UC Berkeley
UC Berkeley Previously Published Works

Title
Addressing the IEEE AV Test Challenge with Scenic and VerifAI

Permalink
https://escholarship.org/uc/item/8xx5h6gt

ISBN
9781665434812

Authors
Viswanadha, Kesav
Indaheng, Francis
Wong, Justin
et al.

Publication Date
2021-08-26

DOI
10.1109/aitest52744.2021.00034

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Addressing the IEEE AV Test Challenge with SCENIC and VERIFAI

Kesav Viswanadha†, Francis Indaheng†, Justin Wong†,
Edward Kim†, Ellen Kalvan‡, Yash Pant†,
Daniel J. Fremont‡, Sanjit A. Seshia†
† University of California, Berkeley
‡ University of California, Santa Cruz

2021 IEEE International Conference On Artificial Intelligence Testing (AITest) | 978-1-6654-3481-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/AITEST52744.2021.00034

***Abstract*—This paper summarizes our formal approach to testing autonomous vehicles (AVs) in simulation for the IEEE AV Test Challenge. We demonstrate a systematic testing framework leveraging our previous work on formally-driven simulation for intelligent cyber-physical systems. First, to model and generate interactive scenarios involving multiple agents, we used SCENIC, a probabilistic programming language for specifying scenarios. A SCENIC program defines an *abstract* scenario as a distribution over configurations of physical objects and their behaviors over time. Sampling from an abstract scenario yields many different *concrete* scenarios which can be run as test cases for the AV. Starting from a SCENIC program encoding an abstract driving scenario, we can use the VERIFAI toolkit to search within the scenario for failure cases with respect to multiple AV evaluation metrics. We demonstrate the effectiveness of our testing framework by identifying concrete failure scenarios for an open-source autopilot, Apollo, starting from a variety of realistic traffic scenarios.**

1. Introduction

Simulation-based testing has become an important complement to autonomous vehicle (AV) road testing. It has found a prominent role in government regulations for AVs, for example, one of the National Highway Traffic Safety Administration (NHTSA) missions [10] states that AVs should be tested in simulation prior to deployment. Waymo, a leader in the AV industry, has used simulation-based test results to support its claim that its autopilot is safer than human drivers [9].

However, there are fundamental challenges that need to be addressed first to meaningfully test AVs in simulation. First, the simulation must effectively capture the complexities of real-world environment, including the behaviors of traffic participants (e.g. pedestrians, human drivers, cyclists, etc), their interactions and physical dynamics, and the the roads and other infrastructure around them. Furthermore, tool support is necessary to (i) specify multiple evaluation metrics with varying priorities, (ii) monitor the performance of the AV according to the specified metrics, and (iii) search for failure scenarios where performance does not meet requirements.

This report summarizes how we formally address these fundamental challenges as participants in the IEEE Autonomous Driving AI Test Challenge. To *model* and *generate* interactive, multi-agent environments, we use the formal scenario specification language SCENIC [6], [7]. A SCENIC program defines an *abstract* scenario as a distribution over *scenes* and behaviors of agents over time; a scene is a snapshot of an environment at any point in time, meaning a configuration of physical objects (e.g. position, heading, speed). Sampling from an abstract scenario yields many different *concrete* scenarios which can be run as test cases for the AV. Henceforth we will refer to abstract scenarios simply as "scenarios".

Using SCENIC, developers can intuitively model abstract scenarios, rather than coding up specific concrete scenarios, by specifying distributions over scenes and behaviors. In conjunction with SCENIC, we used the VERIFAI toolkit [4] to specify multi-objective evaluation metrics (e.g. do not collide while reaching a destination), monitor AV performance, and search for failure cases by sampling from the distributions in the scenario encoded in SCENIC. We tested an open-source autopilot, Apollo 6.0 [1][1], in the LG SVL simulator [2] via a variety of test scenarios derived from a NHTSA report [10]. Using this architecture, we were able to achieve a high variety of scenarios that highlighted several issues with the Apollo AV system, and we provide some quantitative measures of how well the space of potential scenarios has been covered by our framework.

