2021-01-0106 Published 06 Apr 2021

No Cost Autonomous Vehicle Advancements in CARLA through ROS

Gabriel Prescinotti Vivan, Nick Goberville, Zachary Asher, Nicolas Brown, and Johan Rojas
Western Michigan University

Citation: Prescinotti Vivan, G., Goberville, N., Asher, Z., Brown, N. et al., "No Cost Autonomous Vehicle Advancements in CARLA through ROS," SAE Technical Paper 2021-01-0106, 2021, doi:10.4271/2021-01-0106.

Abstract

Development of autonomous vehicle technology is expensive and perhaps more complicated than initially thought, as evidenced by the recent rollback of anticipated delivery dates from companies such as Tesla, Waymo, GM, and more. One of the most effective techniques to reduce research and development costs and speed up implementation is rigorous analysis through simulation. In this paper, we present multiple autonomous vehicle perception and control strategies that are rigorously investigated in the user friendly, free, and open-source simulation environment, CARLA. Overall, we successfully formulated potential solutions to the autonomous navigation problem and assessed their advantages and disadvantages in simulation at no cost. First, a lane finding method utilizing polynomial fitting and machine learning is proposed. Then, the waypoint navigation strategy is described, along with route planning. Object detection is then implemented using pre-trained convolutional neural networks. A classic PID control strategy and the Stanley Method were investigated for lateral and longitudinal control of the vehicle. Finally, each of these components are simultaneously applied in the simulation environment using the robot operating system (ROS). As a result, we have achieved successful self-driving simulation. The key takeaway is that the perception and control strategies proposed can be easily transitioned towards physical implementation, through the use of ROS. The overall conclusion is that the CARLA simulation environment is a reliable workbench to test innovative solutions that could become technology enablers for the autonomous vehicle industry.

Introduction

Autonomous vehicles have recently become a hot topic for research, and nearly all major car manufacturers have either created entire new departments or partnered with startups whose main goal is to make self-driving cars a reality in the near future. This booming industry segment has fostered the development of disruptive technologies, from innovative neural network architectures to cutting-edge sensor fusion algorithms. On the other hand, the initial excitement around developing new technologies might have contributed to an underestimation of the challenges associated with making self-driving cars safer than human drivers. This is evidenced by the recent rollbacks of delivery dates of fully autonomous vehicles by major Original Equipment Manufacturers (OEMs) [1-3].

In the mid-2010s, the advancements of autonomy levels seen in vehicles, from SAE level 1 to SAE level 3, coupled with positive results from AI developments, led to predictions that level 5 autonomy would be a reality by 2020 [4, 5]. These predictions were followed by announcements from Waymo, Toyota, Honda, and GM that they would be releasing autonomous vehicles by then [6-8]. Tesla had an even more aggressive deadline, by 2018, which was then postponed once proved unsuccessful [9].

There are multiple reasons why the anticipated delivery deadlines were not partially or fully met, ranging from ever increasing safety and reliability requirements and low public acceptance, up to AI technology limitations or low number of miles driven. Common to all of these issues, developing self-driving cars is expensive and time consuming.

Physical data collection is perhaps one of the slowest and most expensive methods to obtain training and testing data to the autonomous algorithms being developed. A clear outlier, Waymo has driven 20 million miles since its creation in 2009, half of which has been driven in the past year [10]. The second place goes to Cruise with 0.5 million miles [11]. However, even when all of the driven miles of all autonomous vehicles are added together, we are still statistically unable to prove that self-driving cars are safer, since human driving produces one fatal accident every 100 million miles [12].

As means to augment testing and validation data, researchers have recently been developing high-fidelity simulators, allowing algorithm designers to obtain billions of driven miles in a much shorter time frame. On the other hand, the platforms needed to run these simulations are often expensive, essentially preventing lower cost solutions to be tested by smaller companies or university research groups.

With the purpose of increasing simulation access to lower budget research initiatives, it is important that open-source software are tested and validated. This is the main goal of this study. We present multiple different solutions to the autonomous navigation problem, and implement them using the CARLA open-source platform [13]. First, we provide a brief literature review of similar papers that either had simulations or the CARLA platform as their object of study. Then, we introduce the software and methodology used in this work. Next, the perception algorithms are presented, namely the object detection and lane finding nodes. The localization strategies are then explained, as well as the two different control strategies that were used. Finally, we conclude by providing the next steps to implement the proposed solutions in a physical vehicle.

Related Work

There have been previous studies that utilized open-source simulation environments to test and validate (T&V) algorithms. References [14, 15] further describe the benefits that simulations can bring to T&V. The former introduces a methodology for validation that uses machine learning and deep neural networks to integrate each level of development of an autonomous system. They also argue that there is much uncertainty around how much simulation is needed to guarantee that a safe autonomous system is deployed. The latter introduces a novel testing platform for self-driving cars with hardware in the loop (HiL) through Gazebo [16], which is similar to our object of study.

