SALVO: Automated Generation of Diversified Tests for Self-driving Cars from Existing Maps

Vuong Nguyen
University of Passau
Passau, Germany
nguyen58@ads.uni-passau.de

Stefan Huber
University of Passau
Passau, Germany
stefan.huber.niedling@outlook.com

Alessio Gambi
University of Passau
Passau, Germany
alessio.gambi@uni-passau.de

***Abstract*—Simulation-based tests are more cost-effective and less dangerous than field tests; hence, they are becoming the norm for thoroughly testing self-driving cars in virtual environments. High-quality simulation-based testing requires physically accurate computer simulations, detailed and photo-realistic maps, and systematic approaches for generating tests. Moreover, since creating detailed maps is a manual process, they are expensive, and testers should avoid wasting such valuable resources, for example, by generating irrelevant test cases. To address those issues, we propose SALVO a fully automated approach to identify quantifiably diverse and critical driving scenarios that can be instantiated on existing high-definition maps and implement them in an industrial driving simulator as executable test cases. The evaluation of SALVO in the context of the 2021 IEEE Autonomous Driving AI Test Challenge showed that it could analyze maps of different complexity and identify many critical driving scenarios in minutes. Furthermore, the tests SALVO generated stressed a state-of-art self-driving car software in quantifiably diverse ways and exposed issues with its implementation.**

***Index Terms*—Test Case Generation, Autonomous Vehicles, Software-in-the-loop, Diversity, Feature Space**

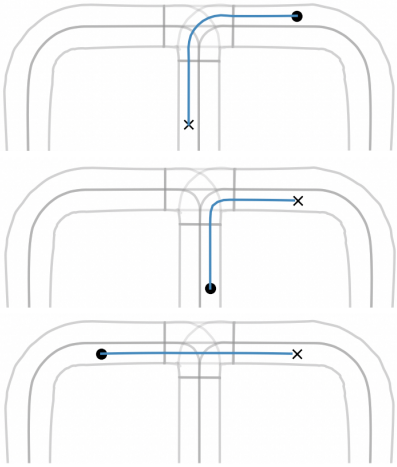

Fig. 1. Sample driving scenarios generated by SALVO from an intersection in the Cube Town map available for the 2021 IEEE Autonomous Driving AI Test Challenge. Driving scenarios begin before entering an intersection (•) and end after passing it (×). Top: left turn; Middle: right turn; Bottom: straight through.

I. Introduction and Motivation

Self-driving cars promise to reduce traffic accidents [1], increase fuel efficiency [2] and enhance comfort [3]. Testing self-driving car software is laborious and challenging since self-driving car's development is still in an early stage, and there is a lack of standardized procedures and established approaches [4]. Therefore, many researchers and organizations are currently investigating testing approaches to increase the quality of autonomous vehicles before they roam the thoroughfares. Naturalistic field operational testing is a popular approach to validate and verify the safety of autonomous vehicles that leaves them free to drive on actual roads. However, as discussed by Kalra and Paddock [5], this method is inefficient since relevant execution conditions (e.g., traffic situations that lead to traffic accidents) are hard to observe and ineffective as dangerous situations are impossible to recreate. Those, in turn, make limit the replicability of natural field operational tests' results [6]. An alternative approach is virtual testing, also known as simulation-based testing, which uses computer simulations to generate diverse test cases to challenge the self-driving software involving traffic patterns and numerous virtual road conditions [6]. Virtual testing enables automated test execution and can expose problems in autonomous software, including perception, planning, and decision-making systems. However, it requires the availability of accurate geometrical models (e.g., 3D models with textures and material properties) to achieve photo-realism, i.e., to close the simulation-to-reality gap [7], and detailed road maps. Consequently, cartography is an essential building block to conduct thorough virtual testing of autonomous vehicles [8]. Artists and domain experts cooperate to create those models and maps following a manual, laborious, and expensive process.

Detailed road maps consist of roads composed of lanes, their interconnections (e.g., intersections, junctions), and traffic regulations controlling them (e.g., speed limit, traffic direction); hence, they allow testers to create rich, coherent, and diversified driving scenarios [8], [9]. Among the existing formats to model road maps, in this work, we rely on the notion of *lanelets* [9], an innovative concept for road representation adopted by many (see Section III). Specifically, we rely on the semantics encoded in the lanelets representation of existing maps to identify intersections and create relevant driving scenarios around and across them. In other words, we abstract existing maps into the lanelets format to generate relevant

Fig. 2. Apollo Baidu (right, bottom) running with the SVL Simulator [17].

