2022 IEEE Intelligent Vehicles Symposium (IV)
June 5-9, 2022. Aachen, Germany

Stress Testing Autonomous Racing Overtake Maneuvers with RRT

Stanley Bak[†*], Johannes Betz[‡*], Abhinav Chawla[†*], Hongrui Zheng[‡*], and Rahul Mangharam[‡]

2022 IEEE Intelligent Vehicles Symposium (IV) | 978-1-6654-8821-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IV51971.2022.9827237

***Abstract*— High-performance autonomy often must operate at the boundaries of safety. When external agents are present in a system, the process of ensuring safety without sacrificing performance becomes extremely difficult. In this paper we present an approach to stress test such systems based on the rapidly exploring random tree (RRT) algorithm.**

We propose to find faults in such systems through *adversarial agent perturbations*, where the behaviors of other agents in an otherwise fixed scenario are modified. This creates a large search space of possibilities, which we explore both randomly and with a focused strategy that runs RRT in a bounded projection of the observable states that we call the *objective space*. The approach is applied to generate tests for evaluating overtaking logic and path planning algorithms in autonomous racing, where the vehicles are driving at high speed in an adversarial environment. We evaluate several autonomous racing path planners, finding numerous collisions during overtake maneuvers in all planners. The focused RRT search finds several times more crashes than the random strategy, and, for certain planners, tens to hundreds of times more crashes in the second half of the track.

I. Introduction

Controlling autonomous systems near their performance boundaries and in the presence of adversarial agents is hard. Consider the Indy Autonomous Challenge [1][2], a full-size autonomous racing competition with over 30 university teams originally announced in 2019 and scheduled to take place in October 2021. In this environment, vehicles are driving at high speeds with high accelerations in an adversarial environment. Overtaking maneuvers like the one shown in Figure 1 are difficult to plan because the safety of the maneuver depends on the future actions of the opponent, the interaction between the vehicles, different topological constraints at each track segment and the correct observance of the vehicle dynamic limits. Even with the large incentives to create safe and high performance autonomy—including a $1.5 million prize pool—there are strong reasons to be suspicious of the safety of the autonomous logic during overtake. In the most recent simulated race in June 2021[1] [3], in the single-car qualification stage, most vehicles performed extremely well—the lap time for the top 14 of the 16 cars was within half a second. However, during multi-car races, 12 of the 16 competing teams were disqualified for causing crashes with other vehicles or the wall.

[†]Stony Brook University, Department of Computer Science, 11794 Stony Brook, NY, USA stanley.bak, abhinav.chawla@stonybrook.edu

[‡]University of Pennsylvania, School of Engineering and Applied Science, 19106 Philadelphia, PA, USA joebetz, hongruiz, rahulm@seas.upenn.edu

[*]Authors contributed equally.

[1] https://www.youtube.com/watch?v=gTjQ3sWdYh0

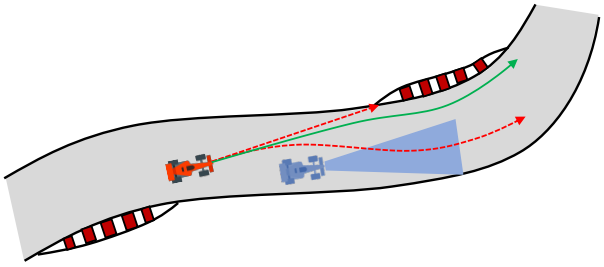

Fig. 1: Illustrative overview of autonomous path planning on the racetrack for two vehicles. While considering the possible motion of the opponent car (blue area), the red trajectories show infeasible solutions, the green trajectory leads to a successful overtaking maneuver.

Part of the reason for the difficulty of creating safe overtake logic is that traditional end-to-end scenario testing is inefficient. The number of situations where the logic must perform well is vast, but only a small number of interactions are exposed during each long simulation run of a single scenario. In this paper, we strive to address this shortcoming by developing targeted test-generation approaches. Rather than testing a large number of scenarios (starting car positions, different racetracks, number of opponents), we instead explore the large number of possibilities within a single scenario. We focus on detecting a wide variety of crashes during autonomous overtaking maneuvers.

To do this, we perform *adversarial agent perturbations*, slightly modifying the opponent vehicle behaviors at different points in time. For example, an opponent vehicle can be altered to drive slightly faster or slightly slower. Repeatedly performing small perturbations creates a large tree of possible simulations, which we then explore to look for collisions. To do this without duplicating simulation effort, our search strategy is based on rapidly exploring random trees (RRT) [4]. Further, we target the search on a projection of the full simulation state space where overtake maneuvers are likely to occur, which we call the *objective space*.

