Adversarial Scene Generation for Virtual Validation of Off-Road Autonomous Vehicles

Ted Sender, Ram Vasudevan, Bogdan Epureanu
University of Michigan
Ann Arbor, MI
{tsender,ramv,epureanu}@umich.edu

Mark Brudnak
U.S. Army DEVCOM Ground Vehicle Systems Center
Warren, MI
mark.j.brudnak.civ@army.mil

Reid Steiger
Ford Motor Company
Dearborn, MI
rsteige3@ford.com

ABSTRACT

Perception models based on machine learning and deep neural networks are susceptible to misclassifications from subtle perturbations to their inputs. This is concerning because on-road autonomous vehicles (AVs) encounter a variety of perturbations that arise from uncertainty due to the unpredictable behaviors of other actors. Off-road AVs face even greater uncertainty due to the large variation in natural, unstructured environments. Numerous AV simulation tools and algorithms have been created to efficiently explore (and even improve) the robustness of machine learning algorithms for on-road AVs, however, only a handful of tools have demonstrated usefulness for the off-road domain.

This paper presents a modeling and simulation approach for generating adversarial scenes for off-road AVs using reinforcement learning. By "adversarial" we mean that the scene is (ideally) maximally problematic to navigate by the vehicle's autonomy system while constrained to be realistic. Our work consists of three components: a high-fidelity simulation platform, an adversarial scene generator, and an autonomy system under test. The simulation platform is designed using Unreal Engine 4 and uses a custom plugin to enable users to automatically create basic off-road scenes (e.g., flat ground plane) and run various navigation scenarios. The adversarial scene generator (ASG) uses a distributed Twin Delayed Deep Deterministic Policy Gradient algorithm with Prioritized Experience Replay and a novel Action Saturation Penalty to create test scenarios. The example autonomy system under test has a perception system with a U-Net architecture to predict traversable regions from camera images and uses an A* path planner to avoid the non-traversable regions.

We present results that demonstrate that the ASG architecture can generate pathological scenes on a flat ground plane with up to 32 obstacles. We present studies that highlight various features of the generated scenarios and their implications, and finally we conclude with limitations and future work.

ABOUT THE AUTHORS

Ted Sender is a Ph.D. candidate in the Department of Mechanical Engineering at the University of Michigan. His research focuses on the development of algorithms to identify weaknesses in vehicle autonomy algorithms.

Mark Brudnak is a Senior Technical Expert in Immersive Simulation at the U.S. Army DEVCOM Ground Vehicle Systems Center.

Reid Steiger is a Technical Expert in Automated Driving at Ford Motor Company.

Ram Vasudevan is an Associate Professor in the Departments of Mechanical Engineering, Electrical Engineering and Computer Science, and Robotics at the University of Michigan.

Bogdan Epureanu is an Associate Professor in the Department of Mechanical Engineering and the Director of the U.S. Army Automotive Research Center of Excellence at the University of Michigan.

Adversarial Scene Generation for Virtual Validation of Off-Road Autonomous Vehicles

Ted Sender, Ram Vasudevan, Bogdan Epureanu
University of Michigan
Ann Arbor, MI
{tsender,ramv,epureanu}@umich.edu

Mark Brudnak
U.S. Army DEVCOM Ground Vehicle Systems Center
Warren, MI
mark.j.brudnak.civ@army.mil

Reid Steiger
Ford Motor Company
Dearborn, MI
rsteige3@ford.com

INTRODUCTION

Autonomous vehicles (AVs) are complex machines with advanced vehicle dynamics and (recently) advanced perception, navigation, and control subsystems. Due to the advancements in machine learning, numerous perception/control algorithms within an AV's autonomy stack rely on deep neural networks (DNNs) (LeCun et al., 2015), however, DNNs have been shown to be susceptible to subtle variations in their inputs (Szegedy et al., 2014). Small perturbations to a DNN's inputs can result in significantly incorrect output predictions/classifications. A large reason for this nature is simply due to the difficulty of accounting for uncertainty in the DNN's inputs. While one may be concerned with an adversary attacking the DNN by modifying the input stream, we focus our attention to uncertainty that arises from interactions with the natural world. Both on-road and off-road AVs share sources of uncertainty from weather and interactions with dynamic actors, but specific to on-road AVs is the unpredictable behavior of other actors, and specific to off-road AVs is the large variation in natural, unstructured environments. Validating whether an AV's performance in its natural environment is robust and reliable is critical to their widespread deployment but also remains a major difficulty.

Simulation has become an influential tool for many researchers as a cost-effective alternative for validating and testing AVs. Most of the published research in the field of AV testing, however, are designed for or only demonstrate effectiveness in on-road applications and specifically pertain to what are known as safety-critical cases in which vehicle failure can be catastrophic. Ding et al., (2022) provide a thorough summary of the recent safety-critical works. Consequently, numerous driving simulators (CARLA (Dosovitskiy et al., 2017), WeBots (Michel 2004), Prescan (Hendriks et al., 2010), VISTA 1.0 (Amini et al., 2020), VISTA 2.0 (Amini et al., 2021), etc.) and various datasets (ApolloScape (Wang et al., 2019), Berkeley DeepDrive (Yu et al., 2020), KITTI (Geiger et al., 2013), Cityscapes (Cordts et al., 2016), etc.) have been created to assist developers improve the robustness of the on-road AV machine learning components. But few, if any, of these works are helpful for autonomous off-road driving. The only known work for off-road vehicle testing, to the best of our knowledge, is from Han et al., (2021) in which an off-road simulation platform is developed to test AV driving policies.

