

Adversarial Scenario Generation for Virtual Testing of Off-Road Autonomous Vehicles

by

Ted Sender

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2025

Doctoral Committee:

Professor Bogdan I. Epureanu, Chair
Professor Tulga Ersal
Professor Xun Huan
Professor Ram Vasudevan

Ted Sender

tsender@umich.edu

ORCID iD: [0000-0003-4499-5807](https://orcid.org/0000-0003-4499-5807)

© Ted Sender 2025

DEDICATION

For my parents Abbie and Steve, my brother Jon, my late
grandmother Rose who survived the Holocaust, and my late
great Uncle Steve.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my PhD advisor Prof. Bogdan Epureanu for being there for every step of the way throughout my PhD journey. Prof. Epureanu deeply cares about all of his students, and I am grateful for his kindness, empathy, and eagerness to help. I could not have asked for a better advisor. I would also like to thank Prof. Ram Vasudevan for acting as an unofficial co-advisor and for his continued guidance and support throughout this entire project. I would like to thank the Automotive Research Center (ARC) at UofM and the U.S. Army Ground Vehicle Systems Center (GVSC) for sponsoring this research project (through Cooperative Agreement W56HZV-19-2-0001). Being part of an ARC-sponsored project, I would like to thank the individual members from the government and industry partners who attended the weekly research meetings and provided valuable feedback to guide the project's directions: Dr. Mark Brudnak (GVSC), Dr. John Brabbs (GVSC), and Reid Steiger (Ford Motor Company). Mark especially provided invaluable guidance on the type of adversarial scenarios that would be most useful to the army. I would like to thank postdoc Dr. Elena Shrestha for her guidance and direction in our brief time together. Elena introduced me to the RL papers that eventually lead me into developing the BACRE algorithm. I would like to thank my colleagues Dr. Congkai Shen, Dr. Siyuan Yu, and James Baxter for providing me with their simulation setups so I could perform the BACRE-3D experiments with them. I would like to thank everyone in the Epureanu Research Group for helping me navigate the challenging times as a PhD student and for their valuable feedback on my presentations. I especially want to thank Dr. Sean Kelly, Dr. Haochen Wu, and Mihai Cimpuiaru for always lending an ear and offering help whenever I was stuck. Finally, I would like to thank my family and friends for their love and support throughout my academic endeavors. This has been my most challenging academic journey and I thank everyone who has helped me along the way.

TABLE OF CONTENTS

Dedication	ii
Acknowledgments	iii
List of Figures	vi
List of Tables	x
Abstract	xi
Chapter	
1 Introduction	1
1.1 Motivation	1
1.2 Contributions and Dissertation Outline	6
2 A Regret-Informed Evolutionary Approach for Generating Adversarial Scenarios for Black-Box Off-Road Autonomy Systems	10
2.1 Introduction	10
2.2 Off-Road Scenario Decomposition	13
2.2.1 What is a Scenario?	13
2.2.2 Off-Road Scenario Attributes	14
2.3 Off-Road Simulation Platform	16
2.3.1 Platform Overview	16
2.3.2 Scenario Configuration	18
2.4 Black-Box Adversarially Compounding Regret Through Evolution	19
2.4.1 Regret for General Navigation Tasks	22
2.4.2 Identifying Fit Parents	23
2.4.3 The BACRE Algorithm	24
2.5 Numerical Experiments and Results	26
2.5.1 Autonomy System	27
2.5.2 Experiment Parameters and BACRE Configuration	32
2.5.3 Iterative Design Procedure: Results and Discussion	36
2.5.4 BACRE Comparison: Results and Discussion	44
2.6 Conclusions	46
3 An Evolutionary Approach for Generating 3-Dimensional Adversarial Scenarios for Black-Box Off-Road Autonomy Systems	49

3.1	Introduction	49
3.2	BACRE-3D	50
3.2.1	Improving Search Efficiency	51
3.2.2	Avoiding Local Maxima	59
3.3	Methodologies for Defining Mutations and Regret for Off-Road AVs	63
3.3.1	Generating 3D Terrain Mutations	63
3.3.2	Regret	69
3.4	Numerical Experiments and Results	71
3.4.1	Region of Influence: Validation and a Comparison	71
3.4.2	BACRE-3D vs. TPP	79
3.4.3	BACRE-3D vs. MPC	84
3.5	Conclusions	88
4	Conclusions, Summary of Contributions, and Future Work	91
4.1	Conclusions	91
4.2	Summary of Contributions	92
4.3	Future Research Directions	93
	Bibliography	95