1. In the rest of the report, Apollo refers to Apollo 6.0.

978-1-6654-3481-2/21/$31.00 ©2021 IEEE
DOI 10.1109/AITEST52744.2021.00034

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

```
behavior PullIntoRoad(laneToFollow, target_speed):
    while (distance from self to ego) > CUT_IN_TRIGGER_DISTANCE:
        wait
    do FollowLaneBehavior(laneToFollow=ego.lane, target_speed)

behavior EgoBehavior(target_speed):
    try:
        do FollowLaneBehavior(target_speed)
    interrupt when withinDistanceToAnyObjs(self, SAFETY_DISTANCE):
        do CollisionAvoidance()

target_speed = 5 # meter / second
ego = Car with behavior EgoBehavior(target_speed)
spot = OrientedPoint on visible curb
badAngle = Uniform(-1,1) * Range(10,20) deg
parkedCar = Car left of spot by 0.5, # meter
            facing badAngle relative to roadDirection,
            with behavior PullIntoRoad(ego.lane, target_speed)
require (distance from ego to parkedCar) < 20
require eventually (ego.lane is parkedCar.lane)
```

Figure 1: An example SCENIC program modeling a badly-parked car pulling into the AV's lane

1.1. Background

We give here some background on the two main tools previously developed by our research group which we utilize in our approach.

SCENIC [6], [7] is a probabilistic programming language whose syntax and semantics are designed specifically to model and generate scenarios. A scenario modelled in this language is a program, and an execution of this SCENIC program in tandem with a simulator generates concrete scenarios. SCENIC provides intuitive and interpretable syntax to model the spatial and temporal relations among objects in a scenario. The probabilistic aspect of the language allows users to specify distributions over scenes and behaviors of objects. The parameters of the scenes and behaviors, from initial positions to controller parameters, form the *semantic feature space* of the scenario. Testing with a SCENIC program involves sampling concrete scenarios from this semantic feature space. An example of a SCENIC program, describing a badly-parked car in the ego AV's lane, is shown in Figure 1. Please refer to [7] for a detailed description of SCENIC.

VERIFAI [4] is a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components. The architecture of VERIFAI is shown in Fig. 2. As inputs, it takes the environment model encoded in SCENIC, system specifications or evaluation metrics, and the system being to be tested. VERIFAI extracts the semantic feature space defined by the SCENIC model, and searches this space for violations of the specification. Concrete scenarios sampled from the space are executed in an external simulator, while the system's performance is monitored and logged in an error table which stores the results of all simulations. VERIFAI

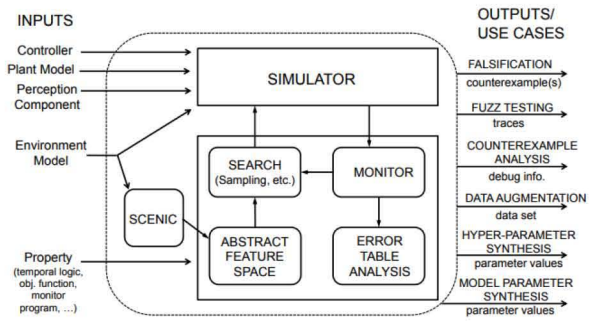

Figure 2: The architecture of the VERIFAI toolkit, from [4].

employs a variety of sampling strategies to perform *falsification*, i.e., to search for scenarios that induce the system to violate its specification. These include passive samplers based on random sampling or Halton sequences [8], as well as active samplers which use the history of the system's performance on past tests to guide search towards counterexamples, such as cross-entropy optimization, simulated annealing, and Bayesian optimization. Recently, we added a multi-armed bandit sampler which supports multi-objective optimization [12].

2. Metrics and Scenarios

This section presents the safety properties and associated metrics that we use to evaluate an AV's performance, as well as the scenarios these metrics are computed over.

2.1. Safety Properties and Metrics

For all of our generated scenarios, we use the following four safety metrics, based on the evaluation criteria proposed by Wishart et al. [13]. The mathematical formulations for each metric are also given below.