Considering the wide array of applications that one could use in their simulations, there are also multiple different agent-based platforms that could be tailored after particular real-world problems. Jing et al. [17] provide a systematic literature survey of different models that were developed to study the performance of autonomous vehicles (AVs), environmental implications or traffic congestion, modal share of AVs versus existing travel modes, among other topics. In our study, we are more interested in the localization and navigation problem, which motivated our selection for a simple yet complete simulation environment, such as CARLA.

CARLA has also been the simulator environment of choice by multiple academic studies. Dworak et al. [18] assessed the capability of this simulator to generate unlimited samples for LiDAR point cloud training data. Upon validation with three different neural network architectures, they confirmed that CARLA is a valid source of artificial data for the automotive industry. Another benefit of this platform is its client-server architecture and the ability to communicate using Robot Operating System [19]. Stević et al. [20] provide a validation study of a perception algorithm using software in the loop (SiL), through CARLA and ROS. Lastly, trajectory prediction and decision modules can also be simulated with CARLA, as shown by Buhet et al. [21]. That study proposes a mid-to-end prediction strategy using a conditional navigation goal, showing that a mix of high- and low-level data improve the performance of deep learning algorithms.

Nonetheless, there exists other open-source, state-of-the-art simulators that are also being used in academic research, such as Gazebo, DeepDrive [22], LGSVL [23], and others. The motivation for using CARLA was a combination of availability of published results, uncomplicated implementation of ROS, vast libraries of urban environments, and good quality performance and visuals.

Methodology

The algorithms for perception, localization, navigation and control were developed independently, and then integrated into a larger system that communicates through ROS, further detailed next. The strategy for making the vehicle autonomous was separated into two main areas: one related to computer vision, collecting, processing, and interpreting environment data; whereas the other relied on collecting GPS data, as well as using the output of the first algorithm to send control messages to the vehicle. The simulation environment is responsible for providing the inputs to the object detection and control scripts, as well as reproducing their outputs.

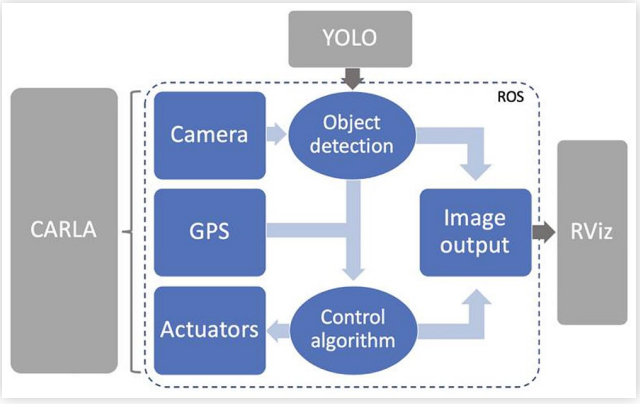
Another input to the object detection script is a pre-trained object detection model named You Only Look Once (YOLO). This model takes a single end to end neural network with a family of convolutional neural networks to detect objects in real time [24, 25]. Figure 1 displays the basic strategy for the entire algorithm.

As seen, in order to integrate each individual software and pass commands from the algorithm to CARLA, a middleware is needed. For that, the team decided to use ROS, which provides a combination of libraries and tools that can be used to simplify the task of creating complex robot behavior.

ROS

An algorithm that is composed of different software and relies on multiple programs for data collection, processing, and subsequent autonomous decision making, must have an

FIGURE 1 Algorithm integration overview



operating system capable of providing interaction between them. In brief, ROS allows this to happen through user defined nodes (i.e. a python script), which can subscribe to different messages, such as sensors (camera, lidar, radar), GPS tracking, or an object detection algorithm. The node then interprets the messages being published through ROS, applies a control strategy, and publishes the messages to the vehicle actuators (acceleration, braking, steering).

