# Development of ADAS perception applications in ROS and “Software-In-the-Loop” validation with CARLA simulator

Stevan Stević, Momčilo Krunić, Marko Dragojević, and Nives Kaprocki, *Members, IEEE*

***Abstract* — Higher levels of autonomous driving are bringing sophisticated requirements and unpredicted challenges. In order to solve these problems, the set of functionalities in modern vehicles is growing in terms of algorithmic complexity and required hardware. The risk of testing implemented solutions in real world is high, expensive and time consuming. This is the reason for virtual automotive simulation tools for testing are heavily acclaimed. Original Equipment Manufacturers (OEMs) use these tools to create closed sense, compute, act loop to have realistic testing scenarios. Production software is tested against simulated sensing data. Based on these inputs a set of actions is produced and simulated which generates consequences that are evaluated. This creates a possibility for OEMs to minimize design errors and optimize costs of the vehicle production before any physical prototypes are produced. This paper presents the development of simple C++/Python perception applications that can be used in driver assistance functionalities. Using ROS as a prototyping platform these applications are validated and tested with “Software-In-the-Loop” (SIL) method. CARLA simulator is used as a generator for input data and output commands of the autonomous platform are executed as simulated actions within simulator. Validation is done by connecting Autoware autonomous platform with CARLA simulator in order to test against various scenes in which applications are applicable. Vision based lane detection, which is one of the prototypes, is also tested in a real world scenario to demonstrate the applicability of algorithms developed with simulators to real-time processing.**

***Keywords* — Autonomous driving, perception, ROS, CARLA, AUTOWARE, SIL, ADAS, C++, Python.**

Paper received May 01, 2020; revised July 17; accepted July 18, 2020. Date of publication July 31, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

*This paper is revised and expanded version of the paper presented at the 27th Telecommunications Forum TELFOR 2019 [23].*

This work was partially supported by the Ministry of Education, Science and Technological Development of Republic of Serbia under Grant III_044009_1.

Stevan Stević is with the RT-RK Institute for Computer Based Systems, Novi Sad, Serbia (e-mail: stevan.stevic@rt-rk.com).
Momčilo Krunić is with the Faculty of Technical Sciences, University of Novi Sad, Serbia, Novi Sad, Serbia (e-mail: momcilo.krunic@rt-rk.com).
Marko Dragojević is with the RT-RK Institute for Computer Based Systems, Novi Sad, Serbia (e-mail: marko.dragojevic@rt-rk.com).
Nives Kaprocki is with the RT-RK Institute for Computer Based Systems, Novi Sad and Faculty of Technical Sciences, University of Novi Sad, Serbia (e-mail: @rt-rk.com).

## I. Introduction

DEVELOPMENT of autonomous vehicles is a major trend in automotive industry. Pushing towards Society of Automotive Engineers (SAE) levels [1] four and five and fully automated vehicle as an ultimate product, engineers are facing issues that have never been addressed. As they progress with the development of some functionality new problems arise because of uncertainty of physical world, as there are many unpredicted situations that could cause accidents. Due to this, the development of virtual simulators to test vehicle’s cognitive computing [2] becomes a crucial part of the development. With this approach the perception module [3] receives input from computer-generated scenes and mathematically modelled movement patterns for pedestrians, bicycles, and other entities. An acting module [3] on the other side outputs commands to simulators that implement these as actions. Using this, billions of kilometres that are required [4] to demonstrate the reliability of autonomous vehicles in terms of fatalities and injuries, have been already simulated by OEMs [5]. By using simulator’s abstract visualizations, engineers can focus on the development of core capabilities for autonomous driving, such as: driving models and systems, remote assistance, mapping, localization, perception, etc.

This paper emphasizes and explains the importance of using simulators in the modern automotive development and gives a practical example by showing how simple Advanced Driver Assistance Systems (ADAS) applications can be tested within the context. The rest of the material is organized as follows. The first part explains current state of the industry, problems and practices used. Also, it provides some academic and industry related as a background. The third section explains platforms and tools that are used for development and simulations. Section IV describes the purpose of test applications and presents an existing setup for connecting Autoware [6] with CARLA and describes sensor and start up file configuration. Section V presents validation for these use-cases in simulators and real world, and final section concludes the paper with the review of work done and some future steps.