1) *Distance*: We require that for the full duration of a simulation, the ego vehicle stays at least some minimum distance away from every other vehicle in the scenario. In our case, the distance between the centers of the AV and any other vehicle must be greater than 5 meters.

$$\text{always}(\text{distance}(\text{ego}, \text{other}) \geq 5)$$

2) *Time-to-Collision*: Given the current velocities and positions of the ego and adversary vehicles, the amount of time that it would take the vehicles to be within 5 meters of each other

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

if they maintained their current projected trajectories must always be above 2 seconds. Let $x(t)$ represent the ego vehicle's future position vector as a function of time, using its current position x_0 and its current velocity v_0, and similarly for another vehicle $x'(t)$ with position x'_0 and v'_0. Also, let $\|\cdot\|$ be the Euclidean (L2) norm of the input. Then, we have:

$$x(t) = x_0 + v_0 t$$

$$x'(t) = x'_0 + v'_0 t$$

$$\|x(t) - x'(t)\| = 5 \tag{1}$$

Let t_1 and t_2 be the solutions to Equation 1, which is a quadratic equation in t. We assert that:

$$\text{always}(\min(t_1, t_2) \geq 2 \text{ or } \max(t_1, t_2) \leq 0)$$

This corresponds to either a future near-miss/collision being at least 2 seconds away, or being in the past (i.e. the distance between the vehicles is increasing over time).

3) *Made Progress*: Measures that the ego vehicle moved at least 11 meters from its initial position. This is useful for checking that Apollo is properly communicating with the SVL Simulator, which was not always the case. Let X be an array containing the ego vehicle's position vector at each timestep, for N timesteps. Then, we have:

$$\|X_1 - X_N\| \geq 11$$

4) *Lane Violation*: The ego vehicle's average distance from the centerline of its current lane over the entire trajectory must be less than 0.5 meters. Let ℓ_i be the center of the current lane at timestep i. Using the positions stored in X as defined in the previous metric, this gives us:

$$\frac{1}{N}\left(\sum_{i=1}^{N} \|X_i - \ell_i\|\right) \leq 0.5$$

We encode each of these metrics using monitor functions in VERIFAI [4]. We then run multi-objective falsification to attempt to find scenarios that violate as many of these metrics at the same time as possible [12].

2.2. Evaluation Scenarios

The test scenarios we selected are scenarios from the NHTSA report [10], encoded as SCENIC programs. We consider there to be three main components of a scenario, namely: 1) road infrastructure (e.g. 4-way intersection), 2) the type and number of traffic participants (e.g. car, pedestrian, bus), and 3) their behaviors (e.g., lane change). A brief description of the test scenarios studied in this report is shown in Table 1.

Scenario #	Road Infrastructure	Traffic Participants	Behaviors
1	4-way intersection	3 sedan cars	AV performs a lane change / low-speed merge
2	1-way road	2 sedan cars	AV performs vehicle following with a leading car
3	2-way road	4 sedan cars	AV parallel parks between cars
4	2-way road	1 sedan car 1 school bus	AV detect and respond to school bus
5	2-way road	2 sedan cars	AV responds to encroaching oncoming car
6	1-way road	1 sedan car 1 pedestrian	AV detects and responds to pedestrian crossing
7	4-way intersection	1 sedan car	AV crosses intersection while oncoming car makes unprotected left-turn across path
8	4-way intersection	2 sedan cars	AV makes unprotected left turn at intersection while car from lateral lane cuts across path
9	4-way intersection	2 sedan cars	AV makes right turn at intersection while car from lateral lane passes
10	4-way intersection	1 sedan car 1 pedestrian	AV makes unprotected left turn at intersection while pedestrian crosses

TABLE 1: Description of the scenarios tested.

3. AV Simulation Test Scenario Generation

The purpose of test scenario generation is to search for failure cases while also aiming to cover the test space well. Here, failure is defined by the evaluation metrics used for testing (Sec. 2.1), and the test space is comprised of the parameters defined in a SCENIC program. Any identified failure cases can inform the AV developers of potential susceptibilities of their system. However, biased search towards failures may only ex-

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

ploit a subset of the test space and thereby only provide a partial assessment of the robustness of the system in an abstract scenario. Hence, balancing this trade-off is the key in determining which concrete test scenarios to generate. In this section, we elaborate on our design choices to address this trade-off and highlight a workflow for AV simulation testing.