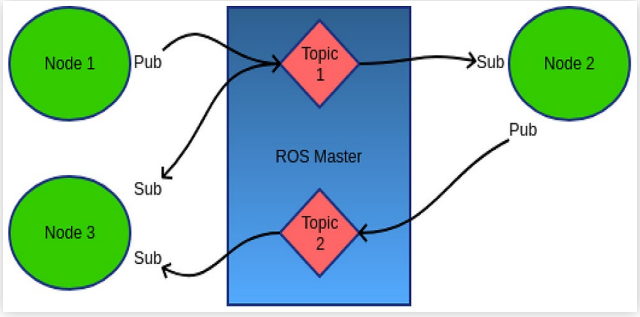
ROS acts as the middleware of the AV stack, meaning it is how commutation from each subsystem is accomplished. In ROS terms, each subsystem is a node within the network. The ROS Master distributes data to and from each of these nodes in the form of a message. These messages must have a specific format according to the type of data that is contained within the message. For example, a message coming from a GPS sensor must have latitude, longitude, and altitude, while a message coming from a camera may just contain an image. In order to distribute these messages, ROS has a technique of naming these messages in the form of a topic. A topic can be published to (giving it data) or subscribed to (receiving the data) by any node, depending on what that node needs. For example, the perception node will be publishing a topic containing a message for a camera image, and the localization node will subscribe to that topic to receive the message and extract the camera image for use with computer vision and machine learning techniques to find the lane lines. A depiction of the flow of ROS topics between nodes can be seen in Figure 2.

The example above shows how nodes publish and subscribe to topics handled by the ROS Master. One key concept shown is that topics must be published to by exactly one node (no more or less), but topics can be subscribed to by as many nodes as possible.

The implementation of ROS is key in the design and architecture of our autonomous vehicle software stack. As seen previously in Figure 1, ROS is responsible for connecting all different topics in a well-organized fashion. Each oval consists of an algorithm written by this team, whereas boxes represent an already existing software (gray boxes) or a built-in topic (blue boxes). As mentioned, topics communicate via publisher or subscriber commands through ROS, present in the library *'rospy'*.

ROS is able to communicate with CARLA by using commands from the library *'carla-ros-bridge'*, while the images obtained from the simulator are converted into arrays with *'CV-bridge'*. This operation is necessary in order to manipulate the camera frames using the computer vision software OpenCV [26]. After the camera frames are preprocessed and converted to *NumPy* arrays, they can be used as inputs to the object detection algorithm.

FIGURE 2 ROS operating scheme



Object Detection

There are many possible ways to achieve real-time object detection, and a good majority of these methods include machine learning [27, 28]. In this study, a pre-trained convolutional neural network created by YOLO was used. The developers of this project argue that their method is much faster than compared to deformable parts models (DPM) or region-based convolutional neural networks (R-CNN) [24], which is one of the reasons why this model was selected.

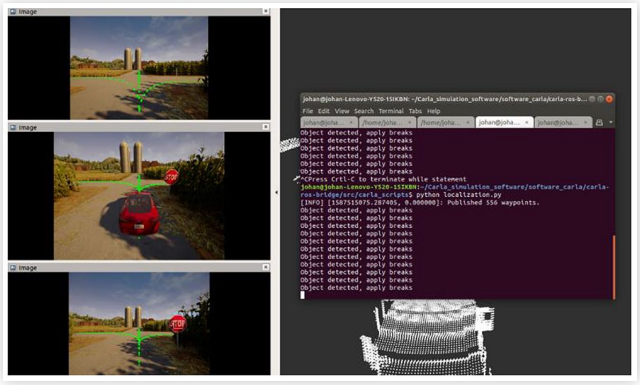
In brief, the inputs to this algorithm are the images from a camera sensor attached to the EgoVehicle, the agent. The frame is then resized according to a user-defined setting, a high definition frame results in slow but more accurate results. The neural network outputs a class ID for the type of object detected, its detection confidence, and a set of coordinates for the rectangles to be drawn around the detected object. The algorithm then matches the class ID to a list of string values for possible detections and writes the detection in an output frame. This frame is published to RViz through ROS-bridge, as seen in Figure 3 and Figure 4, while the class IDs are published to the control algorithm.

Upon implementing and testing the pre-trained model, it was noted that using the entire object library provided by YOLO seemed infeasible. This is because the team did not possess a highly capable graphics processing unit, and therefore feeding a video through the neural network resulted in a highly lagged output. Nonetheless, YOLO also provided a smaller library [24] in order to allow for faster detections,

FIGURE 3 Stop sign detection

FIGURE 4 Car detection

FIGURE 5 EgoVehicle applying brakes once object is detected



which was used instead. Test results showed that while lagging still occurred it occurred less frequently, whereas objects that were most important to an autonomous vehicle were still detected in a timely manner (i.e. stop signs, cars, traffic lights, pedestrians, etc.).

Lastly, once the control algorithm received a class ID signal, a simple logic was applied. In the case a stop sign is detected, the control message is to apply brakes and stop accelerating, as seen in Figure 5. Once the stop sign is not detected anymore, the controller returns to its usual subroutine. This logic raised some questions within the team, such as what would happen in the case where the stop sign is still visible when the vehicle comes to a full stop. Therefore, an extra layer to the logic was implemented, where the vehicle would only look for a stop sign if the vehicle is in motion. In any case, since the object detection algorithm lagged the control algorithm by a few seconds, the stop sign was not visible anymore by the time the vehicle came to a full stop. Fine tuning this operation is considered future work, as the team did not have computational capacity to completely overcome the lagging problem.

