AmbieGen: A Search-based Framework for Autonomous Systems Testing

Dmytro Humeniuk, Foutse Khomh, Giuliano Antoniol

Polytechnique Montréal, 2500 Chemin de Polytechnique, QC H3T 1J4, Montréal, Canada

Abstract

Thorough testing of safety-critical autonomous systems, such as self-driving cars, autonomous robots, and drones, is essential for detecting potential failures before deployment. One crucial testing stage is model-in-the-loop testing, where the system model is evaluated by executing various scenarios in a simulator. However, the search space of possible parameters defining these test scenarios is vast, and simulating all combinations is computationally infeasible. To address this challenge, we introduce AmbieGen, a search-based test case generation framework for autonomous systems. AmbieGen uses evolutionary search to identify the most critical scenarios for a given system, and has a modular architecture that allows for the addition of new systems under test, algorithms, and search operators. Currently, AmbieGen supports test case generation for autonomous robots and autonomous car lane keeping assist systems. In this paper, we provide a high-level overview of the framework's architecture and demonstrate its practical use cases.

Keywords: evolutionary search, autonomous systems, self driving cars, autonomous robots, neural network testing

Metadata

The project metadata is presented in Table 1.

1. Motivation and significance

Autonomous systems, including autonomous vehicles, robots, or drones can provide a number of benefits such as driving assistance, high-risk zone

Preprint submitted to Science of Computer Programming

January 4, 2023

Nr.	Code metadata description	Please fill in this column		
C1	Current code version	v0.1.0		
C2	Permanent link to code/repository	For example: https://github.		
	used for this code version	com/swat-lab-optimization/		
		AmbieGen-tool		
C3	Permanent link to Reproducible	https://codeocean.com/		
	Capsule	capsule/1741442/tree		
C4	Legal Code License	MIT license (MIT)		
C5	Code versioning system used	git		
C6	Software code languages, tools, and	python		
	services used			
C7	Compilation requirements, operat-	indicated in requirements.txt		
	ing environments and dependencies			
C8	If available, link to developer docu-	https://github.com/		
	mentation/manual	swat-lab-optimization/		
		AmbieGen-tool/blob/master/		
		README.md		
C9	Support email for questions	dmytro.humeniuk@polymtl.ca		

Table 1: Code metadata (mandatory)

exploration, and aid in rescue operations. At the same time, these are safetycritical systems and it is very important to ensure they are robust to unseen environments and conditions. This can be done by thorough testing prior to their deployment. Typically, at the initial development stages model-inthe-loop testing is performed [1], where the system is tested in a simulation environment. Given the complexity of autonomous systems, the number of potential test scenarios is vast and exhaustive execution is not feasible. For example, an autonomous vehicle scenario could involve a variety of parameters such as road topology, the movement and behavior of other vehicles and pedestrians, traffic signs, weather conditions, etc. We surmise that in order to identify the most critical scenarios for a given system, application of search algorithms is necessary.

In this work, we propose AmbieGen, a search based framework for generating adversarial test scenarios for autonomous systems. By leveraging evolutionary search AmbieGen allows to find challenging and diverse test scenarios. The problem of identifying critical scenarios for a system has been addressed in several previous works on falsifying temporal logic requirements of cyber-physical systems, such as S-Taliro [2], Breach [3], and ARISTEO [4]. These works typically consider falsifying a model of the system that takes a set of input signals and produces a set of output signals.

In our work, we focus on testing autonomous systems for which the input signals are complex and may include data from various sensors and cameras. Generating a valid combination of falsifying input signals (such as lidar readings and RGB camera readings) directly would be challenging. Therefore, we propose a method for generating test cases that specify a virtual environment for the autonomous system, rather than the input signals. The input signals are generated in the virtual environment during simulation based on the actions of the autonomous agent.

Several approaches have been proposed for generating virtual environments for testing autonomous driving and robotics systems, including As-Fault [5], Frenetic [6], DeepJanus [7], DeepHyperion [8] and others presented at the SBST 2021 [9] and SBST 2022 [10] tool competitions.