There are two clear gaps in the context of off-road AV evaluation - the need for both off-road simulation platforms *and* testing/validation algorithms for AV performance. Even though simulators provide the necessary tools to programmatically manipulate the virtual environment, we do not know of any library specifically designed for creating off-road driving scenarios automatically (i.e., <1 second and at runtime) for the purpose of testing an AV in diverse off-road conditions. The drawback of the platform developed by Han et al., 2021 is that all the pre-made driving environments are fixed and most/all modifications to the scene must be made manually. Regarding off-road AV testing, first, we believe there should exist a common framework for defining off-road driving scenarios, like NHTSA's operational design domain (ODD) (Thorn et al., 2018) framework. Second, since safety-critical scenarios are not as relevant to the off-road domain, we believe there is a greater need for identifying unsuspecting scenarios that pose a challenge for the vehicle's autonomy stack and result in unexpected/poor performance. And third, given that any off-road driving scenario can easily be parameterized by tens or hundreds of parameters at a minimum, we strongly recommend that any algorithm designed for testing/validating off-road AV performance be capable of working with large decision spaces.

The goal of this paper is to help bridge the off-road AV evaluation gap (as discussed above) with the following contributions:

1. We propose a scenario decomposition strategy tailored to the off-road domain.
2. We present a deep reinforcement learning framework using a Distributed Twin-Delayed Deep Deterministic Policy Gradient algorithm with Prioritized Experience Replay and a novel Action Saturation Penalty to generate adversarial scenarios for an off-road AV's autonomy stack.
3. And we present a few studies demonstrating capabilities of our adversarial scene generation framework.

OFF-ROAD SCENARIO DECOMPOSITION

In this section we will introduce our proposed off-road scenario decomposition. The U.S. National Highway Traffic Safety Administration (NHTSA) developed the operational design domain (ODD) (Thorn et al., 2018) as an attempt to provide a unified framework for testing (on-road) automated vehicles. The ODD is composed of attributes which effectively describe the range of scenarios/conditions that a vehicle is designed to operate in. While the ODD framework is imperfect and it can be challenging to list/quantify all possible attributes (Koopman & Fratrik 2019), we believe that AV developers should adhere to a unified taxonomy when discussing driving scenarios. Since the focus of our work is the off-road domain, we propose a modified set of primary attribute categories: *structural attributes*, *textural attributes*, and *operational attributes*.

Structural attributes define the geometry of all the structural objects that compose the scene. This includes the ground surface / heightmap, static obstacles (e.g., trees, bushes, rocks), water bodies, etc. In a general manner, we refer to the obstacles as structural scene actors or SSAs. When defining simulated test scenarios, for most SSAs the following attributes will suffice: x-position, y-position, yaw angle, and mesh scale factor (z-position can be omitted because the simulator has access to the ground plane).

Textural attributes enhance the visual realism by describing visual effects of the scene. Common attributes include the intensities of various weather/lighting conditions and the sun position. If one has a capable enough simulator, then one may even include the color of foliage leaves and the ground surface textures at various locations.

Operational attributes describe operational constraints imposed on any vehicle's behavior within the scene (e.g., go/no-go areas, speed ranges/limits, and areas without GPS), can define the vehicle's task (e.g., start/goal locations), as well as describe the behavior of other agents (e.g., non-player characters).

We can then define an *off-road driving scenario* by a collection of attributes, $x \in \mathcal{X}$, where $\mathcal{X}$ is the set of all possible off-road scenario descriptions from the user-defined attributes.

PROBLEM DEFINITION

At a high-level, our goal is to identify reasonable scenarios (that would not pose any difficulty for a human driver) that lead the vehicle to exhibit unexpected and poor behavior, such as traveling on a longer route when a shorter one exists or colliding into an obstacle. We can describe this idea with what we call the *scenario difficulty gap*, which represents the "scenario's difficulty perceived by the AV" minus the "scenario's actual difficulty". If we can define a function $d: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ to compute this difficulty gap, where $\mathcal{X}$ is defined in the previous section and $\mathcal{Y}$ represents the set of all possible vehicle trajectories, then our goal is to find the scenario(s) with the largest difficulty gap defined by

$$x^* = \arg\max_{x \in \mathcal{X}} d(x, y(x)), \tag{1}$$

where $y(x)$ represents the resulting vehicle trajectory. Unfortunately, this notion of difficulty gap is abstract and near impossible to quantify, so we instead focus our attention to a modified optimization problem of solving

$$x^* = \arg\max_{x \in \mathcal{X}} \tilde{d}(x, y(x)), \tag{2}$$

where $\tilde{d}$ is a concrete and computationally tractable proxy function for d and should be designed to capture a specific failure mode, such as traveling along unnecessarily long/windy paths or colliding into obstacles.

ADVERSARIAL SCENE GENERATION

This section describes our approach for solving (2). Prior works in the literature propose simulated annealing (Tuncali et al., 2018, Tuncali et al., 2020) and genetic algorithms (Zheng et al., 2020) to identify scenario parameters to maximize a difficulty/cost function, but these methods are known to incur exponentially larger runtime costs as the decision space increases. Since off-road scenarios can be defined by many attributes, we propose the use of gradient-based search methods because these methods generally scale well with the number of decision variables. We specifically propose to frame the problem of solving (2) in the reinforcement learning (RL) context and use a Distributed Twin-Delayed Deep Deterministic Policy Gradient algorithm with Prioritized Experience Replay and a novel Action Saturation Penalty. This approach falls under the actor-critic RL framework in which we use the Twin-Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto et al., 2018) as the base actor-critic algorithm. We also utilize prioritized experience replay (Schaul et al. 2016) to prioritize learning from past difficult scenarios and the novel action saturation penalty helps prevent the actor from choosing extreme values during the early stages of training.