LIST OF FIGURES

FIGURE

1.1	A simplified general development and testing cycle that researchers and engineers follow when creating AVs (or any other piece of technology). The red box indicates the focus of this dissertation. The dashed arrow connecting the two blocks indicates that some of the underlying development challenges can be used to influence how the system is tested.	2
2.1	Workflow for interacting with the AutomaticSceneGeneration plugin and ROS2 interface.	17
2.2	Overview of BACRE applied to off-road AVs. In each iteration, a candidate scenario is selected from the current population and one or more attributes are modified while ensuring the reference trajectory is feasible (a new set of modifications must be made to the original scenario until the reference trajectory is valid). Once the prescribed scenario is simulated, its regret is evaluated and the scenario-regret combination is added to the population.	22
2.3	A real (left) and virtual (right) Polaris MRZR vehicle. Left: An MRZR vehicle used by the Automotive Research Center at the University of Michigan. Right: A rendering of an MRZR as used in the UE4 simulations.	27
2.4	A* traversability autonomy stack used in the numerical experiments. The color image and semantic segmentation image shown were generated using the AutomaticScene-Generation plugin.	28
2.5	A rendering of a Barberry bush and Juniper tree as used in the UE4 simulations in this chapter.	33
2.6	Three separate snapshots of AV 1 path planner's state for the maximum regret scenario in iteration 1 with BACRE. The vehicle and all obstacles are indicated by point markers (their actual size is not shown). The red dots coincide with the A* planner's obstacle grid, and the intensity of the red indicates how many votes (as a fraction of $v_{\max}$) are assigned at that vertex. At each of these time instances, the most recently generated path that the vehicle should begin following is shown. A direct path and its associated cost is also shown for validation that the A* path is reasonable.	39
2.7	Three separate snapshots of the internal state for the path planner and perception system in AV 2 for the the maximum regret scenario in iteration 2 with BACRE. Time increases from top to bottom. The images on the left show the path planner state. The images on the right show the raw camera image and the predicted semantic segmentation image with the labels traversable (white), non-traversable (black), and sky (blue).	41

2.8	Three separate snapshots of AV 3 path planner's state for the maximum regret scenario in iteration 3 with BACRE.	43
2.9	Partial sub-tree for the max regret scenario in iteration 1 in Table 2.3. "Level" refers to the tree level and "Iter" refers to the overall iteration that generated the scenario. Dimensions are in meters.	44
2.10	Regret learning curves from the comparison experiment. Shown are the mean and standard error over five trials. Smoothing is used in the left plot of (a) with a moving average window of size 100.	46
2.11	Maximum regret scenarios from the comparison experiment.	47
3.1	A parent-child scenario combination from one of the experiments in Section 2.5. The child scenario has an added tree roughly 30 meters to the right of the start location which does not alter the vehicle's behavior. Each figure shows a front-aerial view of the scene (upper-left), a third-person view from behind the starting position (upper-right), and an orthographic-aerial view (lower) overlaid with additional information.	51
3.2	A comparison of how the BACRE variations construct the scenario population, represented as a tree, using the optimizations presented in this chapter. The maximum regret scenario is not shown in the left or middle diagrams because the goal is to find a failure scenario. In the BACRE-3D diagram, the failure scenario created by the dashed blue line is only present in the final population. Furthermore, the green scenario shown may or may not correspond to the maximum regret scenario in that final population.	52
3.3	An illustration of the ROI mutation mechanism. In this example, the action is to add a bush to the parent scenario shown. To create a mutation, a control point c_1 (black dot) is first selected from part of the parent vehicle path (cyan) that lies outside the obstacle-free radii. The cyan circle centered at c_1 shows the search region from which an action location can be chosen from. The magenta dot shows the selected action location c_2 , and the black line from c_2 to the parent vehicle path represents the action distance. The vehicle paths from the resulting child scenario are shown in red.	59
3.4	A visualization of the tool's local strength function $f_{\text{tool}}(r; \gamma, R)$ for some $R > 0$ and $0 < \gamma < 1$	65
3.5	The circular-based elementary sculpting tools.	66
3.6	Elementary rectangular sculpt tool. The points a and b have a z-component which is not shown. The endcaps are optional.	67
3.7	Sample mutations from each type of elementary sculpt tool applied to a 20 meter $\times$ 20 meter area. The color bar denotes the height perturbation in meters.	67
3.8	An illustration of a pseudo-local maxima (1D case). Consider the function $f(x) = -x^4 + x^2 + 2$ with zeros at $x = \pm\sqrt{2}$. Although $(0, 2)$ is a local minima, it can be considered a pseudo-local maxima with $r = \sqrt{2}$ and $h = 2$ (this is just one of many (r, h) pairs that work for this point).	69