3.1. Scenario Generation Workflow.

We write a series of SCENIC programs for the purpose of generating a wide range of concrete scenarios, corresponding to the generic driving situations described in Table 1. We decided which parameters to vary in each scenario based on empirical observation on what seemed to produce and interesting and diverse set of simulations. Some of the parameters that are varied are sampled from continuous ranges, such as speeds and distances from specific points like intersections, whereas others are discrete, such as randomly choosing a lane for the AV's position from all possible lanes in the map file. A given SCENIC program which encodes an abstract scenario (as defined in the Introduction) is given as input to VERIFAI toolkit, which compiles the program to identify the semantic feature space as defined in Sec. 1.1. Using one of the supported samplers in VERIFAI, a concrete test scenario is sampled to generate a single simulation. Specifically, this sampling consists of *static* and *dynamic* aspects. For each sampling process, an initial scene (e.g. position, heading) is sampled at the beginning and is sent to the simulator for instantiation. During the simulation run-time, distributions over behaviors are sampled dynamically. This dynamic aspect enables generating interactive environment. Specifically, at every simulation timestep, the simulator and the compiled SCENIC program communicates a round of information in the following way. The simulator sends over the ground truth information of the world to the SCENIC program, and the program samples an action for each agent in the scenario in accordance with specified distribution over behaviors. These dynamically sampled actions are simulated for a single simulation timestep. This round of communication continues until a termination condition is reached. At this point, the trajectory is input to a *monitor function* in VERIFAI, which computes the values of the safety metrics described in Section 2.2 and determines whether the scenario constitutes a violation or not.

3.2. Scenario Composition

An important feature of SCENIC that we can leverage is the ability to define sub-scenarios that can be composed to construct higher-level scenarios of greater complexity [7]. This facilitates a modular approach to writing simple scenarios that can be reused as the components of a broader scenario. For example, provided a library that includes an intersection scenario, a pedestrian scenario, and a bypassing scenario, SCENIC supports syntax to form arbitrary compositions of these individual scenarios. Furthermore, we adopt an *opportunistic* approach to composition in which the SCENIC server invokes the desired challenge behavior if the circumstances of the present simulation allow for it. In the same example, as the ego vehicle drives its path, the SCENIC server will monitor the environment; if, say, the ego vehicle approaches an intersection, the SCENIC server will dynamically create the agents specified in the intersection sub-scenario, allow them to enact their behaviors, and subsequently destroy the agents as the ego vehicle proceeds beyond the intersection. This method of composition results in stronger testing guarantees of an AVs performance by providing a sort of integration test that extends beyond the isolated unit test structure of evaluating the AV against scenarios on an individual basis. This yields more opportunities to discover faults in the AV that may otherwise be forgone using only isolated scenario tests.

3.3. Applied Sampling Strategies

We navigate the trade-off between searching for failures (i.e. exploitation) and coverage of semantic feature space (i.e. exploration) by using SCENIC's ability to write scenarios with parameters that can be searched using any of VERIFAI's samplers [5]. Specifically, we used Halton and Multi-Armed Bandit (MAB) samplers. The Halton sampler is a passive sampler which guarantees exploration but not exploitation. The MAB sampler is an active sampler that focuses on balancing the exploration/exploitation trade-off [12].

- **Multi-Armed Bandit (MAB) Sampler.** Viewing the problem as a multi-armed bandit problem, minimizing long-term regret [3] is adopted as a sampling strategy. This simultaneously rewards coverage early while prioritizing finding edge cases as coverage improves.
- **Halton Sampler.** By dividing the semantic feature space into grids and minimizing discrepancy, the Halton sampler prioritizes coverage [8]. Halton sampling covers the entire space evenly in the limit as more samples are collected.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