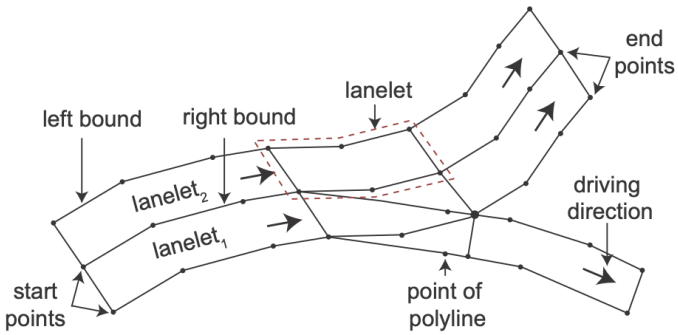

Fig. 3. An example of lanelets [8].

and diversified *abstract* driving scenarios. Next, we instantiate those abstract scenarios into concrete ones and generate simulation-based test cases that automatically challenge the autonomous vehicle's software in existing driving simulators.

In this paper, we present SALVO that implements our approach and summarize its evaluation in the context of the 2021 IEEE Autonomous Driving AI Test Challenge. SALVO, which is dubbed after the author's first names —Stephan, Alessio, and Vuong— and means "safe" in Italian, works under the consideration that because manually generated road maps are costly and valuable, testers must use (and re-use) them wisely and extensively. Consequently, and in contrast to many existing test case generators (e.g., [10]–[14]) that procedurally generate virtual roads, it leverages existing maps to generate as many tests as possible. However, since generating virtual tests without any guidance is not cost-effective either because not all the generated tests are relevant or because many tests are similar to each other, SALVO follows a systematic approach to generate only relevant test cases and maximize their diversity.

The current implementation of SALVO takes existing maps in OpenDrive format [15], generates abstract driving scenarios involving intersections, and selects among them those that result in quantifiably different ego-vehicle trajectories (see Figure 1). Finally, it instantiates the selected scenarios in the industrial driving simulation SVL Simulator [16] as test cases and automatically executes them.

To promote further research and enable reproducibility of our results, we release SALVO as open-source software. SALVO's source code and instructions to configure and run it are available at the following link:

https://github.com/TrackerSB/IEEEAITestChallenge2021

II. SVL Simulator

The SVL Simulator [16] is the driving simulation selected for the 2021 IEEE Autonomous Driving AI Test Challenge. It is an industrial driving simulator that provides the full simulation stack to test autonomous driving software: On one side, it provides a core simulation engine based on Unity for high fidelity, developer-friendly, cost-effective rigid-body simulation [18]; on the other side, it provides a communication bridge that allows connecting existing autonomous driving stacks such as Autoware [19] and Apollo Baidu [20] (see Figure 2). Moreover, the SVL Simulator implements additional features such as easily customizing sensors and an API that enables building and managing controllable objects, adjusting the core simulator's modules, and the automated generation and execution of driving scenarios.

III. Lanelets

Lanelets are a novel concept to represent road maps in both topological and geometrical terms. Lanelets model *drivable* road segments and roughly correspond to (segments of) road lanes [9]. They are defined by left and right bounds implemented as polylines (i.e., list of points), connected to define complex road maps, and overlap to form intersections, joins, and merges (see Figure 3). Since lanelets encode allowed traffic directions, they help determine the route from a predefined starting point to a given destination [21].

Lanelets can be connected to establish semantic relations that form longer road segments, intersections, merges and joins, and complex road networks. Therefore they can be used as an intermediate format to represent existing road maps [8]. For example, *adjacent* lanelets share one of the two bounds while *preceding/following* lanelets share start or end points in accordance to their traffic direction [8]. Notably, lanelets that overlap and share start or end points form intersections [22]. As we detail later, SALVO relies on those features to automatically identify the intersections in existing road maps by analyzing their lanelets representation and define (abstract) driving scenarios across them.

IV. Automatically Generating Diversified Tests from Existing Maps

This section describes our approach to analyze existing maps and generate sets of diversified driving scenarios using two metrics to establish tests diversity. Instead, the following sections present SALVO's evaluation in the context of the 2021 IEEE Autonomous Driving AI Test Challenge, discuss the achieved results and describe promising ideas for future work.

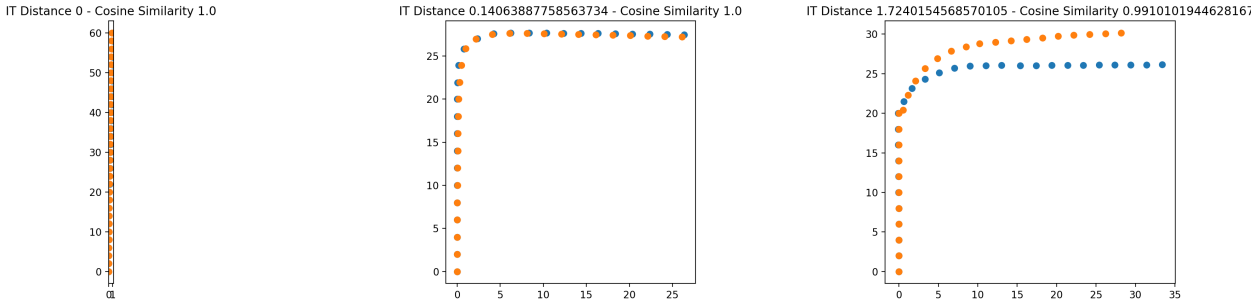

Fig. 4. Examples of similar trajectories. In each plot, we compare pairs of trajectories generated by SALVO and report their similarity computed using standard metrics such as the iterative Levenshtein distance (IT Distance) and Cosine similarity. For example, the first two plots result from the Cube Town map and illustrate cases of very similar roads, while the rightmost plot, which results from the Borregas Avenue map, shows partially overlapping trajectories.