To help frame the problem in the RL context, it will be helpful to decompose the scenario space into a set of fixed attributes $\mathcal{X}_{fixed}$ and a set of variable attributes $\mathcal{X}_{var}$ as the user may want to fix certain attributes. The action space is simply $\mathcal{X}_{var}$. The state space is given by the set $\mathcal{X} \times \mathcal{Y}$, which contains the full state description and the vehicle's resulting trajectory. However, if some of the fixed attributes are not used in the difficulty proxy calculation, then we may omit them from the state vector that is passed to the actor network. We also need to keep in mind that when running a simulation, we have access to the entire vehicle's state and control inputs, however, given the frame rate at which most simulations run at, the resulting vehicle's trajectory at the t-th simulation will contain an excessively large number of points given by $N_{simpath,t}$. To keep the state vector at a reasonable size, we temporally sample the vehicle's state and control information along the vehicle's trajectory into N_{path} data points, where N_{path} is (generally) much smaller than $N_{simpath,t}$. By construction, the state vector requires running the simulation with the given scenario. Finally, the reward is the difficulty proxy's evaluation.

Figure 1 provides an illustration of the adversarial scene generation framework, which includes the process for training the RL agent and for generating adversarial scenarios. The framework consists of three layers: the adversarial scene generator (ASG), an RL backbone, and a simulation platform. Even though the ASG and RL backbone could be considered a single layer, we intentionally separate them since part of the RL backbone is only used for training. The process is as follows. The actor processes the current state and outputs an action, which in this case is the variable scene description. An Unreal Engine (UE) worker is then given a scenario request consisting of the entire scenario description, and the UE worker begins communicating with a remote autonomy stack to carry out the navigation task. When the simulation is complete, the UE worker sends the vehicle's trajectory information to a processing block which outputs three items: 1) the resulting next state vector, 2) the (state, action, reward, next state) RL transition to add to the replay buffer, from which we can sample from and train the critic/actor networks, and 3) a more detailed set of information regarding the entire scenario to store for further review.

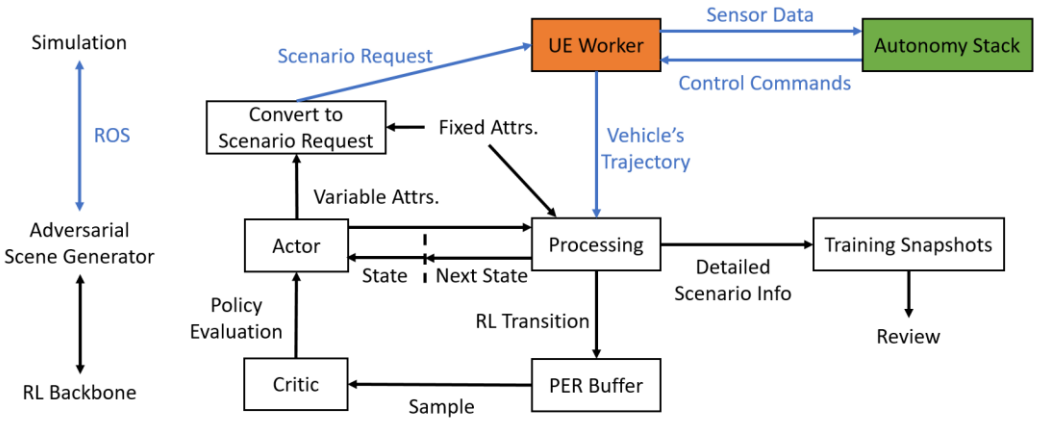


Figure 1. Adversarial scene generation framework.

EXPERIMENTAL RESULTS

We have already discussed the process for generating/identifying adversarial scenes. We will first describe two remaining ingredients needed to perform our experiments: 1) a simulation platform to create virtual off-road scenes, and 2) the base autonomy system under test. Following these two topics, we will then describe our experiments and their results.

AutomaticSceneGeneraton: Creating Off-Road Scenes with Unreal Engine

One of the two main virtual off-road AV testing gaps is the ability to create diverse off-road scenes quickly and automatically in simulation. While almost every modern simulation platform provides the tools to create scenes via a user-friendly graphical interface and provides the functionality to create/modify most components at runtime, to the best of our knowledge, there does not exist such a simulation platform or add-on that provides this desired capability out-of-the-box. Hence, we developed a custom plugin for Unreal Engine 4 (UE4) called AutomaticSceneGeneration specifically for creating off-road scenes and performing navigation tasks. For user convenience, this plugin provides a ROS interface by leveraging the ROSIntegration plugin developed by Mania & Beetz (2019) as the main form of communication.

The main components of the AutomaticSceneGeneration plugin include:

1. *AutoSceneGenWorker*: This is the primary actor whose role is to construct virtual scenes and monitor vehicle navigation tasks. The user provides a ROS scenario request consisting of the scene description (contains structural, textural, and operational attributes) and the navigation task (start location, goal location, etc.). The actor creates the scene, places the vehicle at the start location, and then lets external ROS nodes control the vehicle. Once the vehicle reaches the goal or crashes into an obstacle, the actor will terminate the run, return the vehicle's trajectory data to the user, and wait for the next scenario request.