## II. Simulators

Simulators use different models of environments that can be built from high resolution LiDARs, cameras or even virtual maps that provide annotations with tools like OpenDRIVE [7]. Furthermore, some types of simulators can augment existing data like point cloud to create obstacles [8]. Based on this, modern day simulators like rFpro [9], LGSVL [10], AVS [11] by Uber, CARLA [12], DRIVE CONSTELLATION Simulator [13] by NVIDIA, Gazebo [14] and others are able to create very realistic scenes and complex layouts like road paint or road separation that can be difficult to discern even for humans. On top of that, these ecosystems are scalable and can implement different feature requests, unlike early development tools of this type that stretch their capabilities to satisfy different use-cases.

Simulators usually offer simulation of a wide variety of virtual sensors that can be placed on the ego vehicle to recreate real test vehicles. These sensors can provide a stream of different formats of data like LiDAR point clouds [15], RGB, HDR, depth in meters or material properties of objects and others. The data is produced based on a current scene in the simulator and virtual placement of the sensors.

The problem with using synthesized and generated data comes from the stochastic nature of the real world by which it avoids patterns found in these types of data and can notably impair the performance of trained neural networks or prediction models. Simulators tend to overcome these problems by allowing stochastic behaviours to be injected with disruptive, low likelihood-of-occurrence events. This is utilized for example, by creating unpredictable behaviours for drivers, on some random number of driven kilometres.

The ability to have a simulation that correlates closely with physical world, with different scenes, traffic flow and virtual sensors, provides engineers environment to create SIL [16] systems. This means that every entity like hardware, data, use-cases and such are simulated, whilst using production software to test against it. By testing some features earlier in the design cycle, that would otherwise require finished prototypes, changes are less expansive to implement, and problems are eliminated or predicted on time. Except these advantages, SIL testing also provides a level of certainty, because engineers can be sure that given behaviour is not due to any mechanical or electrical malfunctions but caused by a written software.

It is hard to say which simulators are "state-of-the-art" products, because these metrics can only be defined in the scope of requirements that are being tested. Raging from simulators for algorithm testing to sensor technology readiness level (TRL). However, latest simulators are being designed to work with a client-server architecture which is also the case with Gazebo and CARLA simulator. This approach generated a new set of scenarios where multiple ego vehicles can be used in the same scenes. Furthermore, an even bigger advantage is that a simulator can be cloud-based, simplifying configurations, organizations and even providing opportunities for research to departments that are lacking the technical or finical resources.

First, Gazebo is used, which is one of the most used simulators when working with Robot Operating System (ROS) [17] and connected it with the rest of the system as seen in Fig. 1. However, for creating certain use-cases and scenarios, CARLA is much more efficient, and it is mainly developed for autonomous vehicle, unlike Gazebo. Therefore, CARLA is chosen as a primary simulator for input data and command execution.

## III. Platforms and Tools

In order to fulfil the goal of using applications in SIL environment, fast prototyping tools are needed, an autonomous platform to build upon and a simulation tool. ROS is used for prototyping, which is the reason for initial use of Gazebo. ROS represents a software stack with a set of tools that can be used for robot control and analysis and for that used to put together various robotics solutions. It uses a concept of nodes (processes) that communicate between themselves using topics and services. This is orchestrated by one main node called Master. ROS also offers multilingual bindings making it easy to integrate with any existing code base. For these reasons many researchers and even companies use ROS as a base for autonomous agents. This is also the case with Autoware platform which provides functional components required to accomplish autonomous driving. Platform architecture and the set of algorithms in Autoware are well suited for urban driving scenarios. Also, for processing purposes the platform utilizes GPU optimized frameworks and libraries like OpenGL, CUDA, OpenCL, PCL etc.

A large code base, user community haven't been neglected by CARLA developers, so they used ROS Bridge [18] component for CARLA simulator integration. As the simulator itself is based on Unreal Engine 4 [19] and Python API there was a need to transform data into ROS format of messages for self-driving software to run on CARLA as it is, without modifications. This provided a ground to connect CARLA with Autoware, which is done with the "Autoware in Carla" [20] integration stack that relies on existing bridge.