We evaluate our approach using quantitative autonomy coverage metrics based on spatial clustering approaches. For five different planning algorithms, we show that our approach has superior performance compared with random exploration of the adversarial perturbation search tree.

The main contributions of the paper are:

- The use of adversarial agent perturbations to create a tree of possible behaviors to explore;
- The efficient searching through the possible behaviors using a random and more efficient RRT-based strategy;
- The evaluation of the performance of the autonomy through quantitative failure coverage metrics;

- The application of our framework to five planners in the autonomous racing and overtake scenario.

The rest of this paper is organized as follows. First we review and compare our approach with related test-generation methods in Section II. Next, in Section III, we present our methodology as a generic test-generation algorithm that could be used for any autonomous system with adversaries. In Section IV, we specialize our generic approach to the concrete problem of overtaking in autonomous racing, using an open-source simulator [5] for the F1TENTH autonomous racing competition[2]. The paper finishes with a discussion in Section V followed by a conclusion.

II. Related Work

The line of work closest to our approach is sampling-based testing and verification. RRT has been considered for test generation before [6]. A new distance function, new weighting, and a scheme to modify sampling probability distribution on the fly were introduced, reducing computation time by an order of magnitude. However, the method samples in the full state-space of the system, and does not consider adversarial multi-agent systems that have unknown dynamics. In contrast, since we search a projection of the state space we call the objective space, our method is compatible with black-box systems where only the environment and agent actions are observable. Nonetheless, we could consider adapting some of their proposed optimizations to improve our search, such as using gradient information to improve sample selection.

Other work uses sampling-based verification algorithms to falsify temporal logic specifications of the system [7]. This method focuses on resolution completeness of the falsification problem while we focus on finding a large variety of failure cases. Other falsification approaches for hybrid systems [8], [9], [10] try to minimize robustness metrics for temporal logics. Robustness metrics capture how close a simulation is to an unsafe state. This would not work well for autonomous racing collisions, as high-performance controllers typically drive close to track boundaries where robustness would be low. These methods also generally compute full rollouts which may be slow, and work best when the input dimension of the system is modest. Partial rollout test generation has also been considered [11], [12], using the notion of trajectory splicing. For these methods, an abstraction of the system is constructed based on short simulation runs. When problems are found, it can be difficult to concretize the abstract counterexample, to connect the trajectory pieces to create an actual error witness.

Data-driven verification procedures have also been used for test generation [13], which use SVM to classify whether trajectories will be in the safe or unsafe set. In addition, this work uses an active learning scheme to request additional sample points to update the model. In comparison, our method aims to find many different failure cases by sampling short rollouts of the simulation. We also do not suffer from generalization issues since we do not need to create a predictive model to classify the safety of system trajectories.

[2] https://f1tenth.org

Reachability analysis has also been applied to autonomous vehicles as an attempt to compute all possible future scenarios in order to verify the system's safety [14], [15], [16], [17]. This line of work requires accurate symbolic models of the target system and sometimes other agents in the same environment. Although some of the work can incorporate uncertainty, the problem often becomes intractable for complex dynamics or suffers from excessive error. For this reason, applications of reachability have generally been limited to structured environments like everyday traffic.

Testing scenario search and generation has also been widely applied to the testing and verification of autonomous systems [18], [19], [20]. Some approaches employ optimization or adaptive sampling techniques to accelerate finding test cases with highest risk to the system [21], [22], [23], [24], [25], [26]. In comparison to this line of work, our approach operates in less structured driving conditions, and we focus on finding a large variety of failures rather than rare corner cases. Our approach is compatible with scenario-based testing, as we focus on exploring the possibilities within each individual scenario (intra-scenario testing).

Lastly, in software fuzz testing, test generation can be classified into white-box, gray-box, and black-box methods, depending on whether the inner workings of the software is known to the tester [27], [28], [29]. Our application is different, in that the planner is reactive, tightly interacting with a physical environment simulator. In this classification, our approach is hybrid of black-box and gray-box methods—we treat both the planner logic and environment as a black box, but use their interface to drive the search process.

III. Methodology

We now present the test generation problem and solution framework we propose. In this section, we describe it as a general approach that we could use to create tests for any autonomous system with adversarial agents. In the next section, we will apply this generic method to the concrete problem of overtake in autonomous racing.