2. *AutoSceneGenVehicle*: This vehicle class provides the base functionality for creating a custom virtual vehicle blueprint. This class handles internal communication with the AutoSceneGenWorker and provides a drive-by-wire module that accepts throttle, steering, and braking commands from a ROS message. The user simply specifies the skeletal mesh and attaches any of the custom sensors to the vehicle. All sensor data are published on ROS topics so that external ROS nodes can perform the autonomy calculations and then determine the desired control commands.

3. *Custom Vehicle Sensors*: The existing sensors we provide include: color camera, depth camera, semantic segmentation camera, and a localization sensor.

4. *StructuralSceneActor*: This actor class defines the needed functionality for placing structural scene actors (e.g., trees, bushes, rocks, etc.). The user creates blueprints inheriting from this class and simply provides the desired mesh. The act of placing obstacles in the scene is handled directly by the AutoSceneGenWorker.

We also provide an external ROS2 interface with the needed functionality to communicate with the AutoSceneGenWorker and AutoSceneGenVehicle actors. The ROS2 interface contains two main ROS nodes that the user can inherit from:

1. *AutoSceneGenClientNode*: This is the base ROS node for communicating with the AutoSceneGenWorker. It can connect to any number of AutoSceneGenWorkers (one per UE4 editor), submit the scenario requests, and store the scenario results (vehicle trajectory, vehicle success/crash flags, etc.) from the AutoSceneGenWorker.

2. *AutoSceneGenVehicleNode*: This is the base ROS node the user should use as part of the AutoSceneGenVehicle's autonomy stack.

Figure 2 illustrates the workflow between the AutomaticSceneGeneration plugin and the external ROS2 interface. While this figure only shows the AutoSceneGenClient interacting with a single UE worker, the external ROS2 interface is designed to work with any number of UE workers to parallelize experimentation.

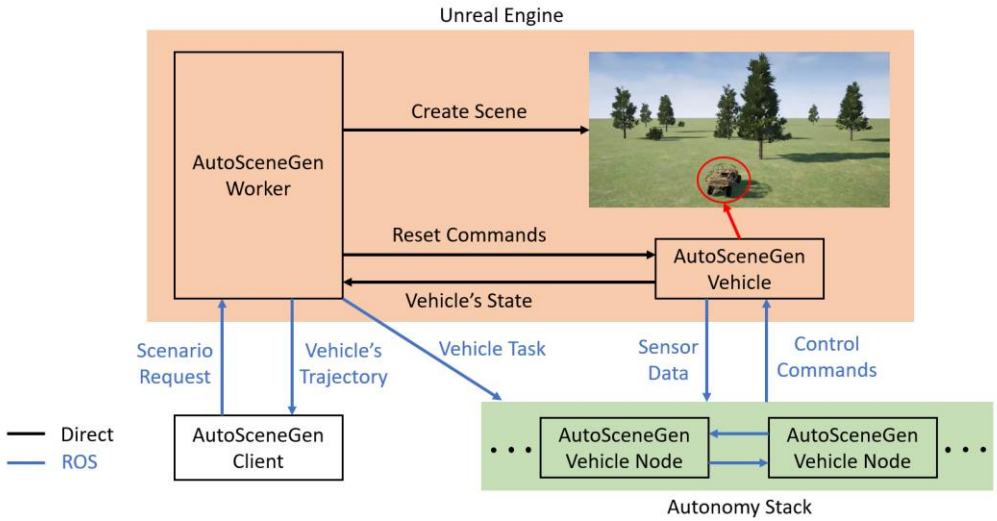


Figure 2. Workflow for interacting with the AutomaticSceneGeneration plugin for Unreal Engine. Not shown in this diagram are ROS status communication lines. All AutoSceneGen* components in this diagram monitor each other's status to ensure all entities are in sync with each other before proceeding with their respective operations.

Base Autonomy System

The test vehicle in our experiments is a Polaris MRZR vehicle model (with hand-tuned model parameters) equipped with a forward-facing camera and a localization sensor. The camera is a combined RGB and depth-camera with a 90-degree field of view and a resolution of 256x256 running at 15 Hz. The localization sensor provides the vehicle's position and orientation and runs at 60 Hz. The full autonomy system is illustrated in Figure 3. The perception system on the vehicle is a custom traversability semantic segmentation DNN that labels each pixel from the camera images as belonging to an object that is traversable (i.e., the ground plane), non-traversable (i.e., SSAs), or to the sky. This semantic segmentation network is based on the U-Net architecture (Ronneberger et al., 2015) and is trained with categorical cross-entropy. The labeled images for the training dataset are created using an automatic image labeler from the AutomaticSceneGeneration plugin. Example raw and semantic segmentation images are shown in the upper-left part of Figure 3. The vehicle's path planner uses an A* voting path planner. The A* planner discretizes the environment into an obstacle grid and vehicle grid. The obstacle grid indicates possible obstacle locations and keeps track of how many votes each vertex has. The vehicle grid contains nodes connected by edges that the path planner uses to generate paths. For computational performance, the obstacle grid may be finer than the vehicle grid, but we set the node spacing to 5 meters in both grids. Example grids are shown in the lower-right part of Figure 3. At runtime, the segmentation network produces predicted labels from the raw camera images, and if any labels correspond to points in 3D space with a height of less than 2 meters, then the closest obstacle node is given a vote, with a 50-vote maximum per obstacle node. The A* path planner waits 1 second before planning the first path, and then every 2 seconds it generates a new path to reach the goal location. The cost of traversing any edge in the vehicle grid is equal to the arc-length of that edge times a scaling factor. The more obstacle votes near a vehicle grid edge, the larger the scaling factor. The path follower runs at 20 Hz and uses a basic PD controller to follow the most recently generated path with a constant tracking speed.