3.9	Shown are the three scenarios from the ROI validation experiment and several visualizations of how adding a single tree to each of these scenarios changes the regret based on the tree's location. Top: The original scenario with the vehicle trajectories. Bottom: Shows a partial brute-force visualization of how the regret changes when a tree is added within the set $S_a^{xy}(0.5, 10, \theta)$. The hatched white region denotes the "unknown", as this area was not tested. The gray areas represent an infeasible placement of the tree due to the feasibility constraints. In the colorbar, $\Delta R = \hat{R}(a(\theta, s)) - \hat{R}(\theta)$ indicates the regret difference between the child scenario $a(\theta, s)$ and the original/parent scenario θ . The region labeled $\Delta R \geq 0.1$ represents $\hat{S}_{ROI+}^{xy}$ with $\epsilon = 0.1$, and the two regions $\Delta R \geq 0.1$ and $\Delta R \leq -0.1$ together form $\hat{S}_{ROI}^{xy}$ with $\epsilon = 0.1$	73
3.10	These plots show the average distance a point from $\hat{S}_{ROI+}^{xy}$ (left) and $\hat{S}_{ROI}^{xy}$ (right) is from the vehicle trajectories T_θ^v for the three scenarios in the ROI validation experiment. Distances are shown as a function of search radius for various regret thresholds of ϵ , which are used to construct the two ROI sets.	74
3.11	These plots show the fraction of points within the <i>feasible</i> subset of the conservative ROI approximation $U(T_\theta^v, r)$ that belong to $\hat{S}_{ROI+}^{xy}$ (left) and $\hat{S}_{ROI}^{xy}$ (right) for the three scenarios in the ROI validation experiment. The feasible subset of $U(T_\theta^v, r)$ is shown by the green, beige, and black points combined in the bottom figures in Fig. 3.9, and the infeasible subset is shown in gray.	75
3.12	Regret learning curves from Section 2.5.4 plus the versions of BACRE with ROI. Shown are the mean values over five trials. Smoothing is used in the left plots (regret vs. iterations) with a moving average window of size 100. The standard error is omitted to improve readability,	77
3.13	Shown are the average iterations needed for the methods in the comparison experiment to create a failure scenario. The x-axis only pertains to the BACRE with ROI variations, which are denoted by the colored X's, to help identify the (r_0, L) parameter combination. The methods tested in Section 2.5.4 are denoted with horizontal lines.	78
3.14	This plot shows how the controlling search radius changed over time for the nine different (r_0, L) combinations. The solid line represents the mean value over the five trials, with the shaded region depicting the standard error.	78
3.15	Shown are the maximum regret scenario and corresponding regret learning curve from two of the BACRE with ROI variations that were tested. Top: Maximum regret scenario. Bottom: Corresponding regret learning curve. The (green) axis on the right of the plot shows how the controlling search radius (also in green) changed over time.	79
3.16	Summary plots from the experiments with TPP. Left: The evolution of the maximum regret attained vs. the number of search iterations. Right: A breakdown of how many scenarios were in each tree level in the latest population. (b) Data from previous populations are shown in dashed lines, but these are only shown in the left plot.	82
3.17	The maximum regret scenarios from two of the BACRE with ROI vs. TPP trials with the vehicle and reference trajectories overlaid.	82