A. Problem Statement

Definition 1: We describe the model of the system under test as a tuple $\mathcal{S} = (\mathcal{X}, \mathcal{T}, \mathcal{U})$ where

- $\mathcal{X}$ is a set of observable states of the system
- $\mathcal{T}$ is the corresponding time steps to the states $\mathcal{X}$
- $\mathcal{U}$ is a finite set of external inputs (adversarial perturbations) possible at each step in $\mathcal{T}$.

Definition 2: We define the testing problem as: given a tuple $(\mathcal{E}, \mathcal{S}, \mathcal{T}_{[i,e]}, \mathcal{V}, \mathcal{F})$ where

- $\mathcal{E}$ is the selected scenario where the system operates in
- $\mathcal{S}$ is system under test
- $\mathcal{T}_{[i,e]}$ is a discrete interval of testing time steps
- $\mathcal{V}$ is the finite set of adversarial agent perturbations
- $\mathcal{F}$ is the failure specification set.

The goal of testing is to find *failing tests*. A failing test maps adversarial agent perturbations from each time step in $\mathcal{T}_{[i,e]}$ to $\mathcal{V}$, such that $\exists t \in \mathcal{T}_{[i,e]}$ where $\mathcal{X}(t) \in \mathcal{F}$. We are interested to both maximize the number of failing tests, as

we well as to explore a variety of failures (defined later using spatial clustering methods).

Specifically in our case, the system under test involves path planning algorithms and controllers for autonomous race cars. The observable states of the system $\mathcal{X}$ consist of the pose (x, y, θ) of the ego vehicle in the world, the current laser scan of the ego vehicle, and the poses of other vehicles in the world. The unobservable states of the system consists of the decision making logic and control rules used by the motion planner, as well as any internal environmental states not exposed to the planners. The adversarial perturbation inputs $\mathcal{U}$ are modifications to the command outputs of the motion planners according to the observable states. The failure specification set $\mathcal{F}$ consists of all states where the ego vehicle is in collision with either the race track or other agents.

B. Stress Testing Pipeline

Our test generation method fits into a larger autonomy design and verification process as shown in Figure 2.

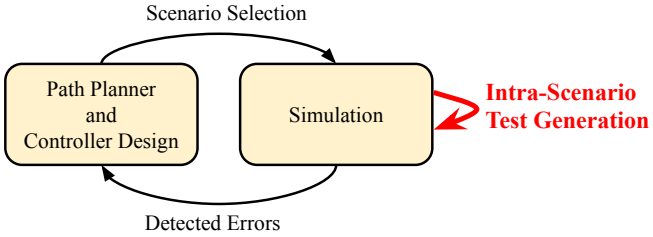

Fig. 2: This work focuses on one part of the autonomous systems testing process, intra-scenario test generation (red).

Although we do not explore scenario selection in this work, scenario selection (inter-scenario testing) and our proposed algorithm (intra-scenario testing) complement each other. After the development of an initial version of the path planner and controller, a test engineer (or algorithm) selects an appropriate test scenario which includes the initial states and environmental conditions. Our method then stress-tests this scenario by finding errors under slightly different behaviors of each of the adversarial agents (red arrow). Lastly, the found failure cases provide feedback on the performance of the system, and can be used by the test engineer (or an algorithm) to either select new scenarios and collect further failures or to tune parameters in the path planner.

C. Intra-Scenario Test Generation Details

We now describe the details of the test generation process. The algorithm uses the following application-specific inputs:

- **Initial Scenario Configuration**: $q_{init} \in \mathbb{R}^n$, initial observed state to initialize the environment simulator;
- **Adversarial Agent Perturbations**: finite set $\mathcal{V}$ of size m, describes perturbations that could be made to the adversarial agents at each step;
- **Objective Space Projection Function**: $\mathcal{P} : \mathbb{R}^n \to \mathbb{R}^o$ maps observed state to the RRT search space;
- **Objective Space Exploration Limits**: $\mathcal{B} \in \mathbb{R}^{2o}$ describes the limits for sampling the objective space;
- **Simulation Step Function**: `step_sim`, computes a short-time rollout of the simulation environment given a start state and adversarial agent perturbation.

