Abstract Simulation Scenario Generation for Autonomous Vehicle Verification

Christopher Medrano-Berumen and Mustafa İlhan Akbaş
Department of Computer Science
Florida Polytechnic University
Lakeland, Florida, USA
{cmedranoberumen2844,makbas}@floridapoly.edu

***Abstract*—Autonomous vehicle technology has the potential to have a significant impact on the transportation system and urban life. However, the autonomous vehicles must be proven to be at least as safe as human-driven vehicles before they are accepted as a new mode of transportation. Current methods of autonomous vehicle verification such as shadow driving or annotated images based testing are costly, slow and resource intensive. Hence, modeling and simulation is an indispensable asset to achieve verification goals for autonomous vehicles.**

In this paper, we propose an abstract simulation scenario generation framework for autonomous vehicle verification. The scenes and the related assertions are defined by a matrix-based semantic language and translated into test scenarios in simulation. The framework allows the design of all possible road topologies and the validation of generated scenarios. The scenarios generated in the framework form a ground truth for possible extended tests of rare conditions in other platforms.

***Index Terms*—Autonomous Vehicles, Verification, Simulation, Scenario Generation**

I. Introduction

Assisted driving systems and autonomous vehicle (AV) technology progressively shifts the driving responsibilities from human driver to the driving systems integrated into the vehicle. Considering the current extremely low and inefficient utilization of vehicles [1] and the potential benefits in terms of safety, logistics and civic life, AV technology has the potential to have a disruptive effect on multiple sectors.

The Society of Automotive Engineers defines six levels of autonomy for vehicles from "Level 0, No Automation" to "Level 5, Full automation" [2]. The automation levels of "Level 1, Driver Assistance" and "Level 2, Partial Automation" become increasingly available in the market. These levels still require a human driver in control all the time and even then there are various accidents reported while these systems are in use, which damages the reputation of AV technology and the corresponding investment [3]. The following question has to be answered for the pervasive use of AVs in public: "How can we trust autonomous vehicles?". Hence, verification is one of the key issues in front of the AV technology.

Currently, the most common method of autonomous vehicle verification is real-world testing, where the vehicles are tested on either streets or regulated test tracks. This is impractical due to the number of hours and resources necessary [4]. There are also other methods of verification such as hardware in the loop (HiL) or annotated image based testing. However, there is no standard system for AV verification and it continues to be an unsettled challenge [5].

978-1-7281-0137-8/19/$31.00 © 2019 IEEE

In this paper, we propose an abstract scenario generation system to address the critical issues of the AV verification through modeling and simulation. For the verification of autonomous vehicles, we are inspired by the successful testing methodologies used in hardware verification and our main focus is applying these methodologies for the verification of AV decision taking process through modeling and simulation. It's important to note that we focus on generating scenarios to test the decision making of AVs and our approach aims to evaluate the decisions taken by the AV under test related to the scenario only. Therefore, environmental conditions are not included and our system produces bare-bones scenarios with only road segments and actors.

In our approach, we first create a method to use a semantic language for describing driving scenarios that can take random values as inputs and then convert them into a logical driving scenario in simulation. For instance, a road segment can be described by the function of the line that it follows as well as the width at each point. Then we use geometric primitives to generate roadways with every possible curve and number of lanes in the road topology. The goal in defining different road pieces is to more easily constrain the generated scenario to realistic road networks and situations. In our implementation examples, the description of these lines as well as the four-way intersections have been demonstrated. By exploring all inputs to these pieces and to this language in general, the behavior of an AV can be tested in any situation. Our approach also verifies itself to eliminate illegal simulation scenarios.

There are four main contributions of this work:

- We develop a method in simulation to use a semantic language for the definition of scenes when generating scenarios.
- We propose a method to compose roadways in simulation with every possible combination of lines or curves, which is critical for enabling the modeling of real-life roads.
- We define a method to stitch road segments of different number of lanes to each other without errors in the simulation.