The Autoware-bridge contains three Carla clients:

1. ROS Bridge - Monitors existing actors in Carla, publishes changes on ROS Topics (e.g. new sensor data) - *Data publishing*.
2. Ego vehicle - Instantiation of the ego vehicle with its sensor setup – *Ego Vehicle Setup*.
3. Waypoint calculation - Uses the Carla Python API to calculate a route – *Route planning*.

With these CARLA clients, overall architecture is presented in Fig. 2. For visualization of actions and information RViz tool from the ROS environment is used.

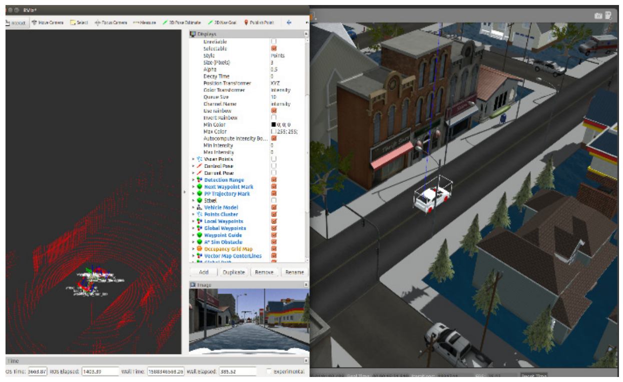


Fig. 1. Gazebo simulator on top of Autoware.

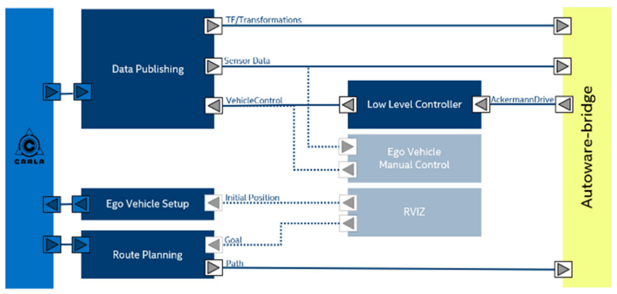


Fig. 2. Architecture of CARLA and Autoware integration.

## IV. System Integration

In this section, the development of demo applications and their integration with Autoware have been provided. Furthermore, sensor configuration for ego vehicle in CARLA simulator and configuration of start-up scripts that create virtual environment alongside a virtual vehicle is described. Some of the parameters given are weather conditions, number of vehicles and pedestrians, selection of maps with waypoints, etc.

### A. *Demo ADAS applications*

The first part of the integration was to implement three demo ADAS applications that could be used in simulation testing. One application was based on the implementation of occupancy grid framework [21] which is mostly used for mapping use-cases. The second one is the implementation of stopping distance monitor functionality which can be used in system like Forward Collision Warning (FCW). The third application handles vision-based detection of lane lines, calculation of their curvature and vehicle offset within a road lane in order to provide data for functions of lane keeping, Lane Change Warning (LCW) and Lane Change Assistant (LCA).

The first two applications were designed to be used with LiDAR data, more precisely point cloud input streams, while the third one uses front placed RGB camera with 1200x720 resolution. Autoware as a platform for an autonomous driving already has nodes that can publish point cloud data and raw image frames on defined topics. Because of this, applications were implemented as ROS packages within the Autoware platform.

#### 1) *Occupancy grid prototype*

The pipeline for this application can be seen in Fig. 3. Based on input data, a parameterized occupancy grid is created. The grid is parameterized by size, topic for point cloud data, and size of every cell with ego vehicle in the centre of the grid. After the data is received, transformation is done from LiDAR coordinate system to a vehicle perspective. Ego vehicle usually has a fixed frame for the centre of its own coordinate system that is used to align all algorithms, sensors and maps. This is also given as a parameter for package and rear axis is used in this demonstration. Next is filtering of Region of Interest (RoI) based on a given size to reduce the cost of updating cells. Parallel with this, ground points are being filtered, which is done to remove false detections and to provide a list of these points that could be used in detection of free space. The update of the cells is the end of cycle processing which is done according to log-odds notation. As all these operations on point cloud are expensive, therefore, C++ is used for performance reasons and many supporting libraries. For example, cloud processing and filtering is done with PCL library.

