The map manager loads the map and is in charge of planning a new path as a sequence of lanelets where each lanelet is defined by two bounds (left and right) named ways. Besides the path, this module serves other queries from other modules related to the map. For example, it should provide the contiguous lanes for the overtake maneuver, the lanes of an intersection to the event monitor for the cross intersection maneuver and it should also provide the position of regulatory elements. For the navigation, three different planners are applied. First, a lanelet path is obtained using an A* algorithm from the lanelet maps. Then, as commented above, a global path planner calculates an executable path by the car that tries to go in the middle of the lanes using the Pure Pursuit approach, and finally a local navigation algorithm based on BCM is executed to perform different behaviors following this path and avoiding unexpected obstacles.

Roads and buildings are loaded using OBJ files from the OSM website. Native OSM format is converted to OBJ format by using the open source tool OSM2World. To create the OSM map we used WGS84 coordinates (made up by latitude, longitude and height) whilst the simulator works in Cartesian coordinates (UTM) relative to an origin (roughly the center of the campus). Transformations from one system to another are done using the libraries implemented in the ROS geodesy package.

B. Vehicle

The visual part of the car seen in simulation (Fig. 2(a)) has been designed to get the best similarity with respect to our real prototype of the University of Alcalá (UAH). Dynamic properties of the vehicle are formed giving a mass to cuboids (straight parallelepipeds) to simulate its inertia. In addition, it is equipped with four wheels with damping, propulsion engine, brakes, steering and sensors. The reference system of the car is centered on the rear axle, taking the X axe pointing to the front, the Y to the left and the Z above. The vehicle is controlled by speed commands (linear in X and angular in Z). The angular velocity at Z is positive according to the rule of the right hand, so the left turns are positive. These speed commands are interpreted by the low-level controller obtaining the corresponding turning angle and accelerations.

The steering wheel is simulated in V-REP as a dynamic system (with inertia and friction) operated by a servo-controlled motor in position. This means that the position of the wheel is internally controlled by V-REP through a PID. The simulator receives steering wheel position commands in the same way as the real vehicle. The current position of the steering wheel determines the steering angle of a central virtual steering wheel according to the tricycle model.

Through some geometric relationships, the orientation of the front wheels is established in such a way that the Ackermann arrangement is fulfilled.

C. Sensors and environment perception

A model is implemented for each of the main sensors aboard the vehicle.

1) LiDAR: Placed on the roof of the vehicle and in the center of it. The Velodyne VPL-16 model included by default in the V-REP sensor repository is used. This sensor is modeled as four equally spaced sensors sweeping a horizontal angle of 90º each one and a vertical one of +/- 15º with 16 different beams. The 360º is obtained fusing asynchronously the 4 sensors one after the other. This model has been modified to obtain a synchronized full sweep. The LiDAR information is published directly in PointCloud2 format.

2) ZED stereo camera: To model the ZED stereo camera, placed on the front of the vehicle just below the rear view mirror and facing the road, two RGB sensors are included, horizontally separated by a baseline. This camera incorporates an internal depth calculation module. To simulate this functionality, a vision sensor located in the middle of the two previous sensors is used, which directly returns a cloud of 3D points without error. Image messages are published with the X axis pointing right, the Y down and the Z inwards in the image plane.

3) GPS: To model a GPS is very easy because we transform the UTM coordinates of the rear axis central point to WGS84 coordinates.

Furthermore, semantic segmentation is carried out in the RGB images taken from the right camera of the stereo pair to detect the driving area and the different obstacles in the field of view of the camera, in an unified way. To do that, we use our Convolutional Neural Network (CNN) called ERFNet (Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation) [21]. After that, the 2D segmented pixels are merged with the 3D LIDAR point cloud. In this way, a coloured 3D point cloud is obtained according to the field of view of the camera, where each colour represents a kind of obstacle (vehicles, pedestrians, trees, etc.). The point cloud outside this field of view is not segmented. Obstacles that can interfere car navigation are obtained from the point cloud using coloring clustering. Velocity and direction of each obstacle are obtained using a Precision Tracker evaluator [22]. This information is published by the event monitor module.