Another strategy that the group suggests as future work is to create regions of interest for detecting objects. If more than one camera is utilized, for example, one for front view, and another for side view, the camera on the side could be assigned to solely detect stop signs, while the one in the front detects traffic lights or other obstacles. This could simplify the neural network by reducing the frame size to be analyzed at all times, potentially requiring less computational power.

Lane Detection

The perception system is what provides the autonomous vehicle with information about the surroundings. CARLA offers the option to add LiDAR, radar, camera, and GPS sensors. The system architecture we decided to use consists of a camera and a GPS sensor. This was chosen because it is a challenge to use just a camera as a main sensor, and GPS was needed to give the vehicle information about traffic light locations, necessary turn locations, and intersection locations.

Local localization is achieved in two parts. The first part relies on computer vision to extract data describing the vehicle's local position from a camera feed. After the data is extracted, the second step turns the extracted data into a model that can be used by control strategies to safely navigate the vehicle.

The computer vision step is achieved with a series of image processing operations including thresholding, masking, and morphology. First, we take the input image and apply an adaptive threshold in order to extract parts of the image that are brighter than their surroundings. An adaptive threshold was chosen against an absolute threshold. This was due the former's ability to adapt to changes in lighting, as it looks for brightness relative to an area's surroundings instead of relative to an absolute number. Next, an area of interest is masked to remove data that is irrelevant to lane keeping. After applying the mask, only data that is near the lane in which the vehicle is driving in remains. Areas such as the sky or sides of the road are removed by the mask. At this point in the process, detected lane lines can be patchy or there can be some noise in the image originated from road texture or precipitation. To account for this, open and close morphology operations were used to remove noise and close holes in lane lines. Now the image is ready for data extraction.

After the image preprocessing step, data is extracted from the image to create a model. In order to detect left and right lanes separately, the image is split into two halves that correspond to the left and right lanes. Instead of splitting the image down the geometric center of the image, this operation is done with respect to the center of the image after perspective transformation, further described next. Splitting the image this way allows for different perspectives

FIGURE 6 Preprocessing operation

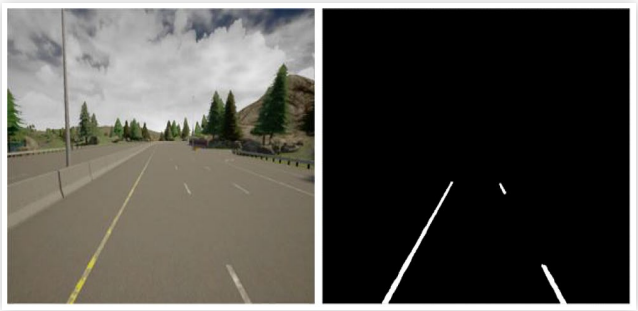
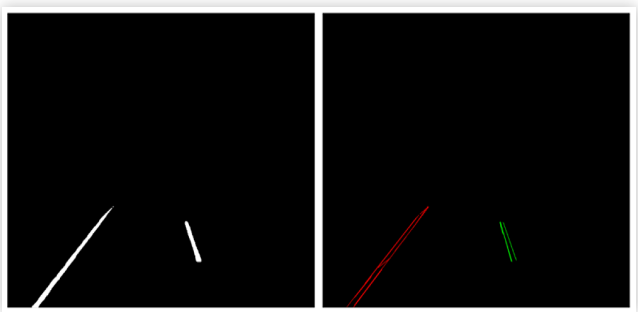


FIGURE 7 Canny edge and Hough transform operations



adjustments that might come from multiple camera heights, angles, and tilts.

On each half of the split image, Canny edge detection is used to turn the relatively thick lane lines into thin outlines. After edge detection, a Hough line transform operation is performed to turn the edges into positional data in the form of a set of lines. The Hough line transform looks for straight lines in the image and extracts their end points. The Hough line transform is then tuned to prioritize many short lines instead of few long lines. This is done for two reasons: to get as many data points as possible and to account for curves in the road. Curves can be detected as a series of short lines connected.

Some control methods, such as the Stanley Method described next, requires a model that orients the center of a lane from a top down perspective. As it stands now, the data is from the perspective of the camera attached to the EgoVehicle.

OpenCV provides perspective transform functionality. As input, it takes two sets of four points. The first set corresponds to the positions of four points in pre-transformed space, and the second set corresponds to where those points will end up in transformed space. A set of points that formed a rectangle was set on the ground and transformed to a rectangle corresponding to a top down view. This resulted in a linear transformation matrix that was applied to all data.