Experiments and Results

The purpose of the experiments that we will present is to demonstrate the capability of our proposed algorithm. While there are numerous vehicle performance failure modes that one may be concerned with (collision, taking unnecessarily long/windy paths, sudden braking, etc.), each deserving its own attention, in this paper we only focus on identifying scenarios that result in unnecessarily long/windy paths taken by our autonomy system. For demonstrative purposes, we will test our RL agent against an intentionally flawed autonomy system. The flaw that we inject is to train the semantic segmentation network on a dataset consisting of only Barberry bushes, despite there being both Barberry bushes and Juniper trees in all our experiments, see Figure 4 for example vegetation meshes. In our preliminary testing, we found that this faulty perception network misclassifies shadows as obstacles. We thus aim to show that the RL

agent can take advantage of this flaw and can identify scenarios that result in pathological routes taken by the vehicle. The goal of the RL agent is to place the obstacles in such a way that the vehicle travels along long/windy paths when a much more efficient path exists. Since we are using a flat ground plane, the dimension of the action space is twice the number of obstacles (for the x- and y-positions).

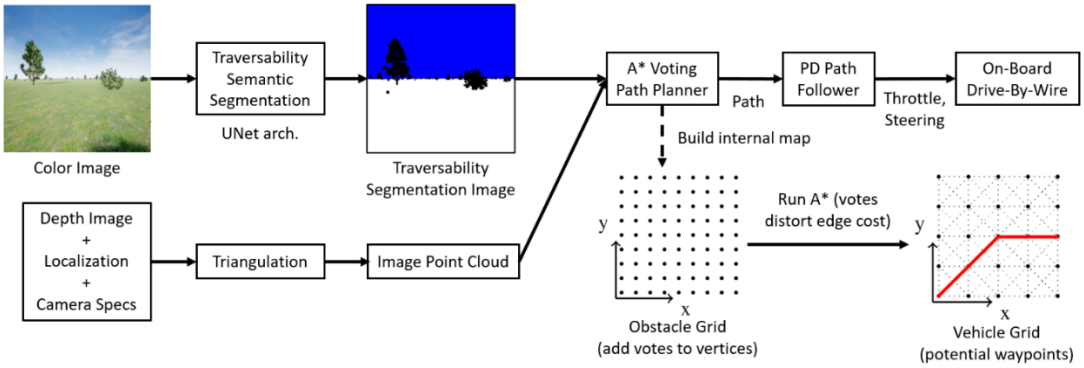


Figure 3. A* traversability autonomy stack.

Figure 4. Vegetation meshes used in our simulations: Barberry bush (left) and Juniper tree (right).

Table 1 lists the parameters that vary between experiments. We impose a safety radius around the start and goal locations in which any obstacle placed in these regions are removed. We also consider the vehicle to have reached the goal if it is within a specified goal radius. Both the safety and goal radius are set to 11 meters. The difficulty proxy function used by the ASG is defined by

$$\tilde{d}_{longwindy} = 0.1vis + \frac{10}{N_{path}}\sum_{i=1}^{N_{path}} \psi_i^2 + PLR^2, \qquad (3)$$

where $vis \in [-1,1]$ indicates how visible all obstacles are in the camera's field of view at some point during the simulation, the second term is a weighted mean-square yaw rate (from the discretized path, with $N_{path} = 250$), and PLR is the path length ratio between the arc length of the vehicle's path and the arc length of the optimal route assuming global knowledge of all obstacles a priori. For each experiment, we train the RL agent for 1000 episodes with five steps per episode. Collecting data for a single experiment, using eight parallel simulations/workers, takes approximately eight hours.

Table 1. Experiment Parameters.

	Landscape Size [m]	Start Location (x,y) [m]	Goal Location (x,y) [m]	Number of Juniper Trees, Barberry Bushes	Vehicle Speed [m/s]
Exp. 1	100 × 100	(20, 20)	(80, 80)	4 , 4	5
Exp. 2	200 × 200	(50, 50)	(150, 150)	8 , 8	5
Exp. 3	200 × 200	(50, 50)	(150, 150)	16 , 16	5
Exp. 4	100 × 100	(20, 20)	(80, 80)	4 , 4	2.5

In the first three experiments, our goal is to show that even with an increase in the size of the action space (by adding more obstacles to the scene), the RL agent is still capable of identifying a variety of difficult scenarios with respect to the proxy function defined. Note, due to the size of the safety/goal radii, we increase the size of the landscape and the distance between the start/goal locations to accommodate more obstacles in experiments 2 and 3. Based on our evaluation of the generated scenes, we find that any scene with a proxy evaluation ≥ 3 is considered worthy of further