V. Behaviours and Experimental results

Several tests have been carried out in the Campus of the UAH in Madrid. These behaviours are inspired in the CARLA Autonomous Driving Challenge [7]. In order to perform each behaviour, the control layer takes the information of the lanelets and nearest regulatory elements, provided by the map manager modules, so as to generate a certain velocity command for the low-level control, following the behaviour commanded by the executive layer through the respective Petri net. Results demonstrates that perception data, navigation algorithms, and the movement commands are synchronized by using the proposed ROS-based architecture. The performance of these behaviors is illustrated by plotting the linear velocity versus time and the described trajectory (Fig. 3).

Due to the size constraints of this paper we do not the Petri net associated to each behaviour in RoboGraph interface, although Fig. 2(b) illustrates the main Petri Net of our decision-making system. Moreover, Table I shows the main Petri nets used to manage the different traffic scenarios and their main features. Inputs are the events that can change the state of the Petri net (most of them correspond to ROS messages). Input modules are the modules that publish the events, such as GUI (module that creates, edits and monitors the tasks), RG Dispatch (executes the tasks) and event monitor commented above. The outputs are the actions (ROS messages) that are sent to these different modules.

Note that the first Petri Net (Background) is a net running always in background. This net is waiting for a message from the user requesting to execute some of the tasks that the car can carry out. Selector PN decides which behaviour to run, according to the traffic situation, and monitors the execution of the behaviours. Each Petri nets implements the behaviour that corresponds to a particular traffic situation.

Below are described, in terms of simulation, some of these interesting use cases based on urban environments in order to validate our proposed ROS-based architecture.

A. Stop behaviour

Intersections are some of the most common danger areas on the road, because there is often a significant number of elements to consider: 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.

This behaviour is triggered if the distance between the stop regulatory element and our car is 30 m. In that moment 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 *followPath* not modifying the current trajectory. However, even if no car is detected, if the distance to the reference line is lower than a threshold D, set in the algorithm, due to the traffic rules of the stop behaviour, the ego-car must always stop, so the car starts to send 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

Petri Net	Inputs	Input modules	Outputs	Output module	Nº nodes	Nº transitions
Background	Man/auto	GUI User	Run Selector	RG Dispatch	8	9
	goToPoint	GUI User	Stop selector	RG Dispatch		
Selector PN	Reg. Element (STOP ..)	Map manager	Run PedestrianCrossing	RG Dispatch	21	29
	Dist. Reg Element	Map manager	Stop PedestrianCrossing	RG Dispatch		
	End Reg. Element	Map manager	Run GiveWay	RG Dispatch		
	Reg. Element (STOP ..)	Event monitor	Stop GiveWay	RG Dispatch		
	End Reg. Element	Event monitor	Run STOP	RG Dispatch		
	FrontCarVel	Event monitor	...	...		
	Odom	Base				
FollowLane	Traffic sign (max speed, ...)	Event monitor	SetMaxVel	Local Navigator	6	8
	Force End	RG Dispatch	STOP	Local Navigator		
Pedestrian Crossing	NoPedestrian	Event monitor	WatchforPedetrians	Event Monitor	10	13
	Pedestrian	Event monitor	SetMaxVel	Local Navigator		
	DistToPedestrianCrossing	Map manager	StopAtPoint	Local Navigator		
	PedestrianCrossingOver	Map manafer				
	Force End	RG Dispatch				
	stopped	Local Navigator				
STOP	SafetoMerge	Event monitor	CheckSafeMerge	Event Monitor	9	12
	NotSafetoMerge	Event monitor	SetMaxVel	Local Navigator		
	DistToStop	Map manager	StopAtPoint	Local Navigator		
	StopOver	Map manager				
	Force End	RG Dispatch				
	stopped	Local Navigator				
GiveWay	SafetoMerge	Event monitor	CheckSafeMerge	Event Monitor	9	12
	NotSafetoMerge	Event monitor	SetMaxVel	Local Navigator		
	DistToStop	Map manager	StopAtPoint	Local Navigator		
	StopOver	Map manager				
	Force End	RG Dispatch				
e curren	stopped	Local Navigator				
Traffic Light	CheckForTrafficLight	Event monitor	CheckForTrafficLight	Event Monitor	10	12
	SafetoMerge	Event monitor	CheckSafeMerge	Event Monitor		
	NotSafetoMerge	Event monitor	SetMaxVel	Local Navigator		
	DistToStop	Map manager	StopAtPoint	Local Navigator		
	StopOver	Map Manager				
	Force End	RG Dispatch				
	stopped	Local Navigator				
Adaptive Cruise Control (ACC)	Current Velocity	Map manager	SetMaxVel	Local Navigator	4	6
	FrontCarVel	Event monitor				
	DistToFrontCar	Map manager				
Overtake	FrontCarVel	Event monitor	CheckLeftLane	Event Monitor	14	21
	SafeChangeLeftLane	Event monitor	CheckRightLane	Event Monitor		
	NotSafeChangeLeftLane	Event monitor	SwichtLeftLane	Local Navigator		
	SafeChangeRightLane	Event monitor	SwitchRightLane	Local Navigator		
	NotSafeChangeRightLane	Event monitor	CheckRightLane	Event Monitor		
	Odom	Base				
	OnLeftLane	Local Navigator				
	OnRightLane	Local Navigator				