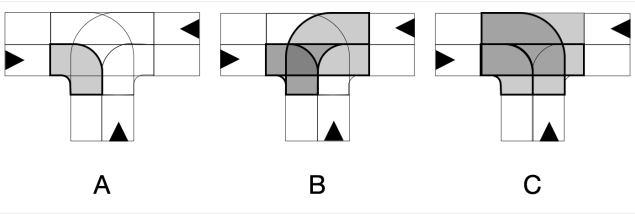

Fig. 5. An example of an intersection constructed by SALVO by transitively identifying overlapping lanelets.

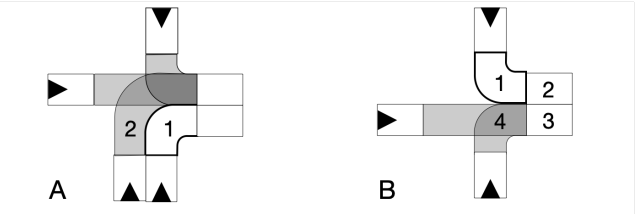

Fig. 6. An example of an intersection constructed by SALVO by exploiting the relationships among lanelets.

A. SALVO *in a Nutshell*

SALVO is entirely automatic and can generate test cases that stress the ego-vehicle in many different ways, possibly involving static obstacles that partially or completely occlude the lane in which the ego-vehicle is driving.

Given a map in OpenDrive format SALVO (1) identifies the intersections the map contains, (2) generates abstract driving scenarios that force the ego-vehicle to plan trajectories to cross those intersections safely, (3) selects abstract driving scenarios that maximize the diversity of the allegedly planned trajectories, and (4) generates executable test cases implementing the selected abstract driving scenarios.

SALVO can also extend the set of generated test cases by reusing existing ones and placing static obstacles, i.e., Non-Playable Characters (NPC) vehicles, in the lane occupied by the ego-vehicle. By doing so SALVO achieves two secondary goals: first, it further increases the diversity of the generated tests, as the obstacles force the ego-vehicle to plan new trajectories; second, it allows testing the ego-vehicle's object detection and collision avoidance in addition to route planning and lane-keeping.

The following sections detail each step of the proposed approach.

B. *Identifying Intersections*

SALVO uses the opendrive2lanelet library [8] to translate the original maps from the OpenDrive format to the lanelets format. As discussed in Section III, lanelets are connected by relations like 'follow' and 'adjacent' that define their semantics and provide geometrical information (e.g., their shape). However, since the lanelets format lacks a standardized way to capture intersections explicitly, i.e., via formalized relations between lanelets, SALVO implements a heuristic to identify them: First, it uses geometric information about lanelets to identify overlapping lanelets (Figure 5-A and Figure 5-B) as they are likely to belong to the same intersection. Second, since not all the lanelets in the same intersection directly overlap with each other, SALVO groups together lanelets that *transitively* overlap (Figure 5-C).

The intersections identified this way may be incomplete because, in some cases, lanelets that logically should belong together do not overlap with the other lanelets already assigned to those intersections. The bolded lanelets in Figure 6 exemplify two instances of this situation. SALVO identifies those missing lanelets by leveraging the relations defining the lanelet network. For example, for the case reported in Figure 6-A SALVO includes in the intersection Lanelet 1 because it is adjacent to a lanelet already assigned to that intersection, i.e., Lanelet 2. For the case reported in Figure 6-B, instead, SALVO includes Lanelet 1 because it precedes Lanelet 2 that is adjacent to Lanelet 3, the successors of a lanelet already assigned to that intersection, i.e., Lanelet 4. Noticeably, the heuristic implemented by SALVO to identify intersections exploiting the relations between lanelets and without user input is different from the one proposed by Klischat et al. [23] that instead relies on hierarchical clustering and requires the user to specify a minimum size for the intersection.

After identifying the intersections in the map, SALVO can generate driving scenarios that force the ego-vehicle to cross them.