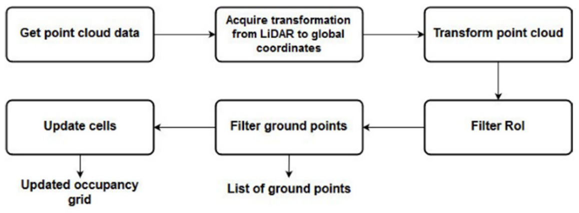


Fig. 3. Occupancy grid package processing.

#### 2) *Stopping distance monitor*

Implementation of the second application, the stopping distance monitor, required additional velocity of the vehicle. This is needed in order to calculate if the ego vehicle could stop before it hits the vehicle in front. In this example LiDAR data is used to determine distance between the vehicles. To acquire this data, the application must subscribe to topics that are advertised by corresponding nodes. Because of this, the application implements a communication adapter that handles communication with ROS stack. Using this design, computational logic can be independent of ROS and can be used in other future stacks. A communication module forwards data to the computational module. Point cloud data is clustered into groups and by determining the nearest detected cluster, which is in the vehicle path, distance between two vehicles is determined. This information, together with the current speed and maximum deceleration model can generate a warning if a vehicle would not be able to stop on time. This is done by a generation module which forwards the warning to the communication module that is also implemented to publish back into the ROS environment. For this a C++ client ROS API and library is utilized. This information can be used by different features to execute automatic braking or notify the driver. Design of the application can be seen in Fig. 4.

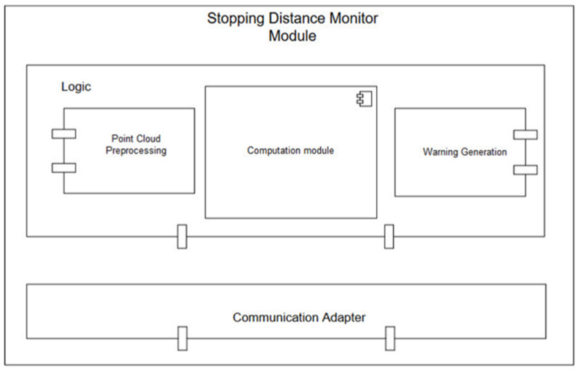


Fig. 4. Stopping distance monitor application design.

#### 3) *Lane and curvature detection*

A lane detection application [22] is written in Python unlike previous two. This is done by utilizing the multi-language bind libraries from ROS. Application subscribes to image_raw topic where generated image data is published. On image receive callback, a lane detection pipeline is triggered which can be seen in Fig. 5.

Images obtained in real world scenarios are distorted due to camera lenses. In order to have precise detection images must be undistorted. In the simulation environment, since the image is rendered, there is no distortion, so this pipeline block is added for a real-world test but it is visualized here for the complete picture.

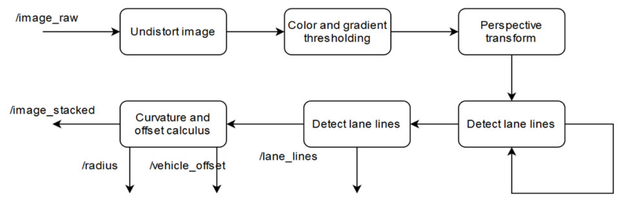


Fig. 5. Lane detection pipeline.

Next, to extract lane lines pixels, combined techniques of gradient and colour space (Hue, Saturation, Lightness - HSL) thresholding are used. However, this also leaves other parts of the image with high gradients. Again, by selecting four polygon points that include lane lines, RoI filtering is applied and transforms only that part of the image into birds-view perspective (Fig. 6).

This approach enabled us to use histograms to detected peaks in pixels along x axis, and with that, the start of the lane lines. The first of the 9 bounding rectangles, is placed at the centre of the peak for both lanes. Rest of the bounding rectangles are placed one on the other following the line, and with this, also detecting curved lines. This can be seen in Fig. 6.