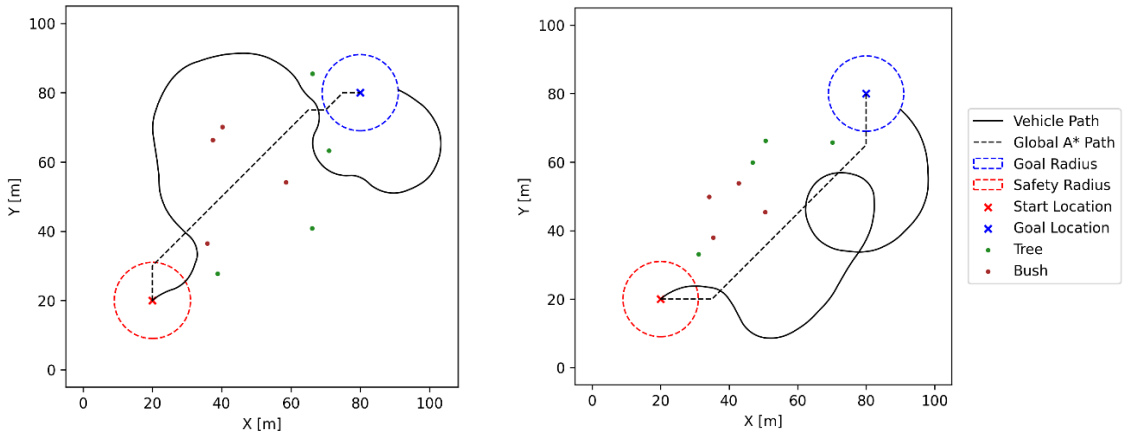
consideration by the AV developer, and evaluations ≥ 4 are strongly worthy of further consideration. Table 2 lists the number of scenarios with proxy scores of at least 3 and 4 for all experiments, and it also shows the largest proxy evaluation encountered. Keep in mind that we only ran each type of experiment once; since deep reinforcement learning utilizes stochastic gradient descent, one cannot say that the RL agent is better at identifying difficult scenarios in one situation than another unless numerous trials are performed. What we can say, however, is that even in just a single experiment, the RL framework we proposed can identify a considerable number of difficult scenarios worthy of closer review. Regarding experiment 4, since it would be unfair of us to remove shadows from all objects, we instead show that even if the vehicle moves at a slower speed of 2.5 m/s, the RL agent can still find difficult scenes.

Table 2. Difficulty proxy summary for each experiment.

	Num. Scenarios with Proxy ≥ 3	Num. Scenarios with Proxy ≥ 4	Largest Proxy Evaluation
Exp. 1	682	138	8.31
Exp. 2	126	7	6.79
Exp. 3	477	82	6.67
Exp. 4	305	67	8.00

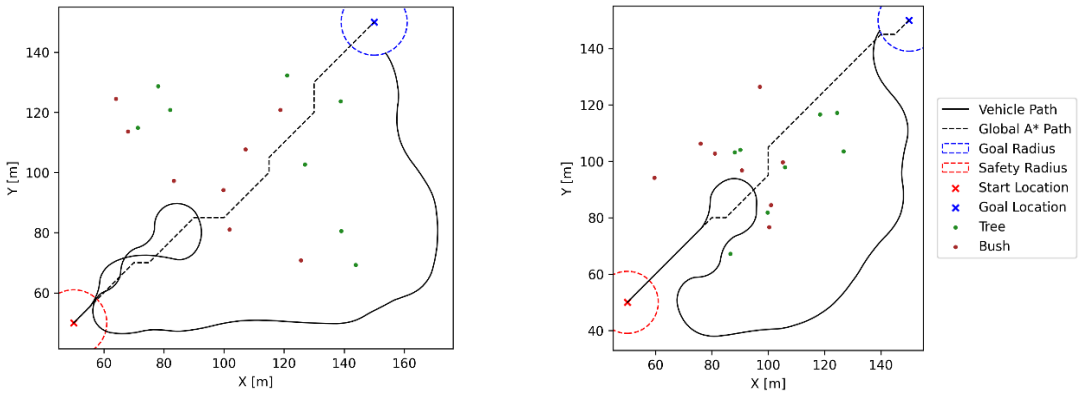
Figures 5-8 show high-level summaries of the top two most difficult scenarios (referred to as rank 1 and rank 2) identified in experiments 1-4, respectively. The dashed path represents the optimal path assuming full knowledge of all obstacle locations a priori. We see that in most of these scenarios, the vehicle's trajectory includes a loop and/or travels around almost all the obstacles with an excessively long path. The reason for this behavior is a combination of the vehicle's flawed perception system (of misclassifying shadows as obstacles) and the various parameters behind the path planner and path follower. Most of the scenarios mentioned in Table 2 exhibit such behavior.

To better understand how the autonomy stack's decisions result in such pathological routes, in Figures 9 and 10 we provide a look inside the path planner and path follower's decisions for a replay of the scenario shown in Figure 5b. Unfortunately, when replaying a scenario in our simulation, we do not always see the same behavior that was seen during the training process; we often see very different behavior during the replays. The reason for this phenomenon is unknown and is an ongoing investigation. Nevertheless, the replay we will look at shows similar vehicle behavior to what is seen in Figure 5b. Figure 9 shows how the vehicle avoids the tree nearest the start location, and Figure 10 shows why the vehicle makes a loop by the tree nearest the goal location. The reason why the vehicle makes the loop is because at the snapshot shown in Figure 10a, the tracking point for the path follower is to the left of the vehicle, and consequently the vehicle begins to turn left. Since our vehicle can only travel at a constant speed, the vehicle drives in a loop. In both Figures 9 and 10, the intensity of the red dots indicates the number of obstacle votes assigned to that obstacle vertex from the perception system.