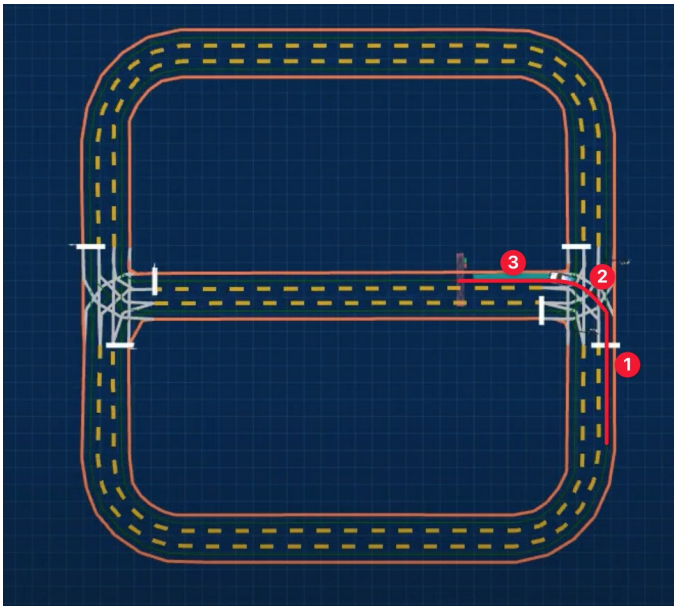

Fig. 7. An example of an abstract driving scenario generated by SALVO in the Cube Town map. The scenario starts on a lane (1), continues inside the intersection by following the lane (2), and ends on the same lane after passing the intersection (3).

C. Generating Relevant Scenarios

SALVO generates abstract driving scenarios that follow legal trajectories across intersections such as those reported in Figure 1. We define those trajectories as legal because they follow the traffic directions encoded into the lanelets. Additionally, we limit SALVO to consider only the simplest form of such trajectories, i.e., those that do not require the ego-vehicle to switch lanes. Therefore, a typical driving scenario generated by SALVO starts on a lane *before* an intersection, continues on the same lane across that intersection, and finally ends on the (logically) same lane *after* that intersection.

SALVO generates driving scenarios by selecting the lanelets that belong to an intersection (Figure 7) and then including those lanelets outside the intersection that precede and follow the selected ones. Notably, because intersections have various configurations (e.g., number of entering and exiting lanes), the generated driving scenarios can stress different behaviors of ego-vehicle by forcing it to turn left, turn right, or go straight through the intersection.

The generated scenarios are composed of the smallest number (i.e., three) of lanelets. This feature arguably increases the understandability of the resulting test cases but does not necessarily guarantee fast test executions. For example, the lanelets preceding or following an intersection may be long, requiring some time to be fully traversed during test execution. Therefore, to reduce execution time, SALVO places the ego-vehicle at a fixed distance before entering the intersection and sets the target position to reach at a fixed distance after passing the intersection. Because those distances are configurable, testers can control the duration of test executions and enable the generation of tests involving obstacles (see Section IV-F).

D. Selecting Diversified Scenarios

Up to this point, SALVO does not take the diversity of scenarios into account as it simply generates as many scenarios as possible to explore the possibilities offered by the input road map. However, many legal trajectories may stress the ego-vehicle similarly; for example, they may force it to drive straight through the same intersections from multiple directions. Therefore, SALVO filters out test cases that are too similar to each other to improve testing cost-efficiency. We experimented with two (greedy) approaches for filtering out test cases based on distance metrics and feature maps.

1) Filtering Similar Scenarios by Similarity: The first approach to select diversified scenarios relies on the similarity between the trajectories imposed by the driving scenarios on ego-vehicle. After an initial exploration considering standard metrics such as edit distance and cosine similarity (see Figure 4), we decided to adopt the Levenshtein distance to quantify the similarity of generated scenarios. The Levenshtein distance quantifies the similarity between trajectories by measuring how many *edits* are required to transform one trajectory into the other; the smaller the number of edits is, the more similar the two trajectories are. Notably, SALVO can use the Levenshtein distance because we represent trajectories as coordinates arrays, and edits correspond to insertions, deletions, and substitutions of arrays' entries. Specifically, SALVO calculates the Levenshtein distance between pairs of trajectories using the code provided by DeepJanus [12] and discards all the driving scenarios whose trajectories are too similar to the trajectories of driving scenarios already selected. Testers can control the similarity between the trajectories using a configurable threshold to create larger or smaller test suites. For completeness, in our evaluation we filtered out trajectories with a distance value smaller than 1.9.

2) Filtering Driving Scenarios using Feature Maps: The second approach to select diversified scenarios utilizes the feature maps defined by Zohdinasab et al. [13]. Trajectory features capture high-level characteristics of the abstract scenarios that are relevant for testing self-driving car software; examples of those features are the number of turns and the minimum curvature[1] that quantify how difficult it would be to follow the trajectory. Instead, feature maps place driving scenarios in a two-dimensional space according to the values taken by their features and provide a compact and understandable representation of their most salient aspects. Intuitively, scenarios with similar features are placed in nearby cells, whereas scenarios that are far away in the map have very different features (see Figure 8).