The tool we present in this paper, AmbieGen, is the winner of SBST 2022 tool competition. It could produce the biggest number of diverse fault revealing scenarios for an autonomous vehicle lane keeping assist system (LKAS) given a limited time budget. More details about the search algorithm implementation can be found in our research paper [11]. In our work we have shown that the simplified model of the system can be effective in guiding the search for producing the test scenarios for the full, simulator based, model of the system.

Our framework can be used for further research in the search algorithms, search operator and fitness function design for autonomous systems adversarial testing. We built the framework to be modular, and thus easily customizable. By referring to project documentation as well as the example implementations we provide, researchers can specify their own test scenario generation problems, fitness functions, crossover and mutation operators. This tool is developed in Python and can be easily run as a python package. More instructions and examples are provided in the AmbieGen repository.

2. Software description

In this work, we present AmbieGen, an open-source Python framework that utilizes evolutionary search for the generation of test scenarios for autonomous systems. Currently, AmbieGen supports the creation of test scenarios for lane keeping assist systems (LKAS) in autonomous vehicles and for autonomous robots navigating a closed room with obstacles.

The test scenarios for LKAS in vehicles are designed to challenge the system with various road topologies, while the scenarios for autonomous robots involve navigating a closed room with obstacles. Examples of the generated scenarios can be seen in Figure 1.



Figure 1: An example of the test case for LKAS system (a) and an autonomous robot (b). The x-axis represents the map length in meters, and the y-axis represents the map width in meters.

2.1. Software architecture

This subsection provides a detailed description of the software implementation of AmbieGen. The key components of AmbieGen are illustrated in Figure 2, which are common components for implementing evolutionary search. We use the Pymoo framework [12] to implement the search algorithms. The most important modules and classes are outlined below:

• Solution - this is one of the most important classes, which contains all the necessary attributes and functions needed to represent the candidate solution of the algorithm. It should contain a *scenario* attribute with the list of test case parameters, function for fitness evaluation, novelty calculation, as well as, optionally, image generation.



Figure 2: AmbieGen architecture

- Sampling this is the class for initial population generation. At the output it provides N instances of the Solution class, with the initialized scenario attribute, defining the test scenario. Typically the test scenario is represented by a two dimensional array, randomly initialized based on the minimum and maximum values of the test case parameters, defined in the configuration file. Each column of the array corresponds to some part of the environment. More information about the representation of the test scenarios that we used can be found in the repository page as well as in our research article.
- *Problem* in this class, we define the logic for evaluating the fitness of each solution. For single-objective search (using GA), we specify the fitness function for evaluating the scenario fault revealing power. For two-objective search (using NSGA-II), we define two objectives: fault revealing power and novelty calculation. The novelty objective is calculated as the average novelty of a given test scenario relative to the 5 solutions with the highest fault revealing power fitness. If the problem has any constraints, such as a minimum required fitness value, they should also be specified in this class.
- *TC to environment* this is a function to transform the test case (TC) encoded as a 2D array of parameters, to the input format suitable for the system model. For example, for the LKAS problem, the model input is a list of the 2D coordinates of points, defining the road topology. The test case itself is represented as a sequence of transformations

needed to perform to obtain the points. For the autonomous robot the test scenario is represented as a sequence of parameters describing the 2D map with obstacles. The TC to environment module is used to create a 2D bitmap from the given parameters. The bitmap is given as the input to the autonomous robot model, which runs a planning algorithm to find the shortest path between the start and goal location.