Equipped with a set of coordinates for the lanes, it is now possible to perform curve fitting operations. But before that, the data was augmented by generating linearly interpolated points between the data points of existing lines. This significantly increased the amount of points that the curve fitting process is based on, and it reduced the amount of noise in the fitted curves.

FIGURE 8 Perspective transform operation

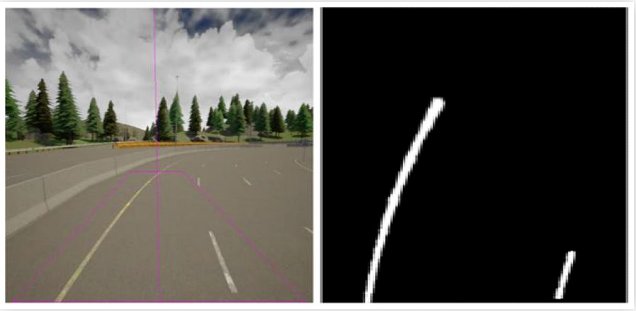
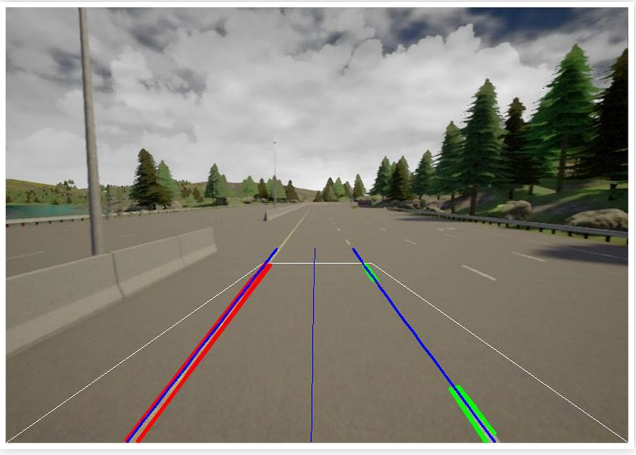


FIGURE 9 Lane detection results



NumPy was then used to fit polynomial models to each set of data describing each line (left and right). A second-degree polynomial was used since the underlying curve of the road can only turn in one direction at a time. The range of the camera data used is not long enough to see any "S" shaped curves. We also explored using a cubic spline to fit a curve to the data, but the cubic spline was too flexible and resulted in large amounts of overfitting.

Once two independent models are obtained for the left and right lane lines, they are combined into a single model that describes the center of the lane. To do this, we simply average the two models.

Waypoint Navigation

Using only cameras for lane detection is beneficial for giving your vehicle a comprehensive understanding of where it is with respect to the lane lines. This is very powerful information since one can completely control their vehicle on a highway with this. But in order to take left and right turns, plan paths at intersections, or drive on a street without lane lines, more information is needed from other sensors to localize the vehicle. This can be done with LiDAR, GPS, or other sophisticated computer vision algorithms that are under development due to high computational expense. Keeping low costs in mind, this group decided to use a GPS sensor for global localization.

Global localization allows the vehicle to know where it is with respect to a large map. This provides important information needed for safe operations since the camera algorithms are not able to determine if the EgoVehicle is at an intersection or not. Using the GPS data, we were able to incorporate the ability for the vehicle to operate at intersections and take left turns. The process used to achieve this is as follows:

- Conduct mapping of the desired route to follow. Since we are just using a GPS sensor for mapping and global localization, we only needed to collect GPS data which included latitude and longitude. This was done by driving the vehicle using CARLA's autopilot mode and collecting *rosbags*.
- Using the collected *rosbags* and a custom python code, we converted the *rosbag* data containing GPS information into a *.csv* file that is used as a map.
- In order to use this data in operations, we needed a fast way to search through the map given a specific latitude/longitude. To do this, we fit a 2D interpolation model that fits a spline to the data. The input to this model is the latitude/longitude and the output is the row of the *.csv* file that the latitude/longitude is located.
- After receiving the row in which the latitude/longitude is stored, the characteristics of the map at that location are

checked by matching the other data stored in that row of the *.csv* file (i.e. flags for traffic lights, or traffic signs)

- Lastly, we fit a curve to the next 25 map points to define a path for the vehicle to follow.