TABLE I

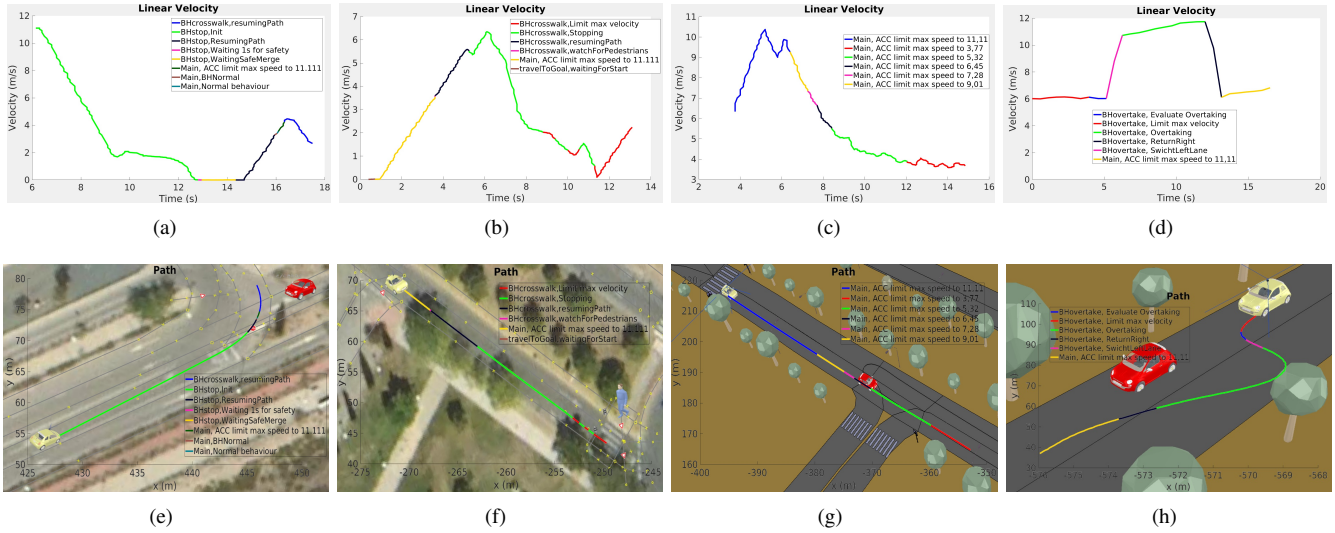
SUMMARY OF THE MAIN FEATURES, INCLUDING INPUTS, OUTPUTS AND NUMBER OF ELEMENTS, FOR THE MAIN PETRI NETS OF OUR WORK

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. For simplicity and extension of this paper, it is not shown both the stop behaviour with the presence and non-presence of detected car but we show the most representative, in this case stop with detected car.