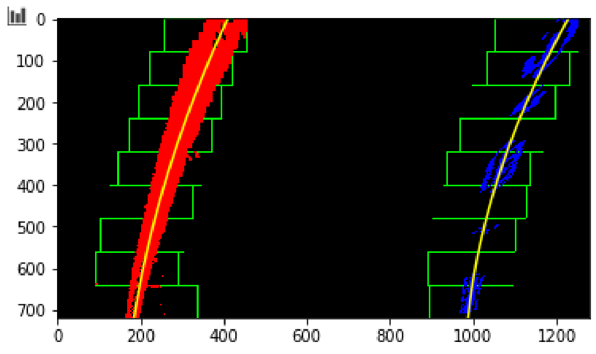


Fig. 6. Bird-view perspective and fitpoly() on pixels within bounding boxes.

After the lane pixels have been identified, polyfit() function is called to find line coefficients. These coefficients are published on /lane_lines topic and consumed by curvature and offset calculation node.

Optimisation is performed with a low pass filter on lanes to avoid sudden jumps due to noise or bad thresholding for a given frame. Another optimisation method is applying a targeted search around detected lines once it is found, to save processing time.

The last step is to calculate lane curvature and vehicle offset from a lane centre so that correct lateral control commands are executed to keep a vehicle in lane.

In the first stage of development, pre-recorded *rosbag* files have been used from real autonomous test drives. With this approach only recorded data is obtained and tests were run for visible markers in RViz visualization tool. The problem with this approach is that it does not provide a closed feedback loop, because it can't execute actions based on given commands. For this reason, CARLA simulator has been introduced.

## *B. Sensors*

The goal of integration with Carla is to get data from the simulator to the Autoware. This is done by utilizing a previously described setup with ROS Bridge for Carla messages and connection with Autoware. In order to send the simulated data the first step is to create it with sensors from Carla database. As both applications needed point cloud data, *lidar.ray_cast* sensor is instantiated on top of the vehicle. In addition to this *camera.rgb* sensor is added to visualize the scene in RViz and provide raw frames for lane detection nodes. Besides these that are directly important for the application, other sensors, like GNSS, for example, are instantiated, and a HD map is provided. Autoware nodes responsible for localization utilize this type of sensors. All sensors are described in JSON file that is consumed by simulator.

## *C. Scene configuration*

Final step needed to run the whole SIL setup, was to create virtual scenes. Carla already offers Python API to create diverse scenes with existing maps which were utilized to get an optimal setup for autonomous driving use-cases. Tests were run against sunny and rainy weather conditions, where algorithms, as expected, worked better in sunny conditions due to disruption of LiDAR beams with rain drops and reflections from a wet road that caused problems when thresholding the image.

Simplified integration of these applications can be seen in Fig. 7. Perception applications (ROS nodes) subscribe to required topics described for each application separately. Based on their algorithms they publish information which is consumed by "Waypoint Updater". This node subscribes to waypoints published by CARLA itself. Fusion of these waypoints and information from perception application produces a set of final waypoints.

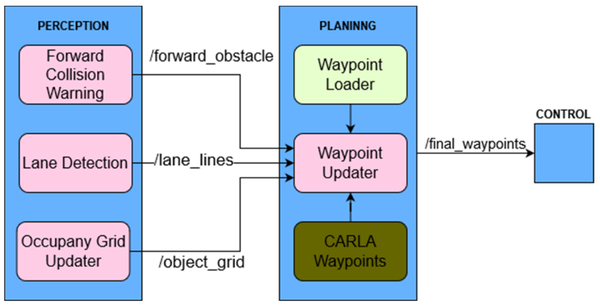


Fig. 7. Architecture of autonomous control system.

Information from a stopping distance monitoring node that contains coordinates of obstacle in front and impact severity, has a longitudinal effect in the generation of final set of waypoints. It determines how many waypoints are generated in distance, so if an obstacle is close and a possibility of impact is high then waypoints will not be generated.

Lane points acquired from "Lane Detection" node affect the position of the waypoint between two lanes and with that helps keep a vehicle in the center of the lane. For this there are other topics like "*/vehicle_offset*" and "*/(lane)radius*" as seen in Fig. 5.

Coordinates from side obstacles generated with "Occupancy Grid Updater" influence the generation of waypoints at turns.