Algorithm 1 RRT Based Algorithm

Output: tree of explored states G

1: $p_{init} \leftarrow \mathcal{P}(q_{init})$
2: G.`init`(p_{init})
3: **while** not finished **do**
4: $p_{rand} \leftarrow$`rand_sample(`$\mathcal{B}$`)`
5: $p_{nearest} \leftarrow$`nearest(`p_{rand}, G, $\mathcal{B}$`)`
6: $q_{nearest} \leftarrow \mathcal{P}^{-1}(p_{nearest})$
7: **for** adv in $\mathcal{V}$ **do**
8: $q_{child} =$ `step_sim(`$q_{nearest}$, adv`)`
9: $p_{child} \leftarrow \mathcal{P}(q_{child})$
10: G.`add_vertex(`p_{child}`)`
11: G.`add_edge(`$p_{nearest}$, p_{child}`)`
12: **end for**
13: **end while**

With these, we perform RRT-based exploration to detect failing tests, shown in Algorithm 1. At line 1, the search is initialized by taking the start state of the scenario q_{init} and projecting it to the objective space. This serves as the root node of the tree, and the tree of explored states is initialized with the root node at line 2. At lines 4 and 5, the algorithm then proceeds like RRT in the objective space, generating random o-dimensional points subject to the limits $\mathcal{B}$, finding the closest node within $\mathcal{B}$ using a normalized L_2-norm distance metric. And at line 6, the state corresponding to the closest node is found. In lines 7 to 11, all adversarial perturbations $\mathcal{V}$ are then explored from this node using short rollouts performed by the `step_sim` function, which also takes in a specific adversarial perturbation to use. The `step_sim` function is custom to the scenario being considered, and we elaborate on it further in the next section in the context of autonomous racing. The resultant states are projected to the objective space using $\mathcal{P}$ and added to the search tree G. We generally use stopping conditions that cap the number of explored nodes, which corresponds to the number of steps the simulation environment has executed, although a timeout could also be used. Every node created in the process also stores all information necessary to recreate the simulation starting at the initial state, guaranteeing all states explored (and failures found) can be recreated. The algorithm returns the m-ary tree of explored nodes G, which can then be quickly checked to provide the lists of perturbations that led to failures using the specification failure set $\mathcal{F}$.

D. Autonomy Test Coverage Metrics

After the process has completed, we compute metrics that provide quantitative feedback about an autonomous algorithm's performance. These could be used in the future for parameter tuning or comparison of different control strategies.

Specifically, we examine: (1) total failures detected, (2) the spread of the failures in space, recorded by the standard deviation of the failure position, (3) the number of unique

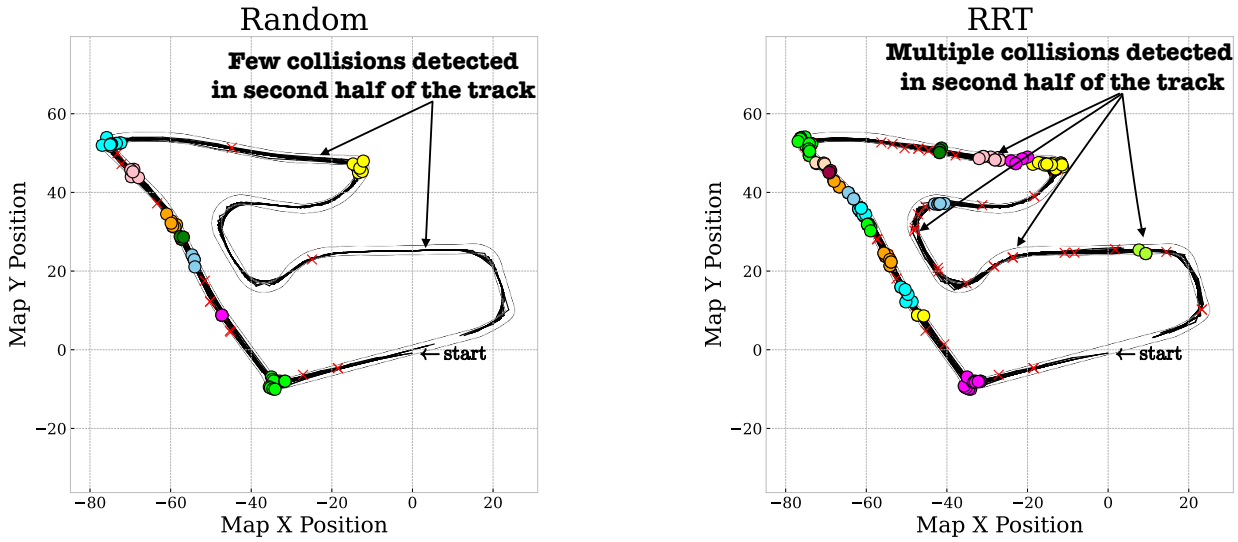

Fig. 3: Spatial clustering visualization of the test results for the gap follower planner. Random adversarial perturbations detected 21 unique crashes (8 clusters and 13 outliers), whereas the proposed approach detected 52 unique crashes (17 clusters and 35 outliers). Crash clusters are shown with colored circles and outliers with a red ×.