Control Strategies

A control algorithm was developed to safely control the vehicle through the CARLA environment. The approach taken was to develop an algorithm that plans the route and controls the vehicle using a PI controller. The waypoints scattered across CARLA were set as targets for the PI controller. The waypoints are located in the middle of each lane and separated by approximately 1 meter. CARLA possesses built-in agents and commands that allow the user to access the location and rotation of each waypoint. The built-in agents are scripts developed to plan the route from a starting point to a given destination. The output of this planned path is a list of all the waypoints from the vehicle's current location to its final destination. Figure 10 is a representation of a section of the route that the vehicle has to follow, where the waypoints are shown in green. The white dots are LiDAR readings not currently used by the control algorithm.

The list of waypoints was fed into the lateral and longitudinal controllers one by one until the vehicle reached the final waypoint. The waypoints need to be introduced individually, because a *for loop* was used to iterate throughout the entire route. However, this is done faster than the response of the messages being published to the vehicle. In other words, the vehicle would want to target the second waypoint of the route during the controller's second iteration, even though it would not have reached the first waypoint yet. Hence, a nested "while loop" was used to compute the Euclidean distance between the vehicle's location and the next waypoint. If the distance is less than a user specified threshold, the vehicle is close enough to the targeted waypoint, and ready to target the next waypoint. This process is displayed in Figure 11.

The proportional and integral gains of the controller were manually tuned, controlling speed and steering torque. Since there was no hardware in the loop, optimization of the controller or the introduction of a derivative gain were not seen as priorities, since the physical model would change as the algorithm is transitioned to physical implementation.

FIGURE 10 Route composed of waypoints

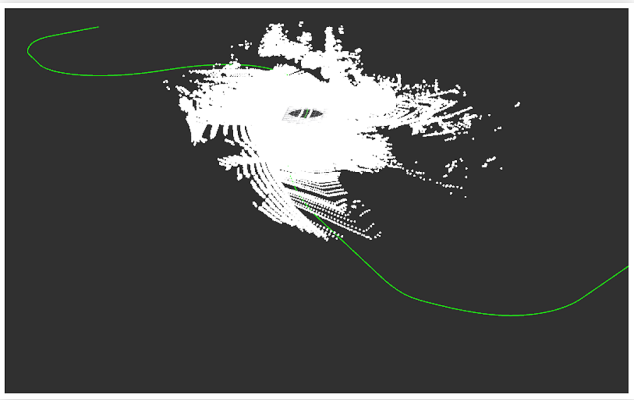
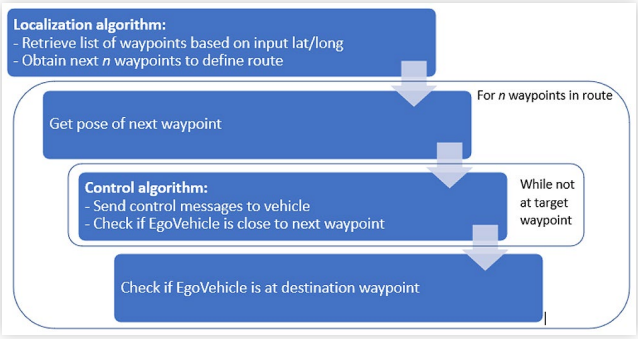


FIGURE 11 Waypoint navigation flowchart



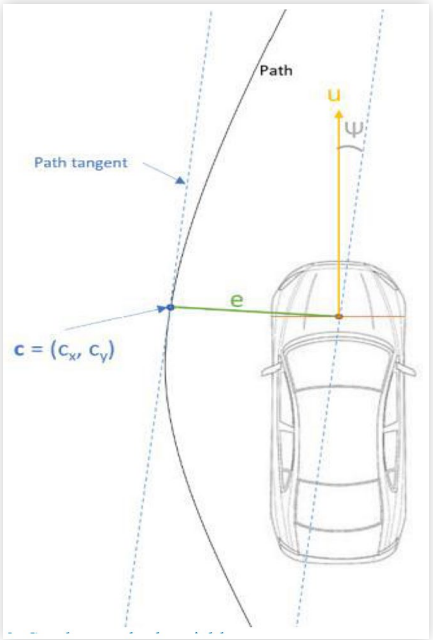
Stanley Method

The next step is to control the vehicle with information from the local and global localizations. While PID controllers offer simplified implementation, the Stanley method is a good alternative control solution with manageable implementation and good accuracy.

The Stanley method, a closed loop control algorithm, was developed by Stanford Autonomous Vehicle group while developing Stanley, the winning vehicle of the DARPA Grand Challenge. This control strategy is centralized about the middle of the front axle of the vehicle, and has three inputs: orientation to the nearest path point, Ψ; vehicle speed, u; and lateral error to the nearest path point, x. The three inputs are shown in Figure 12.