## V. Validation

Once the custom scene was created, all the nodes and bridges started along with the vehicle model loaded in the environment with instantiated sensors. Manual system tests were controlled with RViz by setting a destination point towards which ego-vehicle moved by existing waypoints on the map.

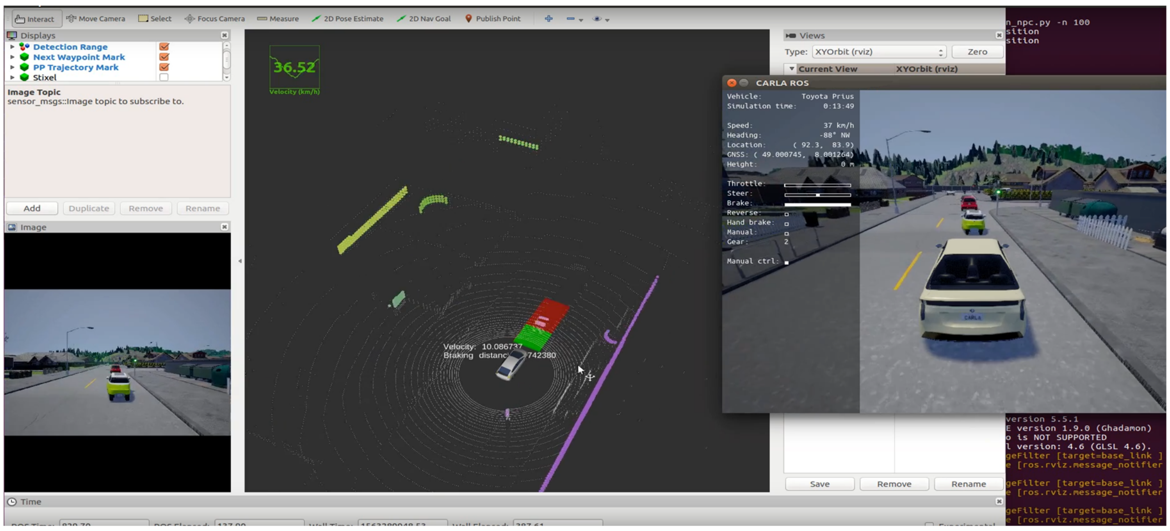
As depicted in Fig. 8, the vehicle model and the custom scene were created with Non-Player Characters (NPC) – vehicles. These vehicles are spawned deliberately in front of the ego vehicle to test if the vehicle will stop. Vehicle was aware of obstacles in front which is visualized with the red part of the rectangle and has generated the warning signals and executed stoppage of the manoeuvre. Similar use cases were created for a mapping application where as many objects as possible are spawned to test the application with different cell size and size of the affected frame

The RGB camera images (bottom left), joint motions and the LiDAR scans that were generated from Carla were visualized in RViz.

In Fig. 9 visualization of "*/image_stacked*" topic that is generated after finding lane lines, from the third application is seen. Image is overlayed with green space between two correctly detected lanes with curvature and vehicle offset.

As already mentioned, the following algorithm is tested in a real urban street scenario by placing the dash camera in front of the real car. Fig. 10 presents one frame from real time detection where lines are drawn on top of lanes in original frame.

Except for empirical confirmation of correctness, other formal methods were used during the testing. Applications were written by Test Driven Development methodology. This ensured testing on both unit and functional (acceptance) level of testing.



```
Image Topic
sensor_msgs::Image topic to subscribe to.
```

```
Type: XYOrbit (rviz)
Current View    XYOrbit (rviz)
```

```
Velocity: 10.086737
Braking distanc    742380
```

```
Vehicle:              Toyota Prius
Simulation time:          0:13:49

Speed:                    37 km/h
Heading:                  -88° NW
Location:          ( 92.3,   83.9)
GNSS: ( 49.000745,   8.001264)
Height:                       0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:       2

Manual ctrl:
```