SALVO's goal is to generate driving scenarios that result in trajectories with different characteristics; therefore, given a feature map filled with all the generated trajectories, SALVO selects from each cell at most one trajectory (i.e., driving scenario) as the representative for the class of trajectories with the same feature combination while discarding the others.

[1]The minimum curvature highlights the presence of sharp turns.

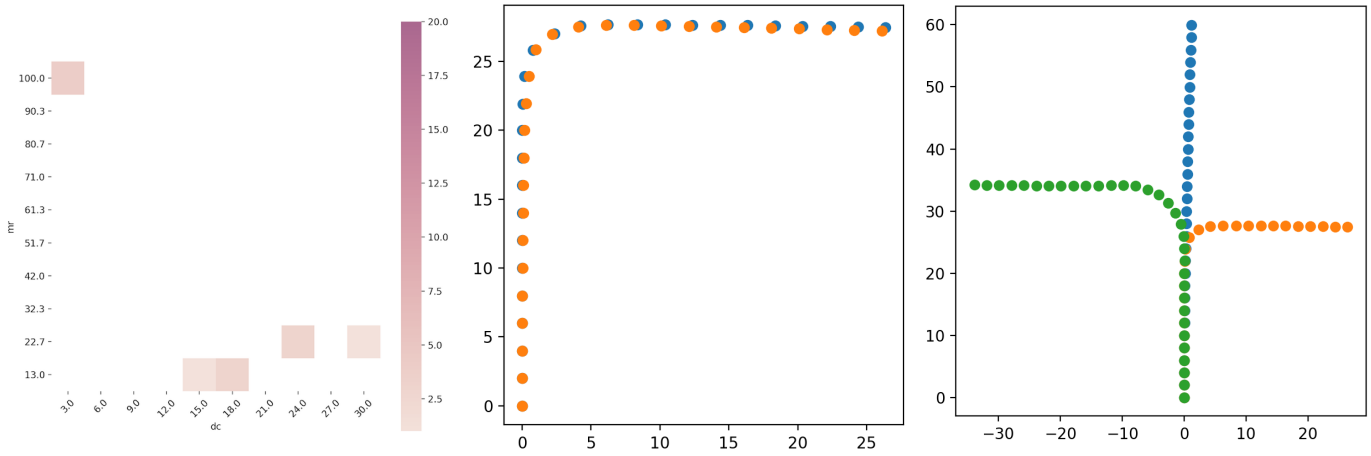

Fig. 8. Examples of feature map and trajectories for the Cube Town map. Left panel: The feature map generated by SALVO using direction coverage (dc) and minimum radius (mr) as features. Central panel: Similar trajectories sampled from the same cell in the feature map, i.e., $(6, 1)$. Right panel: Diverse trajectories sampled from distant cells in the feature map, i.e., $(10, 2), (1, 10), (6, 1)$.

Notably, using the feature maps, testers can easily quantify the coverage achieved by SALVO by counting the number of cells that contain at least one driving scenario. Likewise, content generators can use the feature maps to assess the "richness" of the road maps. For example, figures 9 – 13 show the road map (top plot) and the corresponding feature map (bottom plot) for the five maps we used in the 2021 IEEE Autonomous Driving AI Test Challenge (see Section V). From those figures, we can make the following observations: (1) road maps of increasing complexity, i.e., maps that contain more intersections, result in feature maps that are more covered, and (2) road maps with a repetitive structure result in feature maps with very dark cells, i.e., cells that contain many trajectories with very similar features.

E. Generating Test Cases

SALVO defines abstract driving scenarios as combinations of lanelets. So it needs to concretize them into executable test cases featuring test oracles that check if the ego-vehicle reaches the target location and if it does so within a predefined timeout. Also, SALVO must configure the driving simulation to place the ego-vehicle in the starting position and rotate it according to the road direction. However, doing so may not be enough to ensure that the generated tests are reliable and reproducible because the simulations include additional components with randomized behaviors that can affect the test results. For example, if the ego-vehicle needs to pass an intersection controlled by a traffic light, its state affects the behavior of the ego-vehicle: if the traffic light is red, the ego-vehicle must stop before it, which may cause the timeout to trigger a false positive. However, executing multiple times the same test may produce different results, hence flakiness, as the traffic light's initial state is initialized at random unless configured explicitly by the test. Therefore, to avoid false positives and reduce test flakiness, SALVO programs the traffic lights on the expected trajectories to turn green when the ego-vehicle approaches them.

F. Spicing Up the Test Cases

The tests generated by SALVO are diverse but plain: they stress the ability of the ego-vehicle to plan various trajectories to pass the intersections while keeping the lane but involve no other elements besides the ego-vehicle and the traffic lights. To further increase the diversity of the generated tests and enable testing additional features of the ego-vehicle such as object detection and collision avoidance, we let SALVO extend the basic tests by placing static obstacles, i.e., NPC vehicles, on the course of the ego-vehicle.