4. AV Simulation Test Results

4.1. Scenario classes and test coverage

Scenario Diversity: As mentioned in Sec. 2.2, we selected NHTSA scenarios that cover a range of combinations of road infrastructure, traffic participants, and behaviors, and coded these up as abstract scenarios in SCENIC. There are a few factors that contribute to the diversity of our generated test scenarios; first and foremost, every scenario is encoded as a SCENIC program, which allows for many factors in the generated simulations to be randomized, including the starting positions of the vehicles, their speeds, the weather, and other pertinent parameters of specific scenarios. Furthermore, each non-ego vehicle has a *behavior* associated with it that describes its actions over the course of a simulation. Behaviors themselves contain logic that allows randomization of various aspects of the motion [7]. Because of this, a single SCENIC program yields a large variety of possible concrete scenarios to generate. SCENIC and VERIFAI, in turn, sample from this large space of possible scenarios. Our method for approximating a coverage metric of this SCENIC-defined space of is concrete scenarios discussed below.
Coverage Metric: We implement an estimator for ϵ-coverage [11]. The intuition of this metric is the following. A sampled concrete test scenario corresponds to a point in the semantic feature space of a SCENIC program. After sampling and generating multiple concrete test scenarios, the ϵ-coverage computes the smallest radius, ϵ, such that the union of ϵ-balls centered on each sampled point fully covers the semantic feature space.

However, this is inefficient to compute exactly, so we approximate this by placing a ϵ'-mesh over the feature space and searching for the finest mesh where every mesh point's nearest neighbor is within ϵ'. This can be implemented by performing nearest neighbor search over sampled semantic vectors with mesh points as queries. We then perform binary search on values of ϵ' until the search interval is within a given tolerance (set to 0.05 units for our experiments).

This metric, qualitatively speaking, tells us how large of an area of unexplored space there is in the feature space. Therefore, the larger the ϵ value computed by this metric, the less coverage we have of the feature space. One caveat of ϵ-coverage is that it computes the mesh over the entire feature space, which may not necessarily be the same as the *feasible space* of samples — that is, regions of the feature space which actually lead to valid simulations. This feasible space is usually difficult to calculate exactly *a priori*, and so we make a generalization in this case to use the entire feature space, which is known beforehand. We found that for most of the scenarios, this did not make a huge difference as valid simulations could be found throughout the feature spaces. Another point to note about this metric is that it is computed using only the continuous features sampled by VERIFAI, as computing coverage for sampled variables in SCENIC is much more difficult given the possible dependencies between variables and their underlying distributions. Therefore, this metric is only an approximation of a subset of all of the features sampled by VERIFAI and SCENIC.

4.2. Statistics and distribution report

For our experiments, we ran several of our abstract SCENIC scenarios for 30 minutes each, using both the Halton and Multi-Armed Bandit sampler on each scenario. We use a VERIFAI monitor that combines all of the metrics from Section 2.1. The results, shown in Table 2, demonstrate that we are able to find several falsifying counterexamples for our various metrics for each scenario.

Scenario	Sampler	Total Samples	Progress	Distance	TTC	Lane	ϵ
1	Halton	52	0	52	52	14	–
	MAB	56	2	56	56	54	–
2	Halton	56	6	9	11	0	1.245
	MAB	53	3	11	11	0	5.249
3	Halton	51	1	51	51	44	–
	MAB	52	18	52	48	34	–
4	Halton	60	0	55	58	0	0.415
	MAB	60	0	56	58	0	6.079
5	Halton	76	8	30	30	11	0.903
	MAB	53	8	47	47	3	49.976
6	Halton	61	2	54	54	2	9.497
	MAB	60	11	50	50	3	19.702
7	Halton	61	0	0	0	0	0.122
	MAB	58	0	0	0	0	1.099
8	Halton	57	0	0	0	8	1.392
	MAB	56	0	0	0	1	3.003
9	Halton	55	0	0	0	1	1.294
	MAB	53	0	0	0	0	2.759
10	Halton	60	0	5	6	0	8.081
	MAB	58	0	0	0	0	31.128

TABLE 2: The number of samples and property violations found for each scenario, along with the ϵ-coverage metric.