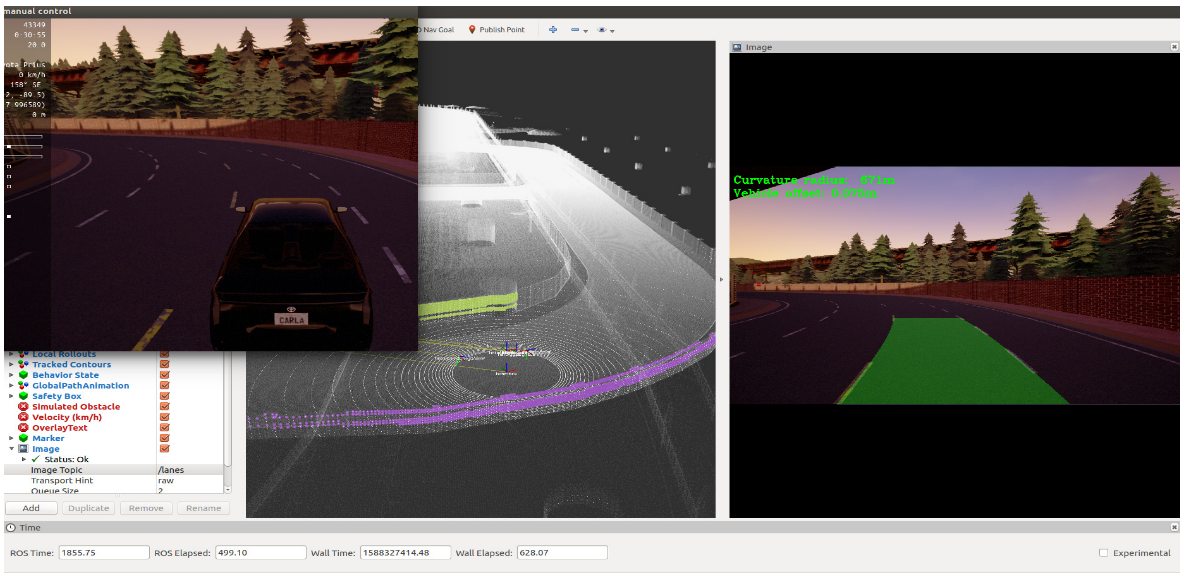
```
n_npc.py -n 100
sition
sition
```

```
version 5.5.1
E version 1.9.0 (Ghadamon)
o is NOT SUPPORTED
l version: 4.6 (GLSL 4.6).
geFilter [target=base_link ]
e [ros.rviz.message_notifier

geFilter [target=base_link ]
e [ros.rviz.message_notifier

geFilter [target=base_link ]
e [ros.rviz.message_notifier
```

Fig. 8. Visualization of FCW application integrated with CARLA simulator.



```
    43349
  0:30:55
     20.0

ota Prius
   0 km/h
  158° SE
2, -89.5)
7.996589)
      0 m
```

```
Status: Ok
Image Topic        /lanes
Transport Hint     raw
Queue Size         2
```

```
Curvature radius:  671m
Vehicle offset: 0.075m
```

```
ROS Time: 1855.75    ROS Elapsed: 499.10    Wall Time: 1588327414.48    Wall Elapsed: 628.07
```

Fig. 9. Visualization of Lane detection integrated with CARLA simulator.

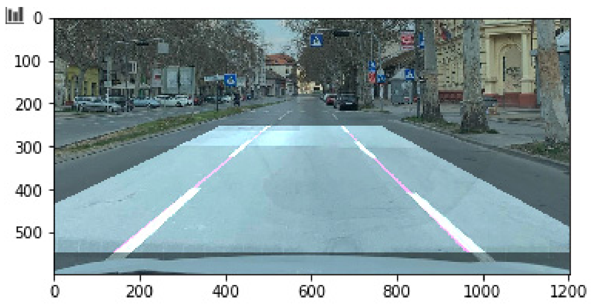


Fig. 10. Lane detection (pink) in real world.

The most important tests were executed at acceptance level where output commands from the system treated as block-box are evaluated. Predefined environment stimulus with expected results is defined. This includes a saved segment of the route with known waypoints, obstacle coordinates, lane waypoints and acceptable vehicle states.

The autonomous control system architecture under test, starts by loading mapped route base waypoints with "Waypoint Loader" and publishing them. The system then starts receiving the car's sensor data and computes the final set of waypoints as previously described. Since the scenarios are known and the expected set of waypoints is known it is easily determined if tests are passing successfully. Valid expected results also include small deviations in the position of generated waypoints. These deviations are allowed if generated waypoints will keep a vehicle in the lane with slight offsets and are determined based on that.