- *fitness evaluation* a function to calculate the fitness i.e fault revealing power of the scenario. It takes the output of the *TC to environment* function as the input and execute the system model. It collects the data about the model behaviour during execution and computes the fitness score. For the LKAS system, the fitness is defined by the biggest deviation from the lane center and for the autonomous robot by the length of the path to reach the goal.
- *Crossover* in this class the crossover operator is defined. Currently we are using a one point crossover, which can be applied to fixed and variable length solutions.
- *Mutation* in this class the mutation operator is implemented. We have 2 types of mutations: exchange and change of variable. In exchange mutation, two randomly selected columns of the test case are exchanged. In the case of the road topology, it would correspond to exchanging the positions of two random road segments. In change of variable mutation, a randomly selected parameter value in the test case matrix is changed. In the road topology example it could correspond to the change of the length of one of the straight road segments.
- post processing The post-processing module of our framework includes several functions for handling the test suite and its metadata. The function $get_test_suite()$ retrieves the test suite, $get_stats()$ retrieves metadata such as fitness and novelty scores, and $save_tcs_images()$ saves the images of the test cases. The size of the test suite, denoted as T, can be specified in the configuration file. In our experiments, Twas typically set to 30, representing the best solutions found by the algorithm.

Metadata for the test suite includes the fitness of the top T solutions, their novelty (calculated as the average novelty between all pairs of scenarios in the test suite), and the convergence (best solution fitness found at each epoch). The post-processing module also includes a *compare.py* script for comparing the results of different algorithms, using the collected metadata to generate convergence plots and fitness and diversity boxplots.

• configuration file - finally we have a configuration file, where the parameters of the algorithm, such as: the population size, the number of generations, crossover/mutation rate, and the test suite size are defined. Users should also specify the allowable ranges for the test case parameters and the paths for saving the resulting test suite and its metadata.

Currently, when adding a new problem, one should provide the implementation of each of the modules as well as the TC to environment and fitness evaluation functions. We are working on reducing the number of additional implementations needed. Our framework includes the implementation of all the modules for the LKAS and autonomous robot test case generation problems.

2.2. Software functionalities

AmbieGen public version 0.1.0 as presented in this paper offers the following major functionalities:

- Autonomous vehicle LKAS system testing: generating scenarios, represented as a list of 2D coordinates defining the road topology.
- Autonomous robot testing: generating scenarios, represented as the 2D bitmap, defining obstacle locations in a fixed sized map.
- Search-based generation: our framework provides options for searchbased test suite generation, including random search, single-objective genetic algorithm (GA), and two-objective genetic algorithm (NSGA-II). The search algorithms are implemented using the Pymoo framework [12], and can be easily extended to support additional algorithms supported by Pymoo with minor modifications.

The single-objective GA optimizes the test suite for scenario fault revealing power, while the two-objective NSGA-II optimizes for both fault revealing power and diversity. As demonstrated in our previous work [11], the two-objective algorithm allows to produce a more diverse set of test scenarios compared to the single-objective search. • Experiment data tracking: AmbieGen tracks the results of each experiment and saves them in a user-defined location. The saved data includes the T (as determined by the user) best test scenarios identified based on their fitness or crowding distance, as well as their associated metadata such as fitness, average diversity, and visualizations. This allows for easy analysis and comparison of the results of different experiments.

2.3. Use cases of the software

In this subsection we provide an illustrative example of how to use AmbieGen to generate test cases for an autonomous robot planning algorithm testing. Suppose we want to perform 30 runs of the NSGA-II algorithm with 150 individuals and 200 generations to evaluate this configuration. We want to save the generated test cases, their illustrations as well as their metadata, such as fitness and diversity. Below you can see the configuration file entries with the parameters we chose for the genetic algorithm and well as the path to save the experiment results:

```
ga = {"pop_size": 150, "n_gen": 200, "mut_rate": 0.4, "cross_rate": 0.9,
"test_suite_size": 30 }
files = {"stats_path": "stats", "tcs_path": "tcs", "images_path": images"}
```

Now we are ready to start the test case generation. We can launch AmbieGen with the following command and parameters:

```
python optimize.py --problem="robot" --algo="nsga2" --runs=30 \\
--save_results=True
```

The search will start and you could see some printouts, such as in Fig. 3 with the current number of generation (n_gen) , number of evaluations (n_eval) , constraint violation (cv_min) , number of non-dominant solution for NSGA-II algorithm (n_nds) and the best solution found (f_opt) for GA algorithm. More details about the printed information can be found on the Pymoo page (https://pymoo.org/interface/display.html).