3.18	Shown are the failure scenarios and the corresponding tire steering/force/moment data from three of the BACRE-3D vs. TPP trials. Top: Various data about the tires are shown as the front right tire rotates from 2% to 75% of its final angle. The top plot shows the steering angles, with the horizontal dashed lines marking the steering range, and the bottom plot shows the tire force F_x in the local x-axis and the moment M_z about the spindle's local z-axis. In the legend, "L" and "R" denote the left and right tires, respectively. Bottom: The failure scenario with the vehicle and reference trajectories overlaid.	84
3.19	Renderings at two snapshots of the vehicle's right side from trial 2 in the BACRE-3D vs. TPP experiment. These snapshots occur around the time 7.6 seconds in the tire plot in Fig. 3.18a, and the front right tire has clearly rotated about 90 degrees at time t_2	84
3.20	Summary plots from the experiments with ATMPC. Left: The evolution of the maximum regret attained vs. the number of search iterations. Right: A breakdown of how many scenarios were in each tree level in the latest population. (b) Data from previous populations are shown in dashed lines, but these are only shown in the left plot.	86
3.21	Shown are three failure scenarios from the BACRE with ROI vs. ATMPC experiment. Top: A summary of the failure scenario showing the terrain height with the vehicle and reference trajectories overlaid. Middle: The linear acceleration about the vehicle's z-axis and angular acceleration about the vehicle's x-axis from one of the five vehicle runs. The magnitude shows how rugged the terrains are. Bottom: A rendering of the vehicle in the area that was most challenging for it to navigate. The vehicle is stuck on the slopes in the trial 1 and trial 3 renderings.	87
3.22	Shown are three failure scenarios from the BACRE-3D vs. ATMPC experiment. Top: A summary of the failure scenario showing the terrain height with the vehicle and reference trajectories overlaid. Middle: The linear acceleration about the vehicle's z-axis and angular acceleration about the vehicle's x-axis from one of the five vehicle runs. The magnitude shows how rugged the terrains are. Bottom: A rendering of the vehicle in the area that was most challenging for it to navigate. The vehicle is rolling over in all of the renderings shown.	88

LIST OF TABLES

TABLE

1.1	Comparison of the level of difficulty for developing an AV in different types of environments. Note that we do not distinguish between the degree of hardness here for the cases labeled as “hard”	4
2.1	DNN architecture for semantic segmentation network. All activation functions with Leaky RELU use a leak rate of $\alpha = 0.2$. All convolutional layers use an L2 kernel regularizer with penalty $\lambda = 0.001$, and all weights are initialized according to the He initializer method [1]. There are a total of 587,123 trainable weights in the network. In the “Name” column, the prefix “e” indicates an encoder block and the prefix “d” indicates a decoder block.	29
2.2	Probability of each possible action when modifying a structural attribute of a parent scenario, assuming an obstacle class “class” has already been chosen.	34
2.3	Results from the iterative design experiment. Shown are the Base AV and its problems, the Test AV and its improvements, the max regret scenario for the Base AV, and the associated regret learning curves. The max regret scenario shows a front-aerial view of the scene (upper-left), a third-person view from behind the starting position (upper-right), and an orthographic-aerial view (lower) overlaid with the paths from both the Base AV and the improved Test AV. In the regret learning curves, the blue curves for regret have been smoothed using a moving average window of size 100.	37
2.4	Additional metrics from the comparison experiment.	46

ABSTRACT

Developing autonomous vehicles (AVs) that operate in diverse and demanding environments is a difficult challenge. These vehicles naturally have advanced vehicle dynamics, and to rise to the environment’s demand, they consequently are controlled by sophisticated autonomy systems, which often consist of interconnected perception, navigation, and control systems. Understanding the weaknesses and limitations of these complex systems is crucial for understanding what they can and cannot do safely, properly, or at all.

Simulation has become a valuable tool for AV testing because it is cheaper, easier, and safer than physical testing. This work focuses on autonomous ground vehicles, and the two classes of such vehicles are on-road/urban AVs and off-road AVs. On-road AVs operate in highly structured environments, but must deal with the uncertainty of what the surrounding dynamic actors (vehicles, pedestrians, etc.) are doing. On the other hand, off-road AVs must operate across a wide variety of highly unstructured and mostly static environments that consist of rugged (and potentially hazardous) terrain whose ground surface is often made of multiple non-uniform materials (e.g., soft-deformable soil, water, foliage, etc.). Much of the existing work for AV development and testing pertains to on-road/urban AVs and is thus centered around the challenges that plague on-road AVs (i.e., uncertainty of the dynamic actors). However, the off-road AV community is slowly growing and is deserving of similarly advanced tools to meet their development and testing needs.