failures, measured using the spatial clustering algorithm DBScan [30]. The reason to use a clustering algorithm is that we noticed random testing may find a large number of failures, but they are sometimes very similar (for example, crashing on the first turn of a race). By clustering results in space, these do not get counted twice, and more information is available on the variety of faults that are found. Note spatial clustering contains parameters that requires further tuning per application, and the method has two outputs: (i) the number of clusters found and (ii) the number of outliers that do not belong to any cluster. Adding these two up we compute the number of unique failures. A preview of our results in the next section with autonomous racing and clustering is shown in Figure 3, where both random search and the described RRT algorithm have clustered crashes around the turns, but the RRT approach discovers more unique failures.

IV. Experiments

A. Experimental Setup

We apply the described approach from the previous section to stress test overtake maneuvers in autonomous racing. Autonomous racing provides a clear objective for the motion planners to drive at high speeds and overtake other agents. The unstructured operating environment of autonomous racing also makes failure analysis difficult. For the simulation environment, we use an open source 2D autonomous vehicle simulator created to test path planners and controllers for the F1TENTH autonomous racing competition [5]. The environment is deterministic with realistic vehicle dynamics modeled by a single track model [19]. The physics engine is capable of faster than real-time simulation as well as state serialization (loading and saving), which is needed for our RRT-based algorithm. In addition, collision with the racetrack boundaries and other vehicles is detected automatically, which we consider as failures. A 2D lidar sensor with noise is also available to enable perception-based algorithms such as object detection and localization methods. Finally, the simulation environment is also modular and allows different racetracks and motion planning algorithms to be considered.

We test the following path planners using the approach:

- **Lane Switcher** creates equispaced lanes that spans the entire track in addition to utilizing a curvature optimal raceline [31]. The algorithm makes decision to switch to a specific lane or switch back to the race line when trying to either overtake or block an opponent.
- **Gap Follower** finds gaps in the LiDAR scan by finding the widest range of scan angles that has the highest depth value [32]. Then it steers the vehicle to follow the largest gap to avoid obstacles.
- **Disparity Extender** is an extension of the Gap Follower where the largest difference in the LiDAR scan is used as the target, with modifications to avoid narrow gaps and walls [33].
- **Frenet Planner** is based on a semi-reactive method [34]. This planner is able to select goal coordinates in the Frenet-Frame of the racetrack and generate multiple trajectories to follow the optimal raceline and avoid obstacles by selecting the appropriate trajectory.
- **Graph Planner** generates a graph covering the race track [35]. The nodes in the graph are vehicle poses in the world frame, and the edges of the graph are generated trajectories similar to those in the Frent Planner. The algorithm selects appropriate actions for the vehicle from the action set for overtaking and following when traversing the graph.

Note that for the Disparity Extender, we contacted the University of North Carolina group which won the 2019 F1TENTH race using this algorithm, and were able to use their source code for testing.

To actuate the vehicle to follow created paths we used Pure Pursuit [36], a Stanley controller [37] or an LQR controller for path and velocity tracking. All testing was done using

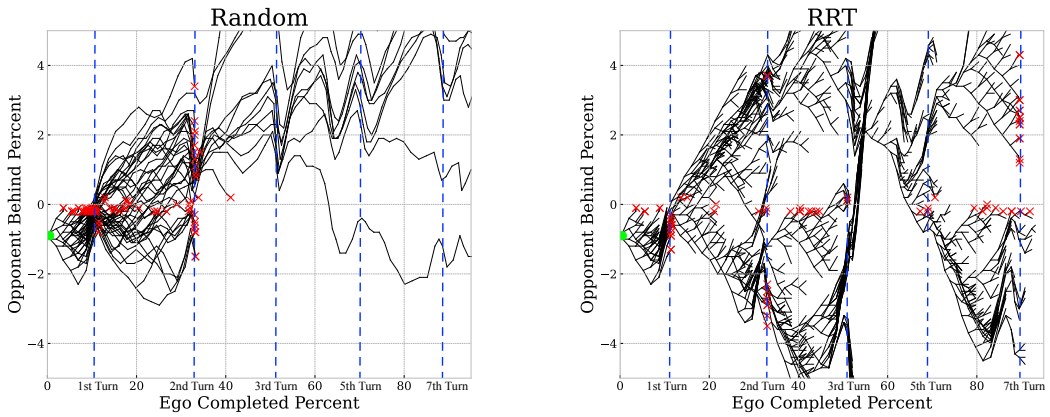

Fig. 4: Objective space plots for 2000 seconds of simulation time for the lane switcher planner using random adversarial perturbations (left) and the proposed RRT method (right). All simulations begin at the same initial state indicated by a green circle at the root of the trees, and collisions are indicated by a red ×.

two vehicles, both of which ran a copy of the same control strategy. Self-testing is useful in this context as the two cars generally travel a similar speed which leads to more overtake interactions. Further, if a collision occurs, there is no ambiguity as to which algorithm is at fault.