We found that in many of these scenarios, the distribution of safety violations across the feature space was roughly uniform. Because of this, the use of multi-armed bandit sampling did not result in a significantly higher number of violations than using Halton sampling. Moreover, as seen in Fig. 3, in some of the scenarios multi-armed bandit sampling did not fully explore the search space. We hypothesize that this may have been due to the low number of samples used in our experiments; further investigation is required to understand the MAB sampler's behavior in these cases.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

For each scenario, we also present our coverage metric based on the semantic feature space defined in the SCENIC program. In all scenarios for which the coverage metric was computed, the value of ϵ is far lower for Halton sampling than it is for MAB sampling, which means Halton gives us better coverage as expected. The plot in Figures 3 reflects the values of these metrics as there is geometrically more empty space in the overall feature space defined by the SCENIC program for MAB sampling than there is for Halton sampling. The metric was not computed for Scenarios 1 and 3 because there were no continuous-valued features sampled by VERIFAI in those scenarios.

4.3. Safety violations discovered via AV testing

Our sampling-based testing approach uncovered multiple safety violations, some of which were a result of the following unsafe behaviors we observed in the simulations:

1) In multiple runs of scenarios involving intersections, Apollo gets "stuck" at stop signs and doesn't complete turns.
2) Apollo disregards pedestrians entirely, even though they are visible to the vehicle as shown in the Dreamview UI.
3) With non-negligible probability, Apollo fails to send a routing request using the `setup_apollo` method in the PythonAPI. Because of this, the car does not move much beyond its starting position, hence the *made progress* metric that we included in our experiments.
4) We also noticed that Apollo sometimes stops far beyond the white line at an intersection, something that would likely present a hazard to other drivers in a real-world scenario.
5) In the "vehicle following" scenario (Scenario 2 in Table 2), we were able to find a few dozen samples where Apollo collided with the vehicle in front of it. A few of these examples are also included in our simulation test reports generated by the SVL Simulator.

Depending on the scenario, we found that the number of simulations in which a metric was violated could be quite high. For example, in the pedestrian scenario, almost all of the simulations violated the *time-to-collision* metric, indicating that perhaps the vehicle was approaching the pedestrian much more quickly than what would feel safe to a human passenger.

Additional materials, including surce of the Scenic programs, videos of a few simulations, and test reports generated by the LGSVL Simulator, are available at this Google Drive link.

5. Conclusion

We demonstrated the effective use of our formal simulation-based testing framework. Our use of SCENIC to model abstract scenarios had the following benefits: (i) concisely represent distributions over scenes and behaviors and (ii) embed diversity into our test scenario generation process, which involves sampling concrete test scenarios from the SCENIC program. To enable intelligent sampling strategies to search for failures, we used the VERIFAI toolkit. This tool supported specifying multi-objective AV evaluation metrics and various passive and active samplers to search for concrete failure scenarios. Using our methodology, we were able to identify several undesirable behaviors in the Apollo AV software stack.

Our experiments for the IEEE AV Test Challenge also uncovered some limitations that would be good to address in future work. For example, running simulations was computationally expensive and limited the number of samples different search/sampling strategies could generate, which seemed to hurt the performance of certain samplers (e.g. the MAB sampler) compared to others from our previous study [12].

References

[1] Apollo: Autonomous Driving Solution. http://apollo.auto/. Last accessed: 07-22-2021.

[2] Advanced Platform Team, LG Electronics America R&D Lab. LGSVL Simulator. https://www.lgsvlsimulator.com/. Last accessed: 07-22-2021.

[3] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, and Peter Auer. Upper-confidence-bound algorithms for active learning in multi-armed bandits. In Jyrki Kivinen, Csaba Szepesvári, Esko Ukkonen, and Thomas Zeugmann, editors, *Algorithmic Learning Theory*, pages 189–203, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *31st International Conference on Computer Aided Verification (CAV)*, pages 432–442, 2019.

[5] Daniel J. Fremont, Johnathan Chiu, Dragos D. Margineantu, Denis Osipychev, and Sanjit A. Seshia. Formal analysis and redesign of a neural network-based aircraft taxiing system with verifai. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2020.

[6] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: a language for scenario specification and scene generation. In Kathryn S. McKinley and Kathleen Fisher, editors, *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 63–78. ACM, 2019.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.