Fig. 3(a) and 3(e) show the linear velocity and odometry projected onto the JOSM map, respectively, merging this information with the signals fired by the corresponding Petri net behaviour. It can be observed that the first 12 s the car is carrying out the stop Petri net (*BHstop, init*, green segment), since the distance to the regulatory element is lower than 30 m. The car decreases its velocity from 11 m/s (40 km/h, our campus speed limitation) to 0 m/s, because even there is a detected car or not, the ego-car must stop in this traffic situation. In this particular case there is a detected car, so the car keeps stopped (0 m/s) at the reference line waiting for safe merge (*BHstop, WaitingSafeMerge*, yellow segment) until the detected car is taken out of our field of view. Notice that this yellow segment can be observed in the linear velocity but not in the odometry because while the car keeps stopped, it does not move so the odometry is not modified. Finally, the car resumes the path (*BHstop, ResumingPath*, yellow segment) and the velocity is set at maximum again (*Main, ACC limit max speed to 11,111*, black segment) until the car faces another regulatory element.

B. Pedestrian crossing behaviour

A pedestrian crossing is the most basic regulatory element that helps people to cross a road. First, a node in the respec-



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 3. First and second row represent, respectively, the linear velocities and described trajectory of the Stop with car detected (a,e), Pedestrian Crossing (b,f), ACC (c,g) and Overtaking (d,h) behaviours

tive Petri net called *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:

1) 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.

2) Pedestrian detected: A pedestrian has been detected (*pedestrian* transition) in the pedestrian crossing area.

3) 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 (*T10* transition).

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* takes place when the car is closer than a threshold distance (D) 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”.

When the car gets close to the pedestrian crossing, it reduces its velocity and keeps following the path (*reduceVel* node) until the reference line of the pedestrian crossing is reached (*distToRegElem <minDist* 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.

Fig. 3(b) and 3(f) show the linear velocity and odometry projected onto the JOSM map, respectively, of the pedestrian crossing behaviour. It can be noticed that the simulation started with the car in static and a distance between the car and the pedestrian crossing exceeding 30 m, for that reason, the velocity is set at maximum at the beginning (*Main, ACC limit max speed to 11,111*, yellow segment). Notice that the final alternance between green and red segments is coherent, since the old pedestrian can disappear, so the PN switchs the token to the *reduceVel* node, and a new pedestrian can appear so the PN switchs the token again to the *stop* node.

C. Adaptive Cruise Control behaviour

Adaptive Cruise Control (ACC) is a behaviour that represents an available cruise control system for vehicles on the road that automatically adjusts the vehicle velocity to keep a safe distance from vehicles ahead. ACC technology is widely regarded as a key component of any future generations of autonomous vehicles. They improve the driver safety as well as increasing the capacity of roads by maintaining optimal separation between vehicles an reducing driver errors.

Fig. 3(c) and 3(g) show how the velocity of our car is decreased from 11,11 m/s (max velocity in the campus) to 3,77 m/s, that represents the velocity of the vehicle ahead. Note that even if the velocity of the front car is not constant but is changing with the time, the response of our car will also adjust to the velocity.

D. Overtaking behaviour

Overtaking is one of the most dangerous maneuvers and its automation is very challenging, especially on two-lane roads.

It comprises a sequence of lane-change and lane-keeping operations, during which the system must coordinate the car steering and speed to eliminate any potential collision with other vehicles. Overtaking another vehicle requires steering from the original lane to the adjacent lane, driving in this adjacent lane, and returning to the original lane in front of the overtaken vehicle. The executive layer decides when to overtake, heeding information provided by the map manager (is overtaking allowed in this lane according to traffic rules, and for how long?) and the event monitor module (what is the speed of the vehicle in front, and is it safe to move to the left lane?). There are three main states in this behavior according to the Table I: the car is changing to the left lane (switchLeftLane place), the car is in the left lane overtaking the other vehicle (overtaking place) and the car is returning to the right lane (returnRight place). The event monitor reports when the adjacent left lane is clear for a sufficient distance (leftLaneInfo transition) and the speed of the car it is going to pass (frontCar transition) while the map manager module knows the distance remaining in the current path where overtaking is allowed. Whenever one of these messages is received, the conditions are evaluated (evaluateCond place) and depending on the outcome of this evaluation the car might start switching to the left lane (switchLeft place) or keep waiting (start place). Similar sequences are also defined while the car is returning to the right lane (returnRight place).