- We propose a self verification method for each appended road segment within the proposed approach, which is critical for simulation efficiency.

The remainder of this paper is organized as follows. Related work is given in Section II. The system model and a detailed description of the approach are given in Section III. We present the implementation details and simulation examples in Section IV and finally conclude in Section V.

II. Related Work

There are various approaches from industry and academia for the testing and verification of autonomous vehicles, using multifaceted approaches utilizing the real and the simulated world [6]. In real world testing, fleets of autonomous vehicles are driven on streets with someone behind the wheel [7]. This type of testing is known as shadow driving. Using test tracks is also common for real world testing, as this allows companies to test specific, extreme scenarios. Hardware-in-the-loop (HiL) testing is another option that allows connecting autonomous vehicle's brain into a simulation test and testing interaction with specific hardware components simultaneously [8]. While this form of testing functions in real time, Software-in-the-Loop (SiL) allows the test to run in simulation time and it has been employed in the unmanned aerial vehicle space [9]. Whether the brain is connected to the simulation test as hardware or software, the system is mostly similar except the sensors. The scene is generated within the simulation software and sent from the perspective of where the vehicle under test would be to either simulated sensors for SiL or real/simulated sensors for HiL. This view is then sent to the brain, either in the real world or simulated world based on what type of testing is being done. From there, once a decision is made, it is sent through any physics component of the vehicle, such as the tire or axis properties, which can also be done in the real or simulated world. After going through that, the vehicle has now made a decision and made itself move the way that it intended to, so the scene is updated with the new information and the next perception-decision step is played out.

Some of the testing approaches focus on using simulation to verify that newly learned maneuvers (e.g. u-turns, merging) are tested until they can be performed at a satisfactory rate. Others use simulations based on scenarios that their vehicles in the real world encountered [6]. Each important scenario from real life can then be fuzzed into generating many more scenarios based on the original to strengthen the coverage of that test. There are also several recent initiatives aiming for the standardization of AV verification by integrating different techniques. Intelligent Testing Framework [10] and PEGASUS [11] are two important examples of such approaches.

The majority of the current approaches target testing of the full vehicle stack, from scene perception and understanding to making a decision for action in that scene. In our approach, we focus mainly on the decision step. In other words, our approach is designed to test the decision making of the AV under test without any problems in any other conditions. This focus point played into deciding the requirements of the simulation platform as well. The framework requires bare bones modeling of the physical world without details on environmental conditions. For that reason, MATLAB is chosen as the simulation platform. The MATLAB ADAS toolbox represents actors in the scenario simply as boxes [12].

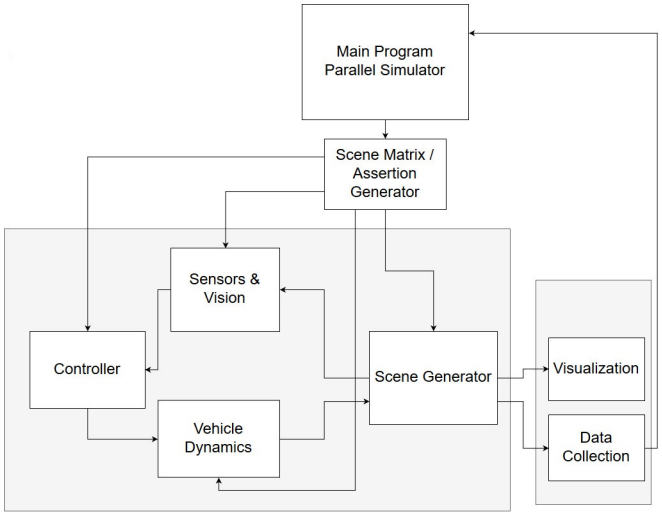

Fig. 1. Simulation Framework Architecture.