After a successful run, you will see the confirmation about the run execution time, saved test cases, their metadata and the images, as in Fig. 4

In Fig. 5 you can see examples of the metadata saved, such as the algorithm convergence 5a (the best fitness value at each generation in the format "evaluation number": best fitness found), the fitness of the test cases in the test suite as well as their average diversity i.e., novelty 5b. Novelty is calculated as the average diversity of all of the pairs of the test cases in the

2022-12- 2022-12- 2022-12- 2022-12- 2022-12- 2022-12-	0 0 0 0 0 0 5 0	2,320 INFO 2,320 INFO 2,321 INFO 2,343 INFO 2,344 INFO	Started te Running th Problem: ro Executing Using rand	st generation, w e optimization obot, Algorithm: run 0: om seed: 1753925	vriting logs to fi nsga2, Runs numb 1990	le: logs.txt er: 30, Saving	the results: True
n_gen	n_eval	n_nds	cv_min	cv_avg	eps	indicator	
1	150	1	5.474517E+01	8.330072E+01	-		
2	300	1	4.684567E+01	7.742613E+01			
3	450	1	4.436039E+01	7.231653E+01			
4	600	1	3.167410E+01	6.692135E+01			
5	750	1	7.8751083190	6.161694E+01			

Figure 3: Printouts during the search

3,072 INFO	Execution time, 6909.677314 sec
,088 INFO	Test suite of 30 test scenarios generated
103 INFO	The highest fitness found: 224.994949
103 INFO	Average diversity: 0.720751
25,148 INFO	Stats saved as stats_nsga2\31-12-2022-stats.json
2022-12-11 01-21:25,157 INFO	Stats saved as stats_nsga2\31-12-2022-conv.json
25,361 INFO	Test cases saved as tcs_nsga2\31-12-2022-tcs.json
53,871 INFO	Images saved in tc_images_nsga2
53,871 INFO	Images saved in tc_images_nsga2

Figure 4: Successful run confirmation

test suite. In Fig. 6 we show an example of the test case images saved for a particular run.

'run0": {	"run0": {		
"150": 97.01219330881972,	"fitness": [
"200": 99.59797974644661,	198.7106781186548,		
"250": 99.59797974644661,	171.0538238691624,		
"300": 106.18376618407352,	192.02438661763966,		
"350": 106.18376618407352,	194.46803743153552,		
"400": 106.18376618407352,	190.36753236814718,		
"450": 107.25483399593897.	211.88225099390866,		
"500": 107.25483399593897.	209.88225099390866,		
"550": 130, 56854249492375.	194.85281374238588,		
"600": 130,56854249492375,	168.71067811865476,		
"650": 130 56854249492375	191.8822509939086,		
"700": 130 56854240402375	181.15432893255073,		
"760". 130.50834243432375,	183.39696961967007		
"900", 130,50054249492575,],		
800 . 150.50854249492575, "850", 450 56954249492575	"novelty": 0.23096571372433472		
850 : 130.50854249492375,	},		
900 : 130.56854249492375,	"run1": {		

Figure 5: Metadata for the generated scenarios

Finally, let us suppose we also want to run a random search with the same evaluation budget to be able to compare the performance of our configuration of NSGA-II algorithm to some baseline. We can run the random search by



Figure 6: Images of the generated scenarios

executing the following command:

```
python optimize.py --problem="robot" --algo="random" --runs=30 \\
--save_results=True
```

The random search will be run and the metadata saved, as in the previous case. Now we can compare the results produced by the two different search algorithms via executing the following command:

python compare.py --stats_path="stats_nsga2" "stats_random" \\
--stats_names "NSGA-II" "Random"

In the *stats_path* argument we specify the paths of the metadata for the runs we wish to compare and in the *stats_names* the names we assign for the runs.

In Fig.7 and Fig. 8 we can see examples of the outputs produced by the *compare.py* script. Fig. 7a shows the fitness and Fig. 7b the diversity of the scenarios in the test suites produced over the specified number of runs. Fig. 8 shows the best values found by the compared search algorithms over the generations.

3. Illustrative examples

In this section, we present the summarized results of several test generation case studies using the AmbieGen tool. The full results can be found in our research paper [11] and the SBST 2022 competition report [10].