We next describe the application-specific parameters needed for the general algorithm presented earlier in Section III-C.

Initial Scenario Configuration: All testing was then done considering a single scenario. The same track and starting position was used for two vehicles, both of which ran a copy of the same control strategy. The opponent car started in front of the ego vehicle. The fixed track start positions define q_{init}.

Adversarial Agent Perturbations: We perturb the adversarial agent by modifying the velocity output command of the opponent vehicle. Two perturbations are considered, $\mathcal{V} = \{opp_{slow}, opp_{fast}\}$, where the former has the opponent moving slightly slower (decreasing the output by 20%), and the latter has the opponent car moving slightly faster (increasing the output by 20%).

Objective Space Projection Function: We define the objective space as the 2-dimensional space with one axis being the percentage of the race track completed by the ego vehicle, and the other as the percentage of the track that the opponent vehicle is ahead of the ego vehicle. To compute the percentage completion along the track, we take a vehicles (x, y) position and project it onto the center line, and divide by the length of the center line. The objective space projection function $\mathcal{P}$ does this computation given the positions of the ego and opponent vehicle.

Given the objective space definition, we can see the results of different exploration strategies. In Figure 4, we compare a random search strategy against the proposed RRT approach. The random approach has simulations that start at the root and end only after completing a lap or a crash, whereas RRT qualitatively explores the objective space more thoroughly.

Objective Space Limits: We use limits $\mathcal{B}$ on the objective space as [-5%, 5%] for percentage of opponent vehicle ahead, and [0%, 95%] for percentage of ego track completion. By limiting the sampling to cases where the cars are close to each other (within 5% of the track distance), we focus the testing effort to cases where overtake is more likely.

Simulation Step Function: The `step_sim` function runs 100 steps of the underlying simulation (one second of simulation time), modifying the opponent's velocity command according to the passed-in adversarial agent perturbation.

B. Experimental Results

We next compare the described RRT-based test generation algorithm with a randomized strategy. The randomized algorithm starts from the initial position, and randomly consider adversarial agent perturbations at each simulation step. The method restarts only when either the ego vehicle crashes or a lap is completed.

We use the autonomy coverage metrics described in Section III-D to evaluate the methods. In addition, we add a metric for crashes detected in the second half of the track, as random testing often only got to the second half of the track when the vehicles were far apart and few overtakes occurred.

The spatial clustering DBScan algorithm requires two parameters. For the maximum distance between two samples to be considered in the same cluster we used 2.1 meters. For the number of samples required to be in the neighborhood to consider it as cluster we used 3. Visually, these parameters produced results that match our intuition about when crashes are similar, as shown before in Figure 3.

The importance of excluding similar crashes and not just counting the total number is also apparent in the objective space plot in Figure 4. Here, a large number of crashes are visible at around 10% and 33% ego track completion, which correspond to the first two turns on the track (vertical dashed blue lines). For all planners, we performed 10 simulation runs with a stopping condition of 2000 seconds of simulation time, corresponding to 2000 nodes explored for the RRT method. Each of the 10 experiments used a different random seed and the metrics are then averaged over all the simulations.

The results are shown in Table I. A result entry of $a \pm b$ shows both the mean a and the standard deviation b for that particular measurement, over all simulation runs. The

TABLE I: Results for five different path planners with both the random and RRT tester. Our RRT test strategy finds more crashes than randomly exploring adversarial agent perturbations.