III. Scenario Generation Framework

The Figure 1 presents a breakdown of the components of our simulation framework. The middle grayed area is the brain of the AV running through the scenario created from the input. The modules at the top define how the user interfaces with the framework to generate the scenarios. The simulation framework begins in the main program which has certain tuning and input options. The scenarios are then randomly generated according to those inputs and fed into the actual model. Here, the scenario is generated, the ego vehicle (vehicle under test) is placed into it, and each step is run with the ego vehicle responding to it through the entire cycle of sensing, taking a decision, and acting it.

The simulation data is collected during and after the run. The data points include safety and legality of each decision and the scenario's definition. Since the scenario was generated by a certain input, it can be recreated whenever it is required.

A. Semantic Language

The model must be capable of creating scenarios to reflect the set of all possible situations. We developed a semantic language for breaking down the factors that define a scenario including roads, actors, and traffic logic. Therefore, the first phase in our approach is the creation of this semantic language.

Our first method for the semantic language uses a string structure and associated rules as follows:

- The road generation starts with a string that is matched with regular expressions.
- The road types are in uppercase letters.
- The parameters are in lowercase letters.
- The number values are placed after parameters.

Fig. 2. Sample simulation scene.

- Parameters and their number values are put before Road Types.

According to these rules, the input is taken from the command line and the formal grammar is parsed to generate tokens. The tokens also have context sensitive parameters included into each type of token. The developed tokens and their attributes are given in Table I.

TABLE I
ROAD GENERATION TOKENS.

Token	Attributes
S - Straight Road	l - Length of the Road
B - Bend Road	r - radius, d - degrees (length of the rotation)
C - Curved Road (Bezier)	x - Delta X y - Delta Y n - Entry Tension t - Exit Tension r - Relative Heading
X - X Crossing	r - Outer Radius f - Filet Radius

After all tokens are generated, they are used to generate an appropriate xml segment for the input file of the simulation. There is a unique id number for each road segment, which is used with a global id variable of the scenario that is incremented with each piece. There is also a global array of points that determines the location of the next road piece and auto-updates on the xml generation.

Figure 2 shows a simulation scene created in PreScan [13] and MATLAB Simulink using the example string, '$l05d0A115A180d1A132BCCX150d0A110A1405$'. The PreScan is one of the most comprehensive simulation platforms to simulate AVs and all sensor configurations on AVs. However, these details are unnecessary at the logical scenario generation phase. MATLAB ADAS Toolbox reduces objects to 3-dimensional boxes and roads to the line they follow along with some other minor details [12]. This scenario generation methodology makes it a suitable tool for our approach.

After the initial string-based language implementation, we decided to include not only input patterns but also the assertions of the system in MATLAB. Therefore, the next iteration of the implementation includes a matrix based system to generalize scenario characteristics and create an efficient system of labeling and sorting. The numerical matrix is read as input where each row is a different assertion describing a single road piece or actor that can then be parsed to generate the scenario. To do this, road network and actors are reduced to their most basic elements in terms of Newtonian physics such as center of mass and dimensions. These elements are then parameterized according to real life conditions. It's important to note that the model contains no environmental factors.

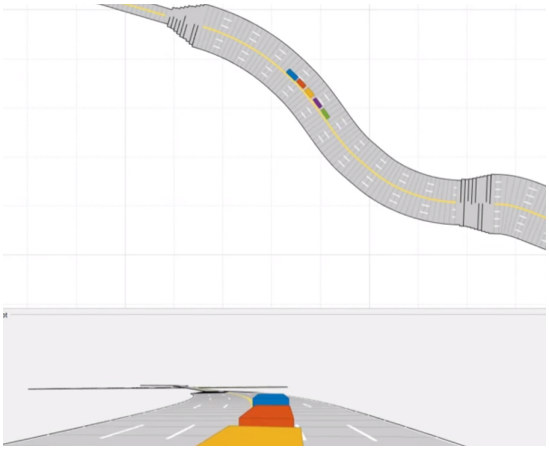

Fig. 3. Example scenario of an ego vehicle leading a platoon

An example scenario created in our framework is shown in Fig. 3. In this scenario, the ego vehicle under test leads a platoon. This example demonstrates the potential capabilities of the implementation methodology for practical testing purposes such as the combination of road segments with variable number of lanes, multiple actors and lane markers.