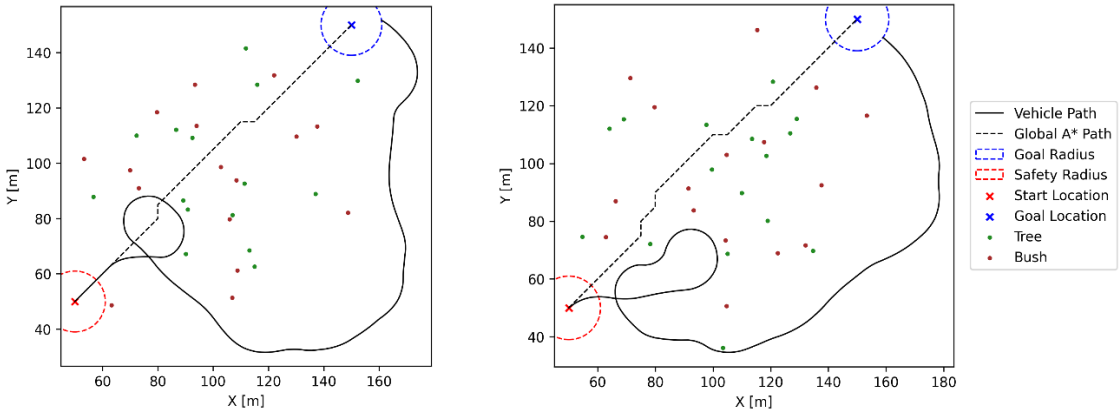
(a) Rank 1 scenario, proxy = 8.31. **(b) Rank 2 scenario, proxy = 7.13.**

Figure 5. High-level summaries of the top two most difficult scenarios from experiment 1.



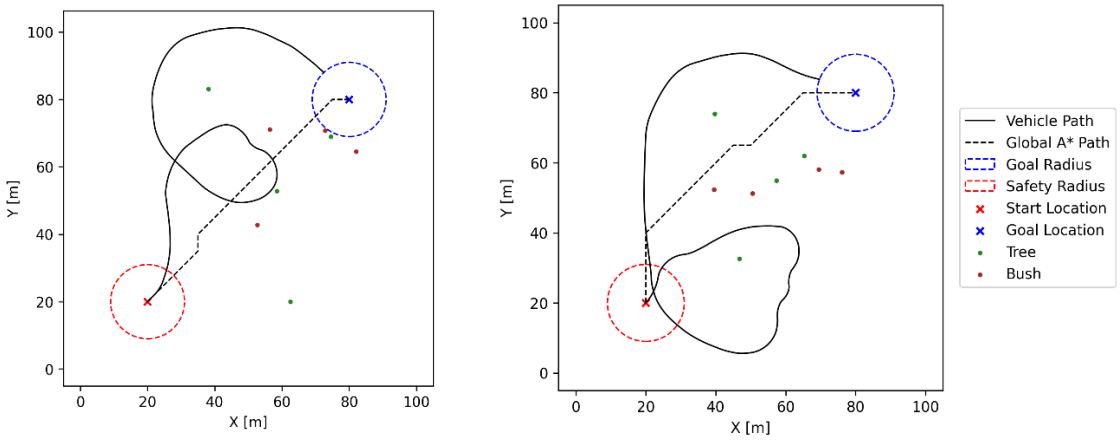
(a) Rank 1 scenario, proxy = 6.79. **(b) Rank 2 scenario, proxy = 5.88.**

Figure 6. High-level summaries of the top two most difficult scenarios from experiment 2.



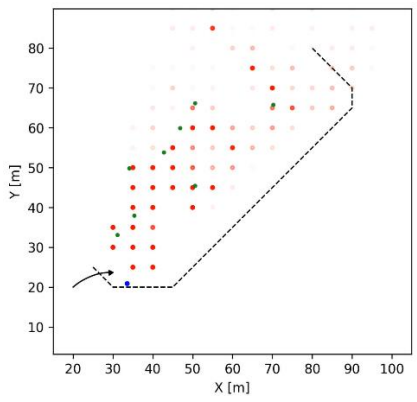
(a) Rank 1 scenario, proxy = 6.67. **(b) Rank 2 scenario, proxy = 6.35.**

Figure 7. High-level summaries of the top two most difficult scenarios from experiment 3.

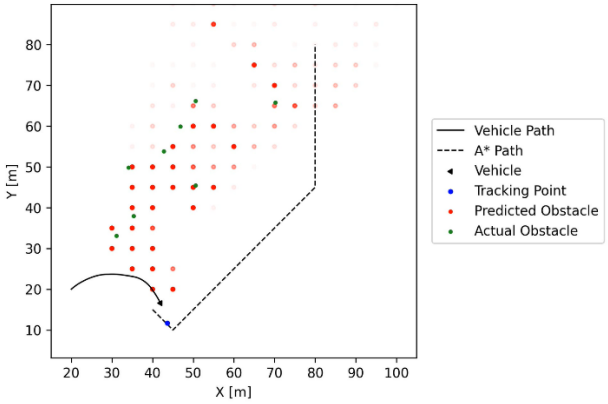


(a) Rank 1 scenario, proxy = 8.00. **(b) Rank 2 scenario, proxy = 7.96.**

Figure 8. High-level summaries of the top two most difficult scenarios from experiment 4.

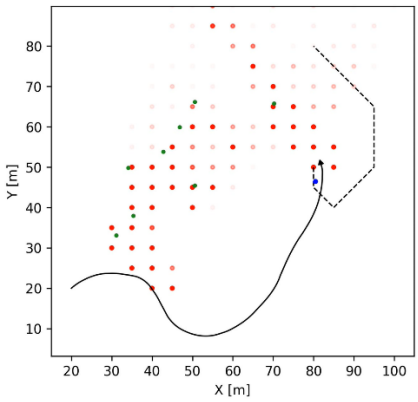


(a) Vehicle approaches a tree's shadow and begins to go around it.