Planner	Tester	# Crashes	# 2nd Half Crashes	Pos Stddev	# Clusters	# Outliers	# Unique Crashes
Lane	Random	83.4±9.6	5.2±0.9	13.9±1.3	9.5±1.9	18.4±4.0	27.9
Switcher	RRT	254.0±30.8 (3.0x)	15.2±5.8 (2.9x)	17.9±1.8	9.4±1.9	21.3±4.6	30.7 (1.1x)
Gap	Random	65.6±7.2	13.4±1.8	19.2±1	5.4±0.4	21.0±2.6	26.4
Follower	RRT	176.8±19.0 (2.7x)	84.3±18.3 (6.3x)	25.9±1.2	13.2±1.4	33.7±4.2	46.9 (1.8x)
Disparity	Random	73.4±3.6	4.4±2.1	18.2±1.2	4.0±0.6	8.4±1.3	12.4
Extender	RRT	342.5±26.5 (4.7x)	85.6±21.0 (19.5x)	21.3±1.4	8.0±2.1	18.8±2.9	26.8 (2.2x)
Frenet	Random	201.4±21.0	2.0±0.2	8.5±0.9	6.3±2.6	17.0±2.4	23.3
Planner	RRT	423.3±10.8 (2.1x)	219.1±6.5 (109.6x)	28.39±0.8	42.3±2.6	27.0±8.8	69.3 (3.0x)
Graph	Random	23.8±4.2	1.2±0.6	16.9±3.2	2.0±0.0	2.5±2.1	4.5
Planner	RRT	33.5±3.0 (1.4x)	3.2±1.0 (2.6x)	17.7±3.9	3.5±0.5	2.7±0.3	6.2 (1.4x)

RRT approach was able to find more crashes on all tested planners, up to 4.7x more crashes for the disparity extender. In terms of second half crashes, the difference is even more pronounced, with up to 109.6x more crashes being found for the Frenet planner. More importantly, our method finds more unique crashes, computed as the sum of the mean number of clusters and mean number of outliers found with DBScan, up to 3x more for the Frenet Planner. An important aspect to keep in mind is that there are diminishing returns with this metric. Thus, if the RRT method detected twice as many unique crashes, this does not mean that one could just use the random approach for twice as long to get the same result.

In terms of runtime, our approach is fast. Testing some of the simpler planners like the gap follower or lane switcher up to 2000 seconds of simulation could complete in a few minutes, faster than real-time. The more complex algorithms, such as the Graph Planner, generally took longer, as the execution of the controller required more computation time. The bottleneck is typically the execution of the simulation environment and the planner logic, not the test-generation logic, although for planners with significant state the state saving and loading process necessary for RRT can have some impact. We also could further optimize our implementation through parallelization, as simulations are completely disjoint.

V. Discussion

One somewhat surprising result of our study was the large number of crashes detected by the different types of planners. This indicates the difficulty of the problem, and is in line with the outcome of the simulated Indy Autonomous Challenge described in the introduction.

We could likely improve the performance of some of the planners through additional tuning of their parameters. We did spend some effort on each planner and in single-car races all completed a lap quickly and without crashing. Manually doing such tuning requires a detailed understanding of each algorithm and is quite burdensome. The RRT test-generation method presented here, along with the qualitative autonomous coverage metrics, opens up the possibility of automated parameter tuning of both path planners and controllers in future work, similar to what has been done for single-car planners in other publications [38].

In terms of our testing strategy, there are several parameters we could change to optimize the search. Other dimensions could be added to the objective space, such as the horizontal deviation from the center line. Other adversarial agent perturbations could also be considered such as slight deviations in the angle command or different magnitude changes on the velocity gain. Rather than testing all perturbations using the simulator, we could also use application-specific logic to select the one that will drive the system to the closest in the objective space, as is more typical with RRT. Our simulation ran for 100 steps (one second of simulation time) with each call to `step_sim`, which could be adjusted to affect the search. Right now, we also only performed self-testing of a planner against itself. Other variations could compare planners versus each other. While these decisions will likely affect the search performance, we did not explore these possibilities in this work. We are nonetheless encouraged that using reasonable values produced an efficient test-generation engine.

VI. Conclusion

In this paper we presented an approach to stress test autonomous path planning algorithms in unstructured high-speed overtake scenarios. We evaluated five different path planners. By perturbing the opponent car's performance slightly, we were able to explore many variations within a fixed scenario, with the goal of finding cases where the ego vehicle crashes. We demonstrated that randomly searching these variations does detect many crashes, although these are sometimes clustered near the start of the track. A proposed variant of the RRT path planning algorithm overcomes this issue, increasing the number of unique crashes found while keeping the simulation effort constant. With our approach, the number crashes increased by up to 4.7x, the crashes in the second half of the track increased by up to 109.6x, and the number of unique crashes increased up to 3.0x. Our quantitative metrics could be used in the future for automated parameter tuning to optimize for safety during overtake.

Acknowledgment

This material is based upon work supported by the Air Force Office of Scientific Research and the Office of Naval Research under award number FA9550-19-1-0288, FA9550-21-1-0121, FA9550-22-1-0450 and N00014-22-1-2156. Any opinions, findings, and conclusions or recommendations

expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or the United States Navy.