B. Road Topology Generation

The placement of roads in the simulation requires the road centers, the width of the road at each of those centers, the banking angles, and the details of the lanes. Then, generating the roads is a matter of taking an input and converting it into a series of points that the road follows within the context of the scene. The first value in the input defines what type of road piece will be created. By limiting the roads to specific road pieces, the generated scenario can be constrained to realistic road networks and situations. As each road piece is generated, it is stitched to the previous piece by rotating it so that the tangent line along the first point is lined up with the tangent line of the last point of the previous piece and shifting it to that coordinate in the driving scenario. Between two consecutive pieces, there is also an intermediary piece to smooth transitions between two pieces with different number of lanes.

1) Parameters: Once constrained to the different pieces that represent the most common roads (e.g. Straight road, Intersection, etc.), the details that can be generalized to most to all of the pieces have to be defined as parameters. There are certain parameters that can only apply to a single piece. Therefore to save memory, some of those are given new meaning within the context of different pieces. Current parameters in the model are given in Table II. As more pieces are defined and implemented,

the parameters will also be expanded to describe all of the new details that must be taken into account.

TABLE II
ROAD AND ACTOR PARAMETERS.

Road Parameters	Actor Parameteres
Road Piece Type	Actor Type
Road Length	Actor Sub-Type
No. Lanes	Path Type
Bidirectional	Moving Speed
Mid-Lane	Dimensions
Speed Limit	Start Location
Intersection Pattern	Forward
Curvature1	Offset
Curvature2	

2) Roads and Geometric Primitives: The first type of road piece is intended to be the simplest toad and serves as a building block for more complex roads. It is referred to as the 'multilane road' and is made up of variable lanes going one or both directions in a line made up of geometric primitives. These geometric primitives, as shown in Fig. 4, include the line, the arc, and the clothoid curve which changes curvature at a constant rate.

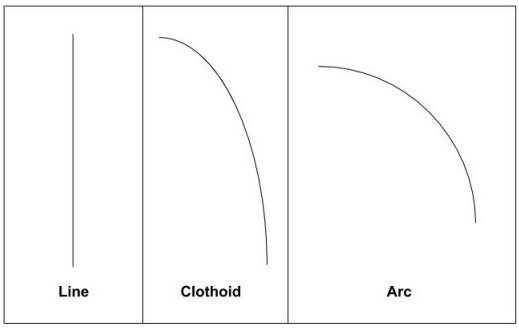

Fig. 4. Geometric primitives that compose roadways

All common forms of the single road can be created through a permutation of the primitives in a set of three. To select what primitives are used in generating the road, two curvatures from the input are used, representing starting and ending curvature. If one of the curvatures is zero, it will go from a straight line to a clothoid curve which transitions to an arc with the non-zero curvature. If they are both non-zero curvatures, the line will randomly become either a clothoid-arc-clothoid or an arc-clothoid-arc shape. For the clothoid-arc-clothoid shape, the curvature begins at zero, transitions to the starting curvature, remains there for the arc, then transitions to the ending curvature in the last clothoid. For the arc-clothoid-arc shape, the line starts at an arc with the starting curvature, transitions to the ending curvature using a clothoid curve, and remains at the ending curvature for the final arc. An example of the road segment connection is shown in Fig. 5.

To get the primitives to seamlessly connect with each other, the tangent line to the first point is made to line up with the tangent line of the last primitive's last point. For a straight line, the facing direction remains the same as it was at the beginning. However, for the arc and clothoid curve, the new facing direction must be calculated. Calculating the arc's new facing direction is simply a matter of multiplying the length of the arc with the curvature and adding or subtracting it from the previous facing direction based on which way it is turning.