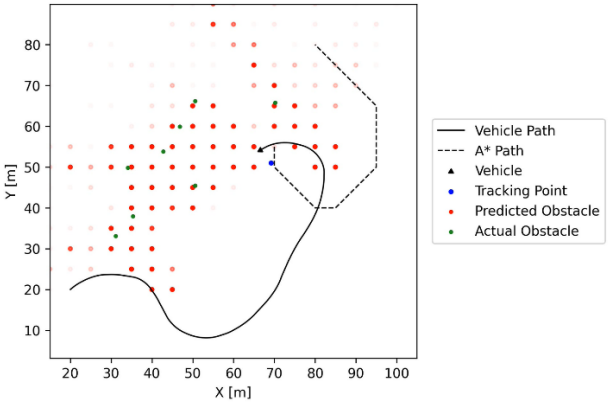


(b) Vehicle sees more of the shadow and continues to avoid it.

Figure 9. Path planner replay of scenario in Figure 5b when the vehicle avoids the first tree.



(a) Vehicle approaches another tree's shadow and begins to avoid it by turning inward.



(b) Vehicle continues to turn inward and eventually makes a full loop before reaching the goal.

Figure 10. Path planner replay of scenario in Figure 5b when the vehicle drives in a loop.

CONCLUSIONS

In this work, we developed a custom off-road simulation platform, proposed a novel off-road scenario decomposition framework, and show that our proposed reinforcement learning framework can identify numerous scenarios that pose navigation difficulties for our (flawed) custom autonomy stack. In future work, we plan to show how our framework can find autonomy weaknesses throughout the AV design process, and we also intend to improve the capabilities of our simulation platform to control weather conditions and test vehicles on a non-flat ground plane.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the technical and financial support of the Automotive Research Center (ARC) in accordance with Cooperative Agreement W56HZV-19-2-0001 U.S. Army CCDC Ground Vehicle Systems Center (GVSC) Warren, MI.

REFERENCES

Amini, A., Gilitschenski, I., Phillips, J., Moseyko, J., Banerjee, R., Karaman, S., & Rus, D. (2020). Learning Robust Control Policies for End-to-End Autonomous Driving from Data-Driven Simulation. *IEEE Robotics and Automation Letters, 5(2)*, 1143-1150.

Amini, A., Wang, T.H., Gilitschenski, I., Schwarting, W., Liu, Z., Han, S., Karaman, S., & Rus, D (2021). Visat 2.0: An Open, Data-Driven Simulator for Multimodal Sensing and Policy Learning for Autonomous Vehicles. *arXiv preprint arXiv:2111.12083*.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). Paper Title. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3213-3223.

Ding, W., Xu, C., Arief, M., Lin, H., Li, B., & Zhao, D. (2022). A Survey on Safety-Critical Scenario Generation for Autonomous Driving – A Methodological Perspective. *arXiv preprint arXiv:2202.02215*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning*, 1-16.

Fujimoto, S., Hoof, H.V, & Meger, D. (2018). Addressing Function Approximation Error in Actor Critic Methods. *Proceedings of the 35th International Conference on Machine Learning*, 1587-1596.

Geiger, A., Lenz, P., Stiller, C., & Urtsun, R. (2013). Vision Meets Robotics: The KITTI Dataset. *International Journal of Robotics Research*.

Han, I., Park, D.H, & Kim, K.J (2021). A New Open-Source Off-Road Environment for Benchmark Generalization of Autonomous Driving. *IEEE Access, 9*, 136071-136082.

Hendriks, F., Tideman, M., Pelders, R., Bours, R., & Liu, X. (2010). Development Tools for Active Safety Systems: Prescan and Vehil. *Proceedings of 2010 IEEE International Conference on Vehicular Electronics and Safety*, 54-58.

Koopman, P, & Fratrik, F. (2019). How Many Operational Design Domains, Objects, and Events? *AAAI Workshop on Artificial Intelligence Safety*.

LeCun, Y., Bengio, Y., &. Hinton, G. (2015). Deep Learning. *Nature, 521(7553)*, 436-444.

Mania, P., & Beetz, M (2019). A Framework for Self-Training Perceptual Agents in Simulated Photorealistic Environments. *International Conference on Robotics and Automation (ICRA)*.

Michel, O. (2004). Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics, 1(1)*, 39-42.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 234-241.

Schaul, T., Quan, J., Antonoglou, I, & Silver, D. (2016). Prioritized Experience Replay. *4th International Conference on Learning Representations (ICLR*).

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing Properties of Neural Networks. *International Conference on Learning Representations*.

Thorn, E., Kimmel, S., & Chaka, M. (2018). A Framework for Automated Driving System Testable Cases and Scenarios. *Technical Report DOT HS 812 623, National Highway Traffic Safety Administration*.

Tuncali, C.E, Fainekos, G., Ito, H., & Kapinski, J. (2018). Simulation-Based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components. *IEEE Intelligent Vehicles Symposium (IV)*, 1555-1562.

Tuncali, C.E, Fainekos, G., Prokhorov, D., Ito, H., & Kapinski, J. (2020). Requirements-Driven Test Generation for Autonomous Vehicles with Machine Learning Components. *IEEE Transactions on Intelligent Vehicles, 5(2)*, 265-280.

Wang, P., Huang, X., Cheng, X., Zhou, D., Geng, Q., & Yang, R. (2019). The Apolloscape Open Dataset for Autonomous Driving and its Application. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., & Darrell, T. (2020). BDD100K: A Diverse Driving Dataset for Heterogenous Multitask Learning. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2633-2642.

Zheng, X., Liang, H., Yu, B., Li, B., Wang, S., & Chen, Z. (2020). Rapid Generation of Challenging Simulation Scenarios for Autonomous Vehicles Based on Adversarial Test. *IEEE International Conference on Mechatronics and Automation (ICMA)*, 1166-1172.