Our approach to "spice up" tests consists in placing an NPC vehicle in predefined positions between the ego-vehicle's initial position (i.e., the starting point of the driving scenario) and the intersection to pass such that the NPC vehicle partially or entirely occludes the lane that the ego-vehicles is supposed to keep. Default occlusions are partial and cause the ego-vehicle to swerve before entering the intersection; SALVO can place the NPC vehicle either on the left or right side of the lane, but not both. Additionally, but only if the geometry of the intersection allows it, SALVO can place the NPC in the middle of the lane in front of the ego-vehicle, forcing it to switch lanes before entering the intersection.

In summary, on top of the diversified test cases that SALVO generates from a given map, testers can quickly generate up to four times more test cases by simply letting SALVO add NPC vehicles in front of the ego-vehicle. Notably, we limit SALVO to place a single NPC vehicle per test case to keep them as simple and understandable as possible.

G. Prototype

We implemented SALVO in Python (v3.9) as a command-line application and integrated it with the SVL driving simulator (v2021.01) in accordance to the rules of the 2021 IEEE Autonomous Driving AI Test Challenge. For the evaluation, we ran SALVO on a gaming laptop running Ubuntu 20.04.2 LTS, the SVL driving Simulator, and Apollo Baidu (v6.0).

TABLE I
SALVO'S EVALUATION RESULTS: TEST GENERATION AND SELECTION

Map Name	Intersections	Test Cases		
		Total	Similarity	Feature Map
Cube Town	2	12	8	5
Borregas Ave	2	28	20	16
Shalun	14	149	124	33
Gomentum	25	126	110	25
San Francisco	89	806	535	52
Total	132	1121	797	131

TABLE II
SALVO'S EVALUATION RESULTS: TEST EFFECTIVENESS

Map Name	Similarity		Feature Map	
	Plain	NPC	Plain	NPC
Cube Town	3/8	1/3	0/5	1/3

V. EVALUATION

A. Experimental Settings

We evaluated SALVO according to the main objectives defined in the 2021 IEEE Autonomous Driving AI Test Challenge, i.e., generation of diverse tests, automated test generation and execution, and generation of tests that can find problems in Apollo Baidu, and the (limited) resources we had at the time the 2021 IEEE Autonomous Driving AI Test Challenge took place.

Since we deem the first two objectives more relevant than the last one, we devoted more effort to address them. Consequently, we evaluated the ability of SALVO to generate diverse and executable test cases automatically using various maps available for the SVL Simulator, but run the tests against Apollo Baidu only for one map.

As summarized in Table I, we selected five maps of increasing size (see figures 9 – 13) and complexity (see column *Intersections* in Table I). Our selection was limited by the availability of the maps and their compatibility with the opendrive2lanelets library. As summarized in Table II, we tested Apollo Baidu only on the Cube Town map with all the test cases generated from the plain scenarios selected using similarity and feature maps (columns *Plain*) and some test cases generated by placing NPC vehicles (columns *NPC*). Specifically, we created the NPC tests by randomly sampling three selected scenarios from each set and placing the NPC vehicle on the ego-vehicle's left, right, and front. The table reports the number of failed test cases followed by the number of total tests executed.

B. Achieved Results

As reported in Table I, SALVO identified more than a hundred intersections and generated more than a thousand driving scenarios that do not involve NPC vehicles. From those scenarios, it selected between 7% (San Francisco) and 57% (Borregas Ave) scenarios for the execution using features maps, whereas using similarity, it selected considerably more cases (up to 87% for Gomentum).

These results suggest that SALVO could handle maps of considerable size and effectively identify intersections in them. Furthermore, regarding test generation, it was able to identify many driving scenarios and selected among them different driving scenarios. The results also suggest that feature maps may be more stringent in selecting test cases than similarity computed using edit distance. However, this observation should be taken with care, as the number of selected test cases using feature maps depends on their configurations, i.e., the selected features and their granularity.

As reported in Table II, the execution of the generated test cases resulted in some failed test cases despite the simplicity of the map. Noticeably, all the failures are caused by the test subject not reaching the target location in time, meaning that the ego-vehicle did not crash into the NPC vehicles nor left the lane but could not plan or execute a suitable trajectory.

Regarding plain tests, we can observe that selecting tests using similarity resulted in more failures than selecting tests using the feature map. Instead, regarding NPC tests, we can observe that one test failed consistently in both configurations. Upon manual inspection of the failed tests, we noticed that the NPC vehicle was always parked in the middle of the ego-vehicle's lane. This observation suggests either the ego-vehicle failed to plan a trajectory that required to switch the lane or that, although we took extra care to remove possible causes of false positives (e.g., traffic lights, the minimal distance of the NPC car from the intersection), we may have missed something. For example, we may have oversight the possibility that ego-vehicle cannot invade the opposite traffic direction to pass the obstacle, forcing it to wait behind the obstacle indefinitely.