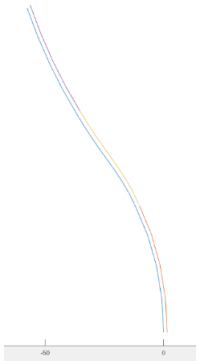

Fig. 5. Clothoids and Arcs being connected

The Clothoid: Clothoids allow for a smooth transition from one curvature to another and are what real roads utilize for easy turning. The points of a clothoid are calculated using fresnel integrals:

$$fresnelc(L) = \int_0^L cos(s^2)ds$$

$$fresnels(L) = \int_0^L sin(s^2)ds$$

$$x = a * fresnels(\frac{s_c}{a})$$

$$y = a * fresnelc(\frac{s_c}{a})$$

where a serves as a normalizing factor, L represents the length from the beginning to that point, and s_c represents the distance along the curve.

For the clothoid, the change in the facing direction must be calculated using the parameters calculated when forming the points using

$$\theta = \frac{L}{2R} \tag{1}$$

where R represents the final curvature and θ is the change in curvature. This provides an unreliable value that does not accurately represent the change. Attributed to the software's level of precision with the fresnel integral (used to calculate the clothoid), we decided to use the arctangent of the last two points to calculate the new facing direction. Because the number of points we chose to use to define this curve puts them extremely close together, this proved to be sufficient for connecting the pieces.

3) 4-Way Intersection: An important example for the road segments in the model is the 4-way intersection as given in Figure 6. We define each of the roads connecting to the intersection as a multilane road with variable lanes and assume these roads can be one-way in either direction or both ways.

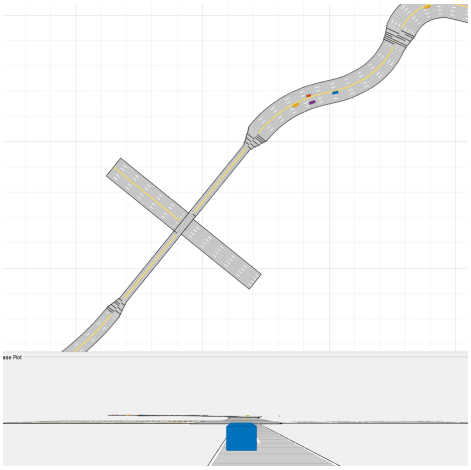

Fig. 6. Example of Generated Four Way Intersection

a) Placement of the roads: The positioning of each road is based on a center rectangle calculated to place all of the roads as close to the center as possible. Each of the four roads can have a different number of lanes and can also be bidirectional or one-way which means that they can all also have different widths. To calculate the rectangle, the largest width between the top and bottom roads and the largest width between the left and right roads are used to determine the size. The center of the top road is lined up with the center of the bottom road and likewise for the left and right ones. To get the roads to their coordinates based on this system, they are shifted from the end of the first road (the center of the bottom road along the rectangle) based on the mathematical relationships between them.

b) Paths of the lanes: Since the scenario moves linearly from piece to piece, only the lanes of the bottom piece need to have the options defined for where the vehicles can go. The other roads have a limited version of this to get potential paths for other actors. If turning left is an option, starting from the left up to two lanes are set as 'Left Turn Only' lanes. The amount set as 'Left Only' is based on how many lanes would be available for the other potential directions as well as how many lanes are available going left. If there are still more lanes left, they are set as 'Forward Only' lanes up to however many lanes are available going forward if any at all. The remaining lanes are set as right turn only lanes up to the number of lanes available going right given that right is an option. If it is an option and there are no lanes left, the rightmost lane is given the extra option of going right. If there are extra lanes available, they are given the same behavior as forward only lanes that merge on the other side. In the case where there are not enough lanes left, such as one lane with each direction being a possible option, the lane is defined as being neutral and allowing any of those directions.