Fig. 3(d) and 3(h) show the coherent steps that the overtaking Petri net should carry out: First, due to the presence of a vehicle ahead, the velocity is limited. Then, the overtaking is evaluated. If the output is true (overtaking is possible), our car speeds up and switch to the left lanelet. While the car is in the left lanelet, it maintains the velocity until the previous car is overtaken, and finally it returns to the right lanelet, finishing the overtaking PN and setting the velocity to the maximum.

VI. Conclusions and Future Works

This work presents how using a hierarchical-Petri-net-based programming environment, to implement the decision-making process of an autonomous navigation framework, and the V-REP simulator, to model real urban traffic scenarios in simulation, in order to validate our ROS-based architecture proposal for our autonomous vehicle. The perception system in which the different modules are based on is made up by the fusion of LiDAR, cameras and DGPS. The proposal has been validated in simulation with an exhaustive study of typical urban driving scenarios such as stop, pedestrian crossing, Adaptive Cruise Control and overtaking. As future work, all these behaviours will be translated both to the CARLA simulator, so as to get more challenging situations in order to improve the robustness and reliability of our system, and to our real electric car prototype, to validate these simulation use cases in the real world.

References

[1] E. D. Dickmanns, B. Mysliwetz, and T. Christians, "An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1273–1284, 1990.

[2] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie mellon navlab," in *High Precision Navigation*, pp. 97–122, Springer, 1989.

[3] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[4] J. Janai, F. Guney, J. Wulff, M. J. Black, and A. Geiger, "Slow flow: Exploiting high-speed cameras for accurate and diverse optical flow reference data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3597–3607, 2017.

[5] 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.

[6] C. Otero, E. Paz, R. Sanz, J. López, R. Barea, E. Romera, E. Molinos, R. Arroyo, L. Bergasa, and E. López, "Simulación de vehículos autónomos usando v-rep bajo ros," 2017.

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

[8] 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.

[9] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3, pp. 2436–2441, IEEE, 2003.

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

[11] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*, vol. 56. springer, 2009.

[12] M. Colledanchise and P. Ögren, "How behavior trees modularize robustness and safety in hybrid systems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1482–1488, IEEE, 2014.

[13] J. L. Fernández, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary petri nets to build robust mobile robot applications: Robograph," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1372–1377, IEEE, 2008.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[15] D. S. Michal and L. Etzkorn, "A comparison of player/stage/gazebo and microsoft robotics developer studio," in *Proceedings of the 49th Annual Southeast Regional Conference*, pp. 60–66, ACM, 2011.

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

[17] N. Y. Ko and R. G. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*, vol. 3, pp. 1615–1621, IEEE, 1998.

[18] J. L. Fernández, R. Sanz, J. Benayas, and A. R. Diéguez, "Improving collision avoidance for mobile robots in partially known environments: the beam curvature method," *Robotics and Autonomous Systems*, vol. 46, no. 4, pp. 205–219, 2004.

[19] J. López, D. Pérez, and E. Zalama, "A framework for building mobile single and multi-robot applications," *Robotics and Autonomous Systems*, vol. 59, no. 3-4, pp. 151–162, 2011.

[20] C. Robotics, "V-rep user manual," *URL http://www. coppeliarobotics. com/helpFiles/. Ultimo acesso*, vol. 13, no. 04, 2015.

[21] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[22] D. Held, J. Levinson, and S. Thrun, "Precision tracking with sparse 3d and dense color 2d data," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1138–1145, IEEE, 2013.