This dissertation focuses on developing one of the most important types of testing tools for off-road AVs, an area that is mostly non-existent. We specifically focus on the creation of an adversarial algorithm that can efficiently create scenarios that highlight weaknesses or limitations in the autonomy system. We propose a method called Black-Box Adversarially Compounding Regret Through Evolution (BACRE), which is an evolutionary algorithm guided by a novel regret-

based metric for general navigation tasks. A black-box approach is often preferable when system complexity can be diverse, like with off-road AVs, and when whole-system testing is required. A custom simulation platform is also provided to assist with the automated testing of AVs in unstructured environments. We start by presenting a base version of BACRE and perform numerical experiments to demonstrate its potential. We then address several of BACRE’s limitations and extend it to the case of generating adversarial 3-dimensional terrains, resulting in the BACRE-3D algorithm. The numerical experiments show that BACRE-3D is capable of pushing the entire simulation platform to its limits; BACRE-3D finds failure cases that show the chosen simulation platform is incapable of properly modeling scenarios it was expected to be capable of modeling, and it finds scenarios that push a state-of-the-art off-road trajectory planner to its limits.

In summary, this dissertation fills a void in the (meager) off-road AV testing literature by showing how to break down the intractable problem of finding performance limitations of off-road AVs in high-dimensional, static, unstructured environments.

CHAPTER 1

Introduction

1.1 Motivation

Autonomous ground vehicles (AGVs, or just AVs in this dissertation) have the potential to benefit society in numerous ways. When most people think of an AV, they generally envision an automated vehicle, like a car, bus, or truck, driving on city roads that can safely travel from one point to another. These vehicles are referred to as *urban*, or *on-road* AVs. Although much of the existing interest and research for AGVs pertains to on-road AVs, there is a growing interest and need for AVs that can operate in *unstructured* environments with various useful applications such as military operations, planetary exploration, search and rescue missions, agriculture, and mining [2]. These vehicles are referred to as *off-road* AVs. As AV technology advances, there is always the lurking question of “what does it take to cause an AV to fail?” Understanding an AV’s limitations is crucial for ensuring its safe and effective operation. This question lies at the heart of this dissertation, and more specifically, with regards to off-road AVs.

When we develop AVs, we generally follow a continuous development and testing cycle in multiple passes, such as the one depicted in Figure 1.1. On the development side, we address various questions such as “what assumptions can we make?”, “what physical design and materials are appropriate?”, “what electronics and hardware do we need?”, “what models and algorithms should the autonomy system use?”, etc. The output of the development stage is then a physical or virtual model that we must then test. In the testing stage, the two main questions we address are “what are we testing?” and “how do we test it?”. The output of the testing stage are usually

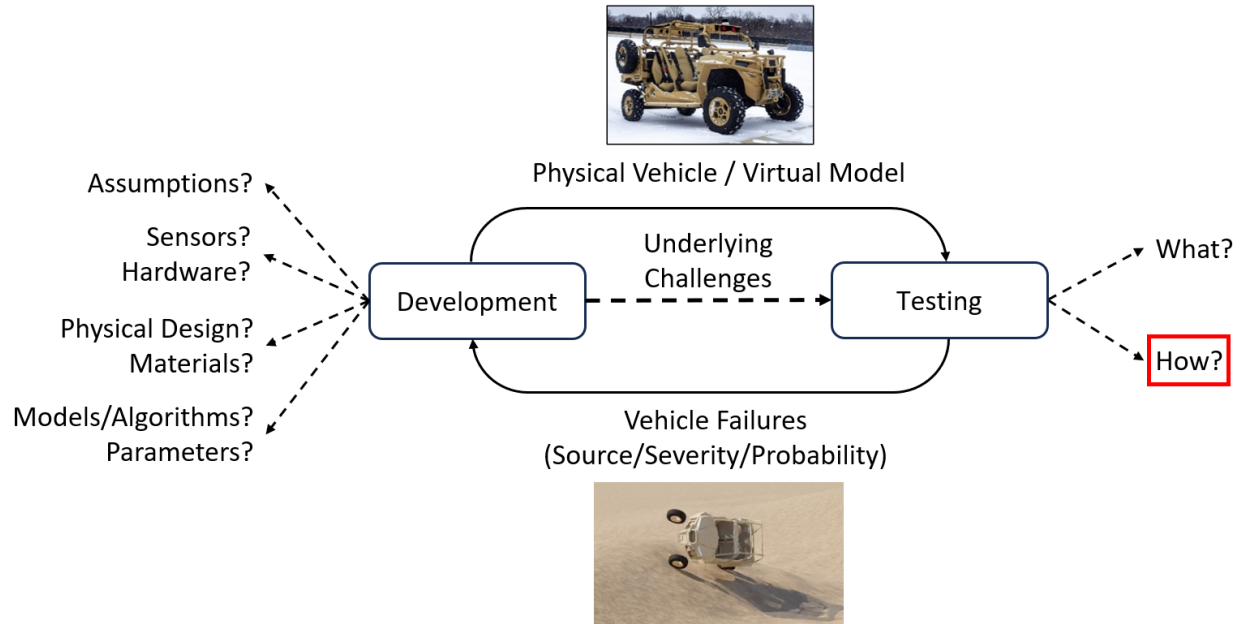


Figure 1.1: A simplified general development and testing cycle that researchers and engineers follow when creating AVs (or any other piece of technology). The red box indicates the focus of this dissertation. The dashed arrow connecting the two blocks indicates that some of the underlying development challenges can be used to influence how the system is tested.