C. Extended Road Properties

When generating the roads, there are extra values from the input that are stored in the road object such as its condition and the speed limit along with it. The speed limit is reflected in the scenario is when the actors' paths are being calculated.

Lane markers are also set during the road calculation. They are passed in as an array with each of the lines' properties. For the center line in two-way roads, it is randomly defined as a double yellow line or as a dashed yellow line. It is also possible to have a turn lane in the center which is always surrounded by solid-dashed yellow lines.

D. Actor Generation

Besides the ego vehicle, actors in the scenario are generated randomly and are also comprised of different types. This includes vehicles which can be cars, trucks, pedestrians, and so on. The different behaviors that they may take in the scenario are also programmed in. These include crossing the road for pedestrians and swerving outside of the lane for vehicles.

The vehicles are assigned lane paths that were calculated as the roads were generated going in either direction (going from the first road to the last or vice versa) and starting at a random point along the network. By this way, the entire road can have actors throughout, rather than just following the ego vehicle.

E. Test Scenario Generation

For the generation of test scenarios, a pseudo-random input generation is used. Integrating this method into the semantic language allows the determination of a level of confidence and coverage analysis. The repeatability of the tests and the coverage analysis are important since they provide a progressive testing structure.

IV. Scenario Validation

Constraints or checks at each new piece are performed to ensure that the generated scenario is constrained to legal test cases only. Checking the scenario for validity as it is being generated prevents illegal scenarios or scenarios that break logical constraints from making it to the test run phase. Considering the extensive number of test cases an ego vehicle would normally be tested against, this is critical to save time in simulations.

The first check to see whether the new piece will create a conflict calculates a rectangle around the piece and compares it with the rectangles calculated for the previous pieces to see if they intersect. These rectangles are calculated to wrap around the minimum and maximum values of the corners' coordinates, making them simple yet effective. If the piece about to be placed intersects with a previously placed piece, the placement is aborted and the next piece in the matrix is attempted.

Then, a rectangle in front of the to-be-placed piece is also checked against the placed pieces to ensure no deadlocks occur, or that pieces can continue to be placed into the scene as shown in Fig. 7 so that a road matrix can place as many of it's assertions into the scene as possible.

One of the ways that we see pieces failing the initial check is in response to the length of the pieces being increased.

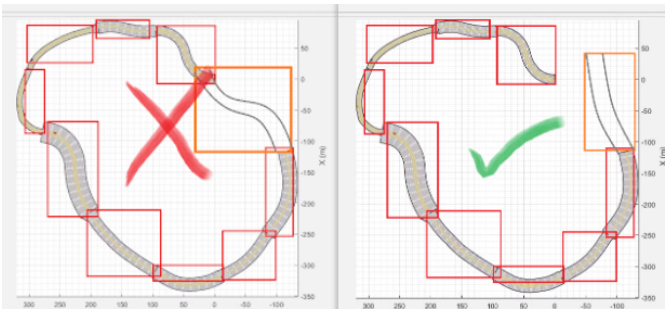

Fig. 7. Checking new road piece validity.

Figure 8 shows the number of failed pieces with increasing road length for a scenario of 10 road pieces. Based on 20 runs at each length, Figure 9 demonstrates what percentage of those runs are created without failing a piece, i.e. the runs where all pieces end up in the scenario. At a high enough length, it becomes virtually impossible to not fail a piece, making the check necessary for creating legal tests more efficiently.

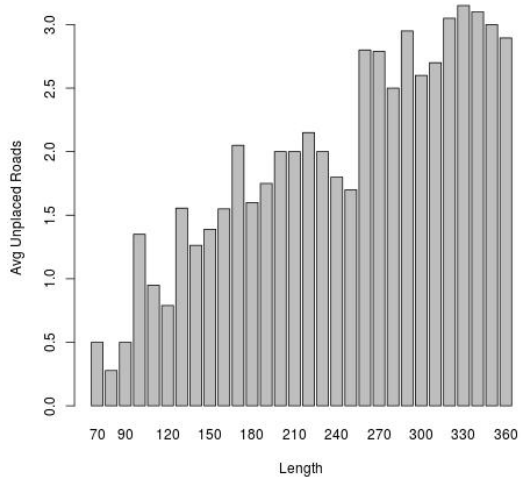

Fig. 8. Number of failed pieces with increasing road length.