We conducted a case study on an autonomous robot with an obstacle avoidance algorithm based on nearness diagrams [13]. The robot model was a Pioneer 3-AT equipped with a SICK LMS200 laser with a sensing range of 10 meters. The simulations were run in the Player/Stage simulator [14].



Figure 7: Evaluating the NSGA-II algorithm for autonomous robot test case generation



Figure 8: Comparing the convergence of NSGA-II and random search for autonomous robot case study

You can see an illustration of the simulation environment in Fig. 9a. We used AmbieGen to generate diverse maps with obstacles to test the robot's performance. We identified several scenarios in which the robot became stuck and failed to reach its goal location. An example of such a scenario can be found in the following video: Video.

To evaluate the effectiveness of our tool, we allocated a two-hour budget for AmbieGen to generate test scenarios. The generated scenarios were then passed to the simulator and executed. We repeated the experiment 30 times, using both the NSGA-II and random search configurations of AmbieGen. The average number of failures detected is shown in Fig. 9b. On average, AmbieGen detected 9 failures in two hours, compared to 2 failures for random search



(a) Executing autonomous robot scenario in the Play- (b) The number of failures revealed by AmbieGen for er/Stage simulator the robot case study

Figure 9: Using AmbieGen for testing autonomous robot navigation algorithm

In the second case study, we evaluated the performance of our test generation tool on an autonomous vehicle lane keeping assist system (LKAS) using the BeamNg simulator [15]. We used the AmbieGen tool to generate diverse, fault-revealing road topologies, which were then simulated in the BeamNg environment (shown in Fig. 10a). During the simulations, we identified a number of scenarios in which the vehicle left its lane (an example of which can be seen in the video at Video).

We ran our tool for a time budget of 2 hours, using the SBST22 competition code pipeline. The failure criterion for the LKAS system was defined as more than 85% of the car's area leaving the lane. The driving agent had a maximum speed of 70 Km/h. We compared the results of AmbieGen's NSGA-II configuration, Random Search configuration, and the Frenetic tool [6], which was also given a 2-hour time budget for test generation.

As shown in Fig. 10b, out of 30 runs, AmbieGen and Frenetic on average produced almost the same number of failures (14), while Random Search produced an average of 9 failures.

The obtained results suggest that AmbieGen could effectively identify failures in the autonomous systems under test.

4. Impact

Autonomous systems testing is an important area of research, and finding test scenarios that reveal a diverse range of system failures within a limited



(a) Executing the LKAS scenario in the BeamNg simulator (b) The number of failures revealed by AmbieGen for the LKAS case study

Figure 10: Using AmbieGen to test autonomous vehicle LKAS model

time and evaluation budget is a significant challenge [16]. One of the common solutions is to use evolutionary search to guide the sampling towards more challenging scenarios [5, 7]. These search based techniques allow to identify potential failures and improve the overall reliability of the system.

AmbieGen is a test generation tool that uses evolutionary search to generate test scenarios for autonomous systems. Its modular design allows for customization of the initial population generation function, fitness evaluation function, search operators (such as crossover and mutation), and the search algorithm itself. Out of the box, AmbieGen supports testing of autonomous robots and vehicle LKAS systems, and additional systems can be added using the provided implementations as examples.

AmbieGen is a valuable resource for research on search-based test case generation for autonomous systems. Its built-in modules enable easy comparison of different search algorithms and their modifications, based on the quality and diversity of the generated solutions, as well as the convergence of the algorithm over time.

AmbieGen can help answer research questions that are not frequently discussed in the literature, such as:

- To what extent the diversity preservation technique A helps improve the diversity of the test suite? The importance of the diversity in test case generation is extensively discussed in the work of Klikovits et al. [17].
- To what extent does the search operator A helps improve the convergence over the operator B? To what extent the algorithm A outperforms

the algorithm B for the test case generation? Improvements to the baseline genetic algorithms implementations can lead to better results, as discussed by Abdessalem et al. [18], where multi-objective populationbased search algorithms and decision tree classification were combined.