For this method to be used, a path must be predetermined. Using the lane line detection, the lanes can be averaged to give

FIGURE 12 Stanley method variables



a central path to be followed. From this central path, the nearest point to the centralization point is found. The lateral error is simply the distance between the two points. The orientation between these two points requires the slope of the path at the closest point and the current heading angle of the vehicle. The vehicle speed is pulled directly from the vehicle's sensors. With these three inputs, Eq. (1) is used to calculate a steering angle,

$$\delta(t) = \psi(t) + \left(\frac{kx(t)}{u(t)} \right) \quad (1)$$

where k is a gain parameter and adjustable. Note that in the absence of lateral error, x(t), the steering angle is set by the orientation difference, Ψ.

Physical Implementation

The end goal of any autonomous vehicle algorithm is physical implementation. In other words, it needs to work in a real car. Therefore, we have laid out what the next steps would be in order to have the proposed algorithms working in the real world.

Considering that ROS-bridge allows for many different systems to communicate, transitioning from a simulation environment to a real environment would not be more challenging than developing the algorithm in the first place. As it was done for the simulation, it is necessary to connect the sensors and use function *'rostopic info'* in order to understand the syntax of the messages being sent. Once this is done, ROS-bridge can convert the messages into a proper input format for the algorithm. This is applicable to both the camera sensors and GPS navigation.

Now, a major difference between the simulation environment and the real world is setting the origin and destination. In the simulation this is done by inputting a set of cartesian coordinates, that map onto the simulation world, which are then converted to location and orientation coordinates in CARLA. In the real world, we would have to estipulate a destination and rely on some GPS equipment to compute a list of waypoints to follow. Since CARLA operates with transform coordinates (latitude, longitude, elevation, pitch, roll, yaw), we believe that the agents present in the current controller would still work for waypoints generated by an actual GPS. Therefore, the current algorithm would actually have to do less than what it does in the simulation, as it would not need to compute the route anymore.

Once the control algorithm is working in the same manner as in the simulation, we are able to send accelerate, brake, and steering commands to the actual vehicle. In the simulation this was done by using *'rostopic info'* again to understand the correct message format. In the actual vehicle we would have to connect to an OBD-II port through a drive-by-wire control, such as PolySync's DriveKit, understand the message format, and modify the code to output what is desired.

It is likely that transitioning from the simulation to a physical vehicle would not require many changes to the algorithm. However, a considerable amount of time would have to be spent on converting the message types and making sure that the vehicle interprets these messages in the correct way. In addition, the PID controller would need to be fine-tuned, according to the car's response to the feedback received.

Conclusion

An autonomous vehicle algorithm was successfully developed. It utilized and combined different features, such as data collection from multiple sensors, information processing through computer vision and machine learning, and autonomous control via waypoint navigation and a PID feedback controller. The vehicle is also able to stop for obstacles, such as stop signs, other cars and pedestrians. Overall, it can achieve high velocities without losing control or producing excessive jerky movements. Based on this, we believe that this algorithm would be a viable solution to trial in a physical vehicle.

The EgoVehicle was quite successful in the simulation environment, and physical implementation was conducted by the Energy Efficient Autonomous Vehicle (EEAV) Lab at Western Michigan University (WMU) for the Indy Autonomous Challenge (IAC). Although this was also a successful experiment, there is considerably much future work needed for performance evaluation and benchmarking against other existing solutions. The key takeaway of this research paper is that open-source simulation platforms are reliable enough in order to test low-cost, simple, and user-friendly solutions from research groups that do not have access to expensive, state-of-the-art hardware or software.

Contact Information

Gabriel Prescinotti Vivan,
gabrielprescin.vivan@wmich.edu

References

1. Korosec, K., "Ford Postpones Autonomous Vehicle Service Until 2022," April 2020, https://techcrunch.com/2020/04/28/ford-postpones-autonomous-vehicle-service-until-2022
2. Ramey, J., "Here's Why Our Gleaming Self-Driving Future Has Been Delayed Indefinitely," June 2020, https://www.autoweek.com/news/technology/a32782600/why-level-5-autonomous-driving-has-not-happened/.
3. White, J., "GM Cruise to Delay Commercial Launch of Self-Driving Cars to Beyond 2019," July 2019, https://www.reuters.com/article/us-gm-cruise/gm-cruise-to-delay-commercial-launch-of-self-driving-cars-to-beyond-2019
4. Adams, T., "Self-Driving Cars: From 2020 You will Become a Permanent Backseat Driver," September 2015, https://www.theguardian.com/technology/2015/sep/13/self-driving-cars-bmw-google-2020-driving.
5. Business Insider Intelligence, "10 Million Self-Driving Cars will be on the Road by 2020," June 2016, https://www.businessinsider.com/report-10-million-self-driving-cars-will-be-on-the-road-by-2020-2015-5-6.