REFERENCES

[1] A. Wischnewski, M. Geisslinger, J. Betz, T. Betz, F. Fent, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle, F. Nobis, L. Ögretmen, M. Rowold, F. Sauerbeck, T. Stahl, R. Trauth, M. Lienkamp, and B. Lohmann, "Indy autonomous challenge – autonomous race cars at the handling limits," 2022.

[2] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," 2022.

[3] Energy Systems Network, "Polimove wins ansys indy autonomous challenge simulation race," https://www.indyautonomouschallenge.com/blog/polimove-wins-ansys-indy-autonomous-challenge-simulation-race, 2021.

[4] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[5] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, H. J. Escalante and R. Hadsell, Eds., vol. 123. PMLR, 08–14 Dec 2020, pp. 77–89. [Online]. Available: http://proceedings.mlr.press/v123/o-kelly20a.html

[6] J. Kim, J. M. Esposito, and V. Kumar, "Sampling-based algorithm for testing and validating robot controllers," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1257–1272, 2006.

[7] P. Cheng and V. Kumar, "Sampling-based falsification and verification of controllers for continuous dynamic systems," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1232–1245, 2008.

[8] A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas, "Robust test generation and coverage for hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2007, pp. 329–342.

[9] Y. Annpureddy, C. Liu, G. Fainekos, and 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.

[10] 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.

[11] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, "Multiple shooting, cegar-based falsification for hybrid systems," in *Proceedings of the 14th International Conference on Embedded Software*, 2014, pp. 1–10.

[12] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski, "A trajectory splicing approach to concretizing counterexamples for hybrid systems," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 3918–3925.

[13] J. F. Quindlen, U. Topcu, G. Chowdhary, and J. P. How, "Active sampling-based binary verification of dynamical systems," in *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 1107.

[14] M. Althoff and J. Dolan, "Online verification of automated road vehicles using reachability analysis," *Robotics, IEEE Transactions on*, vol. 30, no. 4, pp. 903–918, Aug 2014.

[15] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technische Universität München, 2010.

[16] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler, "Recent progress in continuous and hybrid reachability analysis," in *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*. IEEE, 2006, pp. 1582–1587.

[18] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, A. Bertolino *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[17] N. C. Jewell, "Embedded reachability for autonomous racecar," 2021.

[19] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2017. [Online]. Available: https://doi.org/10.1109/ivs.2017.7995802

[20] T. Stahl and J. Betz, "An open-source scenario architect for autonomous vehicles," in *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, Sep. 2020. [Online]. Available: https://doi.org/10.1109/ever48776.2020.9243029

[21] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Advances in Neural Information Processing Systems*, 2018, pp. 9827–9838.

[22] J. Norden, M. O'Kelly, and A. Sinha, "Efficient black-box assessment of autonomous vehicle safety," *arXiv preprint arXiv:1912.03618*, 2019.

[23] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1555–1562.

[24] M. Klischat and M. Althoff, "Generating critical test scenarios for automated vehicles with evolutionary algorithms," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2352–2358.

[25] D. Karunakaran, S. Worrall, and E. Nebot, "Efficient falsification approach for autonomous vehicle validation using a parameter optimisation technique based on reinforcement learning," *arXiv preprint arXiv:2011.07699*, 2020.

[26] T. A. Wheeler and M. J. Kochenderfer, "Critical factor graph situation clusters for accelerated automotive safety validation," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.1109/ivs.2019.8813845

[27] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2021, retrieved 2021-03-12 11:41:11+01:00. [Online]. Available: https://www.fuzzingbook.org/

[28] P. Godefroid, M. Y. Levin, D. A. Molnar *et al.*, "Automated whitebox fuzz testing." in *NDSS*, vol. 8, 2008, pp. 151–166.

[29] S. Sheikhi, E. Kim, P. S. Duggirala, and S. Bak, "Coverage-guided fuzz testing for cyber-physical systems," in *Proceedings of the 13th International Conference on Cyber-Physical Systems*. ACM/IEEE, 2022.

[30] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[31] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle System Dynamics*, vol. 58, no. 10, pp. 1497–1527, Jun. 2019. [Online]. Available: https://doi.org/10.1080/00423114.2019.1631455

[32] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm:"follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[33] N. Otterness, "Disparity extender." [Online]. Available: https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html

[34] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, May 2010. [Online]. Available: https://doi.org/10.1109/robot.2010.5509799

[35] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, Oct. 2019. [Online]. Available: https://doi.org/10.1109/itsc.2019.8917032

[36] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[37] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[38] H. Zheng, J. Betz, and R. Mangharam, "Gradient-free multi-domain optimization for autonomous systems," 2022.