For the actors, in order to minimize their box representation unrealistically occupying the same space, roads store information regarding the actors and at what point in the scene they are going over that lane. If two actors are at the same lane at the same point in the simulation, a different lane will be used or the actor will be made to slow down for a random but small period of time before moving forward. This does not apply to pedestrians as how they interact with each other is less critical in simulations.

V. Conclusion

The verification of AVs is critical for their deployment in real-life traffic. In this paper, we present a simulation framework for AV test scenario generation. For this purpose, we implement a simulation model that uses a semantic language to generate AV verification scenarios. Then we propose a road network generation mechanism for all possible street

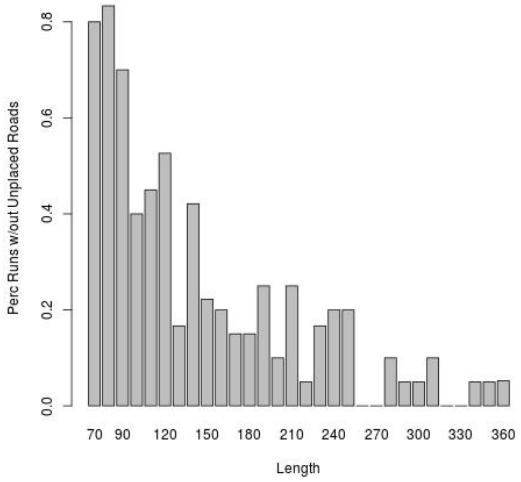

Fig. 9. Probability of generating scenarios without failed pieces.

network geometries. We also developed an iterative validity check for the generated scenarios in order to make the process of generating legal tests efficient.

As future work, we are going to incorporate a model of computation in our simulation framework and extend the road piece and traffic scenario database. We also plan to enable scenario export and import features to exchange scenarios automatically with other well-known tools such as OpenDRIVE.

References

[1] A. Lovins, *Reinventing fire: Bold business solutions for the new energy era*. Chelsea Green Publishing, 2013.

[2] SAE On-Road Automated Vehicle Standards Committee, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *SAE Standard J3016*, pp. 01–16, 2014.

[3] V. V. Dixit, S. Chand, and D. J. Nair, "Autonomous vehicles: disengagements, accidents and reaction times," *PLoS one*, vol. 11, no. 12, p. e0168054, 2016.

[4] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[5] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[6] D. Dolgov, "Google self-driving car project-monthly report-September 2016-on the road," Google, Tech. Rep., 2016.

[7] F. M. Favarò, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, "Examining accident reports involving autonomous vehicles in california," *PLoS one*, vol. 12, no. 9, p. e0184952, 2017.

[8] P. Sarhadi and S. Yousefpour, "State of the art: Hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, vol. 3, no. 4, pp. 470–479, 2015.

[9] M. İ. Akbaş, G. Solmaz, and D. Turgut, "Actor positioning based on molecular geometry in aerial sensor networks," in *IEEE International Conference on Communications (ICC)*, 2012, pp. 508–512.

[10] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: a new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, 2016.

[11] H. Winner, K. Lemmer, T. Form, and J. Mazzega, "PEGASUSFirst Steps for the Safe Introduction of Automated Driving," in *Automated Vehicles Symposium 2018*. Springer, 2018, pp. 185–195.

[12] I. The MathWorks, *MATLAB and Automated Driving System Toolbox*, Release R2018b, natick, Massachusetts, United States.

[13] TASS International, "PreScan," 2018.