6. Heaps, R., "Self-Driving Cars: Honda Sets 2020 as Target for Highly Automated Freeway Driving," June 2017, https://www.autotrader.com/car-shopping/self-driving-cars-honda-sets-2020-as-target-for-highly-automated-freeway-driving-266836.
7. Kubota, Y., "Toyota Aims to Make Self-Driving Cars by 2020," October 2015, https://www.wsj.com/articles/toyota-aims-to-make-self-driving-cars-by-2020-1444136396.
8. Madrigal, A., "The Most Important Self-Driving Car Announcement Yet," March 2018, https://www.theatlantic.com/technology/archive/2018/03/th e-most-important-self-driving-car-announcement-yet/556712/.
9. Hawkins, A.J., "Here Are Elon Musk's Wildest Predictions about Tesla's Self-Driving Cars," April 2019, https://www.theverge.com/2019/4/22/18510828/tesla-elon-musk-autonomy-day-investor-comments-self-driving-cars-predictions.
10. Reuters, "Waymo Self-Driving Vehicles Cover 20 million Miles on Public Roads," January 2020, https://www.reuters.com/article/us-autonomous-waymo/waymo-self-driving-vehicles-cover-20-million-miles-on-public-roads.
11. Piper, K., "It's 2020. Where Are Our Self-Driving Cars?," February 2020, https://www.reuters.com/article/us-autonomous-waymo/waymo-self-driving-vehicles-cover-20-million-miles-on-public-roads.
12. U.S. Department of Transportation, "Fatality Analysis Reporting System (FARS)," Dec. 2019, https://www.iihs.org/topics/fatality-statistics/detail/state-by-state#yearly-snapshot.
13. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., "CARLA: An Open Urban Driving Simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 1-16, 2017.
14. Vishnukumar, H.J., Butting, B., Muller, C., and Sax, E., "Machine Learning and Deep Neural Network — Artificial Intelligence Core for Lab and Real-World Test and Validation for ADAS and Autonomous Vehicles: AI for Efficient and Quality Test and Validation," in *2017 Intelligent Systems Conference (IntelliSys)*, London, 2017, pp. 714-721, doi:10.1109/IntelliSys.2017.8324372.
15. Chen, S., Chen, Y., Zhang, S., and Zheng, N., "A Novel Integrated Simulation and Testing Platform for Self-Driving Cars With Hardware in the Loop," *IEEE Transactions on Intelligent Vehicles* 4(3):425-436, Sept. 2019, doi:10.1109/TIV.2019.2919470.
16. Koenig, N. and Howard, A., "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Sendai, (3):2149-2154, 2004, doi:10.1109/IROS.2004.1389727.
17. Jing, P., Hu, H., Zhan, F., and Shi, Y., "Agent-Based Simulation of Autonomous Vehicles: A Systematic Literature Review," *IEEE Access* 5:79089-79103, 2020, doi:10.1109/ACCESS.2020.2990295.
18. Dworak, D., Ciepiela, F., Derbisz, J., Izzat, I., et al., "Performance of LiDAR Object Detection Deep Learning Architectures Based on Artificially Generated Point Cloud Data from CARLA Simulator," in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Międzyzdroje, Poland, 600-605, 2019, doi:10.1109/MMAR.2019.8864642.
19. Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System," 2018, https://www.ros.org
20. Stević, S., Krunić, M., Dragojević, M., and Kaprocki, N., "Development and Validation of ADAS Perception Application in ROS Environment Integrated with CARLA Simulator," in *2019 27th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 1-4, 2019, doi:10.1109/TELFOR48224.2019.8971063.
21. Buhet, T., Wirbel, E., and Perrotton, X., "Conditional Vehicle Trajectories Prediction in CARLA Urban Environment," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Seoul, Korea (South), 2310-2319, 2019, doi:10.1109/ICCVW.2019.00284.
22. Quiter, C. and Ernst, M., Deepdrive 2.0. https://deepdrive.io/, Oct. 2020.
23. Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., et al., "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving," 2005, https://www.lgsvlsimulator.com/
24. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788, 2016.
25. Redmon, J. and Angelova, A., "Real-Time Grasp Detection Using Convolutional Neural Networks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seatlle, WA, 1316-1322, 2015, doi:10.1109/ICRA.2015.7139361
26. Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
27. Ramik, D.M., Sabourin, C., Moreno, R. et al., "A Machine Learning Based Intelligent Vision System for Autonomous Object Detection and Recognition," *Appl Intell* 40:358-375, 2014, doi:10.1007/s10489-013-0461-5.
28. Szegedy, C., Toshev, A., and Erhan, D., *Advances in Neural Information Processing Systems 26* (Curran Associates, Inc.), 2553-2525.

ISSN 0148-7191