Tests on this level confirm functionality of the integrated system and offer formal verification of implemented solution.

## VI. Conclusion

The key benefit of using a driving simulator in the development of autonomous vehicles is to explore the subjective, as well as the objective, effects of autonomous platform inputs under a range of conditions and circumstances with a lowest cost. Creating these virtual environments using different obstacles, road models, unexpected events etc. enables OEMs to remove the need for expensive prototypes during the development phase. Ultimately this will accelerate the time to market for fully autonomous vehicles and offer safer and futuristic transportation.

This paper emphasized the importance and benefits of early software testing and presented a practical example of simple SIL setup. A setup consisting of Autoware and Carla allowed us to have a complete feedback loop for application and future research testing and better understand the whole development process. Future work will include research on other simulators and platforms to gain wider understanding of real-world needs presented in the automotive industry and autonomous driving problem domain.

## References

[1] "SAE levels of driving automation", https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic [Accessed: September 2019].
[2] S. Behere and M. Torngren, „A functional architecture for autonomous driving", 2015 First International Workshop on Automotive Software Architecture (WASA), Montreal, QC, 2015, pp. 3-10.
[3] Matthaei, Richard & Maurer, Markus, „Autonomous driving – A top-down-approach", Automatisierungstechnik, 63, 2015.
[4] Kalra, Nidhi and Susan M. Paddock, Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability? Santa Monica, CA: RAND Corporation, 2016.
[5] "Waymo simulated kilometres", https://techcrunch.com/2019/07/10/waymo-has-now-driven-10-billion-autonomous-miles-in-simulation/ [Accessed: September 2019].
[6] "Autoware.io", https://github.com/Autoware-AI/autoware.ai [Accessed: July 2002].
[7] "OpenDRIVE", http://www.opendrive.org/ [Accessed: September 2019].
[8] Fang, J., Zhou, D., Yan, F., Zhao, T., Zhang, F., Ma, Y., Wang, L., & Yang, R. (2018). Augmented LiDAR Simulator for Autonomous Driving.
[9] "rFpro", http://www.rfpro.com/ [Accessed: September 2019].
[10] "LGSVL Simulator", https://www.lgsvlsimulator.com/ [Accessed: September 2019].
[11] "AVS", https://avs.auto/ [Accessed: September 2019].
[12] Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. CoRL.
[13] "NVIDIA DRIVE CONSTELLATION", https://www.nvidia.com/en-gb/self-driving-cars/drive-constellation/ [Accessed: September 2019].
[14] "Gazebo simulator", http://gazebosim.org/ [Accessed: April 2020].
[15] Fang, Jin & Yan, Feilong & Zhao, Tongtong & Zhang, Feihu & Zhou, Dingfu & Yang, Ruigang & Ma, Yu & Wang, Liang. (2018). Simulating LIDAR Point Cloud for Autonomous Driving using Real-world Scenes and Traffic Flows.
[16] Sooyong Jeong, Yongsub Kwak and Woo Jin Lee, "Software-in-the-Loop simulation for early-stage testing of AUTOSAR software component," 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), Vienna, 2016, pp. 59-63.
[17] "Robot Operating System", https://www.ros.org/ [Accessed: September 2019].
[18] "ROS bridge for CARLA simulator", https://github.com/carla-simulator/ros-bridge [Accessed: September 2019].
[19] "Unreal Engine 4", https://docs.unrealengine.com/en-US/index.html [Accessed: September 2019].
[20] "Autoware in Carla", https://github.com/carla-simulator/carla-autoware [Accessed: September 2019].
[21] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," in *Computer*, vol. 22, no. 6, pp. 46-57, June 1989.
[22] „Lane finding project", https://github.com/stevanStevic/Advanced-Lane-Lines-Finding [Accessed: April 2020].
[23] S. Stević, M. Krunić, M. Dragojević and N. Kaprocki, "Development and Validation of ADAS Perception Application in ROS Environment Integrated with CARLA Simulator," *2019 27th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2019, pp. 1-4.