VI. FUTURE WORK

In the hope of capturing the intended aim of the challenge, we brainstormed possible ideas to implement for future work. We tried to "think out-of-the-box" while remaining pragmatic. Below we summarize two of such ideas.

a) Parking Lot Madness: Create scenarios that take place inside parking lots to test features like "Smart Summon." The idea is to check if the ego-vehicle can safely drive from a parking spot to the parking lot's exit despite pedestrians and other vehicles. Our vision is to generate scenarios by placing the ego-vehicle in different parking spots and controlling the placement and movement of NPC cars and pedestrians to create various critical situations.

b) To Pass or Not to Pass?: Check the behavior of the ego-vehicle while handling controlled intersections. The idea is to take any map, find where controlled intersections and traffic lights are, and (automatically) generate scenarios where the car must pass the various controlled intersections, possibly "trapping" the ego-vehicle in the middle of them by directly acting upon the traffic lights (e.g., flashing lights, turn lights off).

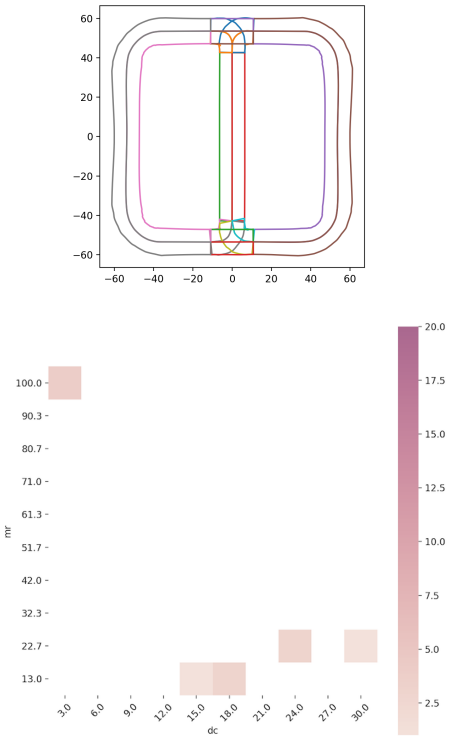

Fig. 9. Cube Town map.

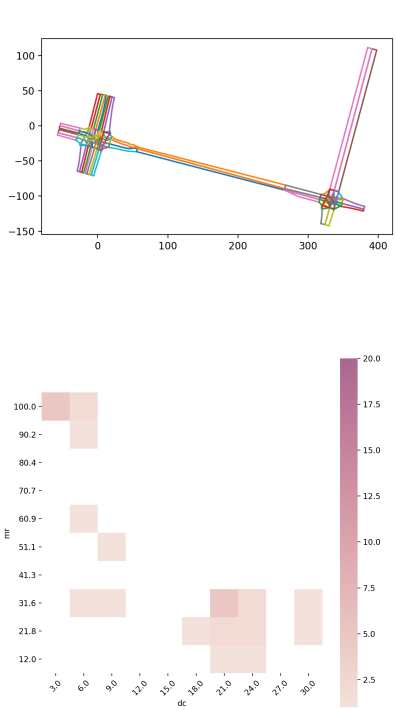

Fig. 10. Borregas Ave map.

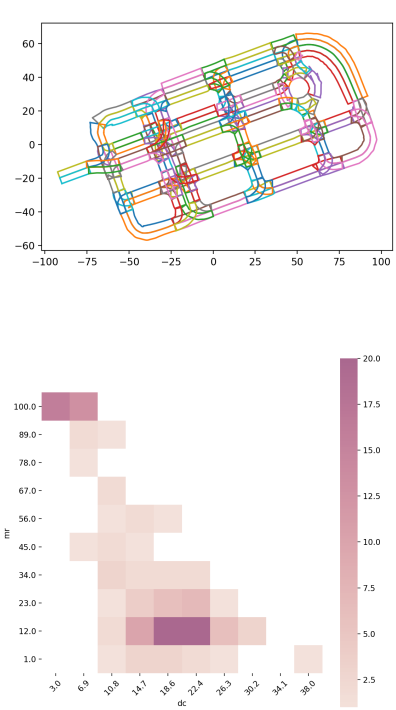

Fig. 11. Shalun map.

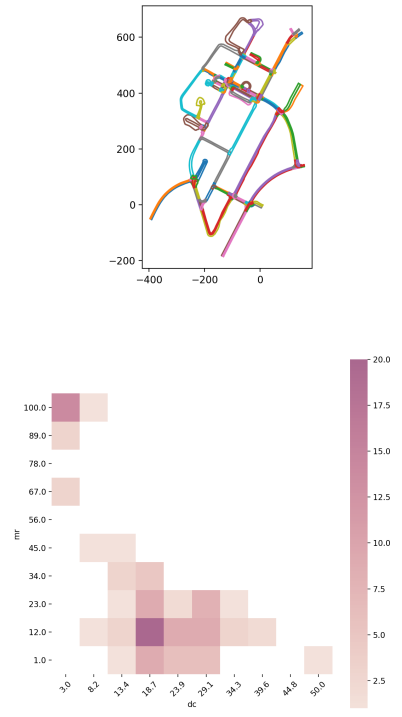

Fig. 12. Gomentum map.