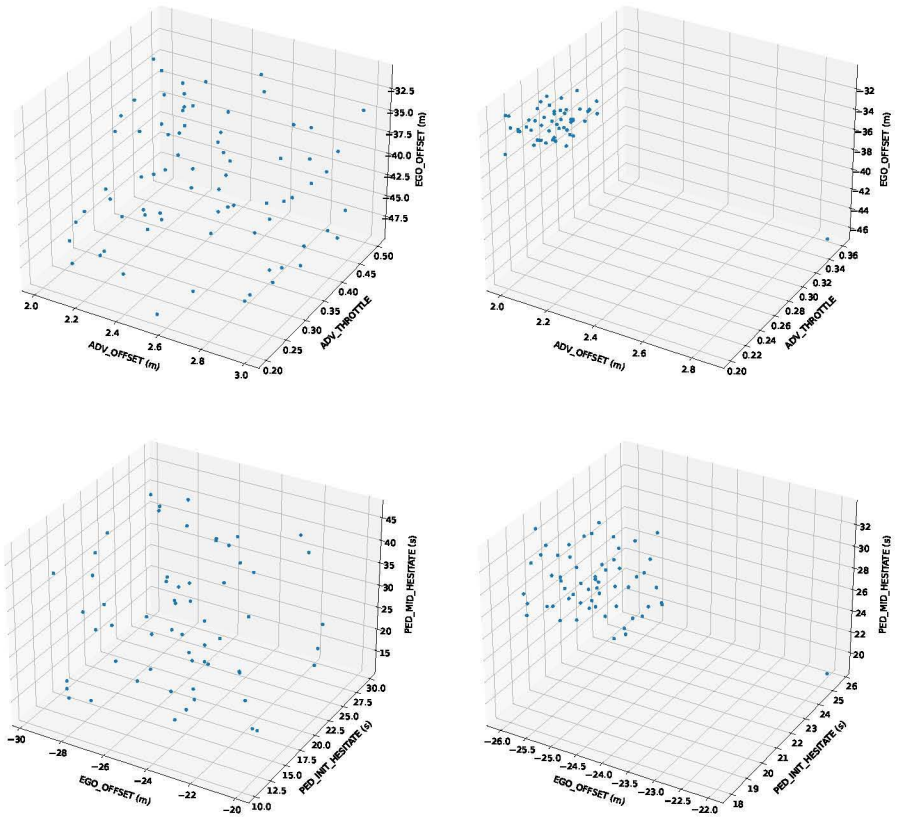

Figure 3: The space of points generated by the Halton (left) and MAB samplers (right) for scenario 5 (top plots) and Scenario 6 (bottom plots). In Scenario 5, the axes respectively are the adversary vehicle's distance from the intersection, the adversary vehicle's throttle value, and the AV's distance from the intersection. In Scenario 6, the axes respectively are the AV's distance from the intersection, how long the pedestrian waits before starting to cross, and how long the pedestrian waits in the intersection.

[7] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and data generation. *CoRR*, abs/2010.06580, 2020.

[8] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 1960.

[9] Andrew J. Hawkins. Waymo simulated real-world crashes to prove its self-driving cars can prevent deaths. *The Verge*, 03-08-2021.

[10] National Highway Traffic Safety Administration (NHTSA). Automated driving systems. https://www.nhtsa.gov/vehicle-manufacturers/automated-driving-systems. Last accessed: 07-22-2021.

[11] Wilson A Sutherland. *Introduction to metric and topological spaces*. Oxford University Press, 2009.

[12] Kesav Viswanadha, Edward Kim, Francis Indaheng, Daniel J. Fremont, and Sanjit A. Seshia. Parallel and multi-objective falsification with Scenic and VerifAI. https://arxiv.org/abs/2107.04164, 2021.

[13] Jeffrey Wishart, Steven Como, Maria Elli, Brendan Russo, Jack Weast, Niraj Altekar, and Emmanuel James. Driving safety performance assessment metrics for ADS-equipped vehicles. 04 2020.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on January 11,2022 at 17:31:07 UTC from IEEE Xplore. Restrictions apply.