• What fitness criteria are more relevant for guiding the system towards fault revealing scenarios? This question includes the comparison of the single, multi-objective based search as well surrogate model assisted search.

AmbieGen can also be useful in the pursuit of actively studied research questions, where the fault revealing test case generation is required, such as: transferability of failures from simulation to the real world [19], autonomous system failure prediction [20], test case prioritization [21] and others.

AmbieGen has proven its effectiveness in fault revealing by winning this year's edition of the SBST 2022 cyber-physical testing tool competition. Our submission is described in the following article [22] and is available at the following link https://github.com/dgumenyuk/tool-competition-av. We have always kept our tool open sourced and we expect more people to start using it. We welcome all the contributions for expanding our framework.

5. Conclusions

In this paper, we present the AmbieGen framework for search based test case generation for autonomous systems, in its public version 0.1.0. We briefly outline the motivation for developing this framework, its workflow and main functionalities. We also provide illustrative examples for using the tool for autonomous vehicle lane keeping assist system testing and autonomous robot obstacle avoiding algorithm testing. The main features of our tool include:

- modular architecture, which allows researchers to easily modify the existing modules, such as initial population generation, crossover, mutation, fitness function as well as introduce new problems and run experiments;
- we provide implementations of test case generation for two systems under test: autonomous vehicle LKAS system and autonomous robot; this implementation includes three search algorithms: random search,

single objective genetic algorithm and a two-objective NSGA-II genetic algorithm;

• our framework is built to be compatible with Pymoo framework [12], allowing to fully benefit from the Pymoo framework features, such as high number of implemented algorithms in Pymoo.

6. Future Plans

Our framework currently includes the implementation of two test case generation problems, as well as three algorithms (random search, GA, NSGA-II) for generating test cases. The fitness function is calculated based on a simplified model of the system, and test scenarios are represented as 2D arrays, with each column describing a discrete aspect of the scenario. In the future, we plan to expand the capabilities of our framework to include:

- new algorithms, especially the ones based on the quality-diversity search [23]
- new test case generation problems, for instance more complex test scenarios that include moving pedestrians, other vehicles and traffic signs;
- new fitness functions e.g based on surrogate models of the system under test, as in the work of Ramakrishna et al. [24], functions based on neuron coverage [25] and surprise adequacy [26] dedicated to testing systems containing neural networks;
- add new problem representations, supporting popular scenario specification languages such as SCENIC [27];
- add an integration with popular simulators, for instance CARLA [28] or LGSVL [29]. This will allow to directly evaluate the system model with the generated scenarios. Also the feedback from the simulators could be incorporated in fitness functions for guiding the test scenario sampling.

Acknowledgements

This work is partly funded by the by the Fonds de Recherche du Québec (FRQ), the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Canadian Institute for Advanced Research (CIFAR).

References

- D. Bruggner, A. Hegde, F. S. Acerbo, D. Gulati, T. D. Son, Model in the loop testing and validation of embedded autonomous driving algorithms, in: 2021 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2021, pp. 136–141.
- [2] Y. Annpureddy, C. Liu, G. Fainekos, S. Sankaranarayanan, S-taliro: A tool for temporal logic falsification for hybrid systems, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2011, pp. 254–257.
- [3] A. Donzé, Breach, a toolbox for verification and parameter synthesis of hybrid systems, in: International Conference on Computer Aided Verification, Springer, 2010, pp. 167–170.
- [4] C. Menghi, S. Nejati, L. Briand, Y. I. Parache, Approximationrefinement testing of compute-intensive cyber-physical models: An approach based on system identification, in: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), IEEE, 2020, pp. 372–384.
- [5] A. Gambi, M. Mueller, G. Fraser, Automatically testing self-driving cars with search-based procedural content generation, in: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, IEEE, 2019, pp. 318–328.
- [6] E. Castellano, A. Cetinkaya, C. H. Thanh, S. Klikovits, X. Zhang, P. Arcaini, Frenetic at the sbst 2021 tool competition, in: 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), IEEE, 2021, pp. 36–37.
- [7] V. Riccio, P. Tonella, Model-based exploration of the frontier of behaviours for deep learning system testing, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 876– 888.
- [8] T. Zohdinasab, V. Riccio, A. Gambi, P. Tonella, Deephyperion: exploring the feature space of deep learning-based systems through illumina-

tion search, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 79–90.