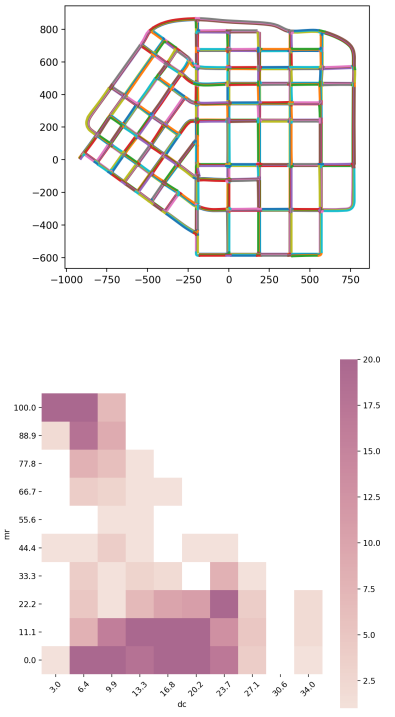

Fig. 13. San Francisco map.

References

[1] M. Peden, T. Toroyan, E. Krug, and K. Iaych, "The status of global road safety: the agenda for sustainable development encourages urgent action," *Journal of the Australasian College of Road Safety*, vol. 27, no. 2, pp. 37–39, 2016.

[2] H. Zamzuri, U. Z. A. Hamid, K. Pushkin, D. Gueraiche, and M. A. A. Rahman, "Current collision mitigation technologies for advanced driver assistance systems–a survey," *Perintis eJournal*, vol. 6, no. 2, pp. 78–90, 2016.

[3] C. D. Harper, C. T. Hendrickson, S. Mangones, and C. Samaras, "Estimating potential increases in travel with autonomous vehicles for the non-driving, elderly and people with travel-restrictive medical conditions," *Transportation research part C: emerging technologies*, vol. 72, pp. 1–9, 2016.

[4] A. Gambi, T. Huynh, and G. Fraser, "Generating effective test cases for self-driving cars from police reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 257–267.

[5] 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.

[6] J. Hasenau, K. Rogale, B. Lackowski, B. Maxim, K. Akingbehin, and J. Shen, "A virtual driving test environment for autonomous vehicles," in *Proceedings of the 16th annual International CAD Conference*, ser. CAD'19, 2019.

[7] N. Cruz and J. Ruiz-del Solar, "Closing the simulation-to-reality gap using generative neural networks: Training object detectors for soccer robotics in simulation as a case study," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[8] M. Althoff, S. Urban, and M. Koschi, "Automatic conversion of road networks from opendrive to lanelets," in *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, 2018, pp. 157–162.

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

[10] A. Gambi, M. Müller, and G. Fraser, "Asfault: testing self-driving car software using search-based procedural content generation," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 27–30. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00030

[11] T. Huynh, A. Gambi, and G. Fraser, "AC3R: automatically reconstructing car crashes from police reports," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 31–34. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00031

[12] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 876–888. [Online]. Available: https://doi.org/10.1145/3368089.3409730

[13] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Deephyperion: exploring the feature space of deep learning-based systems through illumination search," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, C. Cadar and X. Zhang, Eds. ACM, 2021, pp. 79–90. [Online]. Available: https://doi.org/10.1145/3460319.3464811

[14] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "SBST tool competition 2021," in *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 2021, pp. 20–27. [Online]. Available: https://doi.org/10.1109/SBST52555.2021.00011

road networks," in *Proc. of the Driving Simulation Conference Europe*, 2010, pp. 231–242.

[16] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "Lgsvl simulator: A high fidelity simulator for autonomous driving," 2020.

[17] L. E. Inc., "The svl simulator," https://www.svlsimulator.com/docs/archive/2020.06/apollo-master-instructions.

[18] J. Craighead, J. Burke, and R. Murphy, "Using the unity game engine to develop sarge: a case study," in *Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008)*, 2008.

[19] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015. [Online]. Available: https://doi.org/10.1109/MM.2015.133

[20] B. team, "Apollo: Open source autonomous driving," https://github.com/ApolloAuto/apollo, 2017.

[21] C. Pek, V. Rusinov, S. Manzinger, M. C. Üste, and M. Althoff, "Commonroad drivability checker: Simplifying the development and validation of motion planning algorithms," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020.

[22] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017.

[23] M. Klischat, E. I. Liu, F. Holtke, and M. Althoff, "Scenario factory: Creating safety-critical traffic scenarios for automated vehicles," in *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ITSC45102.2020.9294629