- [9] S. Panichella, A. Gambi, F. Zampetti, V. Riccio, Sbst tool competition 2021, in: 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), IEEE, 2021, pp. 20–27.
- [10] A. Gambi, G. Jahangirova, V. Riccio, F. Zampetti, Sbst tool competition 2022, in: 2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST), IEEE, 2022, pp. 25–32.
- [11] D. Humeniuk, F. Khomh, G. Antoniol, A search-based framework for automatic generation of testing environments for cyber-physical systems, Information and Software Technology 149 (2022) 106936. doi: https://doi.org/10.1016/j.infsof.2022.106936.
- [12] J. Blank, K. Deb, pymoo: Multi-objective optimization in python, IEEE Access 8 (2020) 89497–89509.
- [13] J. Minguez, L. Montano, Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios, IEEE Transactions on Robotics and Automation 20 (1) (2004) 45–59.
- [14] M. Kranz, R. B. Rusu, A. Maldonado, M. Beetz, A. Schmidt, A player/stage system for context-aware intelligent environments, Proceedings of UbiSys 6 (8) (2006) 17–21.
- [15] BeamNG.tech, Beamng gmbh. (2021). URL https://www.beamng.gmbh/research
- [16] W. Ding, C. Xu, M. Arief, H. Lin, B. Li, D. Zhao, A survey on safety-critical driving scenario generation – a methodological perspective (2022). doi:10.48550/ARXIV.2202.02215. URL https://arxiv.org/abs/2202.02215
- [17] S. Klikovits, V. Riccio, E. Castellano, A. Cetinkaya, A. Gambi, P. Arcaini, Does road diversity really matter in testing automated driving systems?-a registered report, arXiv preprint arXiv:2209.05947 (2022).
- [18] R. B. Abdessalem, S. Nejati, L. C. Briand, T. Stifter, Testing visionbased control systems using learnable evolutionary algorithms, in: 2018

IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE, 2018, pp. 1016–1026.

- [19] A. Stocco, B. Pulfer, P. Tonella, Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems, IEEE Transactions on Software Engineering (2022).
- [20] A. Stocco, P. Tonella, Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems, Journal of Software: Evolution and Process (2021) e2386.
- [21] A. Arrieta, P. Valle, J. A. Agirre, G. Sagardui, Some seeds are strong: Seeding strategies for search-based test case selection, ACM Transactions on Software Engineering and Methodology (2022).
- [22] D. Humeniuk, G. Antoniol, F. Khomh, Ambiegen tool at the sbst 2022 tool competition, in: 2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST), IEEE, 2022, pp. 43–46.
- [23] J. K. Pugh, L. B. Soros, K. O. Stanley, Quality diversity: A new frontier for evolutionary computation, Frontiers in Robotics and AI (2016) 40.
- [24] S. Ramakrishna, B. Luo, Y. Barve, G. Karsai, A. Dubey, Risk-aware scene sampling for dynamic assurance of autonomous systems, in: 2022 IEEE International Conference on Assured Autonomy (ICAA), IEEE, 2022, pp. 107–116.
- [25] K. Pei, Y. Cao, J. Yang, S. Jana, Deepxplore: Automated whitebox testing of deep learning systems, in: proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 1–18.
- [26] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1039–1049.
- [27] D. J. Fremont, E. Kim, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, S. A. Seshia, Scenic: A language for scenario specification and data generation, Machine Learning (2022) 1–45.
- [28] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, Carla: An open urban driving simulator, in: Conference on robot learning, PMLR, 2017, pp. 1–16.

[29] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, et al., Lgsvl simulator: A high fidelity simulator for autonomous driving, in: 2020 IEEE 23rd International conference on intelligent transportation systems (ITSC), IEEE, 2020, pp. 1–6.