the vehicle failures, often including their source, severity, and/or probability. Other statistical analyses may also be produced, but the failures are of most concern in this work. This process is then repeated until the design requirements are satisfied. Despite the simplicity of the diagram in Figure 1.1, both the development and testing phases are far from trivial and face their own set of challenges.

The key challenges in AV development revolve around the battle between autonomy and uncertainty. An AV must be able to navigate the dynamics of its operating environment for it to be successful. Managing uncertainty, augmented with environment complexity and variability, is arguably the greatest development challenge. Fundamental to on-road AVs is the unpredictable behavior of dynamic actors, and fundamental to off-road AVs is the large variation of natural, unstructured, static environments. Yet another challenge under this theme pertains to vehicle dynamics. Vehicles have complex mechanical dynamics and with tire-ground interactions. However, off-road environments create a significantly harder task for modeling tire-ground interactions due

to soft-deformable soils, rugged terrain, water bodies, foliage, etc. On-road AVs are assumed to be driving on asphalt or concrete which is comparatively easier to model. Furthermore, advancements in deep learning (DL) [3] and reinforcement learning (RL) [4] have allowed for AVs to rely on deep neural networks (DNNs) [5] for use in the autonomy system. However, DNNs are susceptible to subtle variations in their inputs [6].

There are several approaches for AV testing. On the extremes there are physical / in-field testing and simulation, and then there are hybrid approaches of the two, such as hardware-in-the-loop (HIL) [7] and vehicle-in-the-loop (VIL) [8, 9], which attempt to mitigate the cons of one extreme with pros from the other. In-field testing is great for validation, but it is very expensive and time-consuming, and it can pose safety concerns depending on the nature of the test. Simulation is cheaper, easier, faster, and safer than in-field testing. Accordingly, simulation makes it easier to identify scenarios that cause the AV to fail, find the AV's weaknesses, and estimate the probability of failure. However, for simulation to truly be useful, we must have capable and representative platforms that appropriately model the static and dynamic components in the environment. Additionally, determining how to find the scenarios that cause an AV to fail, the AV's weaknesses, etc., are not trivial questions. These questions tend to serve as common research questions.

The nature of the operating environments (which can be static or dynamic) and the uncertainty they present for on-road and off-road AVs is an important topic worth reiterating. On-road AVs operate in highly structured environments (i.e., roads and highways with lane markings, traffic signals, and traffic rules) but must deal with the uncertainty of all of the other vehicles and pedestrians nearby as they navigate from one point to another. On the other hand, off-road AVs mostly operate across a wide variety of highly unstructured, static environments that consist of rugged (and potentially hazardous) terrain along with various ground textures (soft-deformable soil, water, foliage, etc.). In a military context, we may also want off-road AVs to work in teams with other AVs and/or humans, which adds another level of complexity. Table 1.1 summarizes the level of difficulty for developing AVs. We observe that it is far more challenging to develop off-road AVs because it is significantly more challenging to operate in just a static environment. Since operat-

Table 1.1: Comparison of the level of difficulty for developing an AV in different types of environments. Note that we do not distinguish between the degree of hardness here for the cases labeled as “hard”.

Area	Type of Environment	
	Static	Dynamic
On-Road	Easy	Hard
Off-Road	Hard	Hard

ing in a static environment is a requirement for operating in a dynamic environment (for obvious reasons), the majority of the development and testing methodologies for on-road AVs focus on dynamic environments and for off-road AVs focus on static environments.

Over the past few decades, a large emphasis was placed on improving AVs for urban driving. The large number of existing datasets (e.g., [10–13]), software platforms (e.g., [14–18]), and adversarial testing algorithms highlight this. The on-road literature for adversarial AV test methods generally focus on creating safety-critical (high-risk) scenarios [19]. *Risk* commonly denotes a metric quantifying a vehicle’s level of failure. This metric generally includes quantities like minimum time-to-collision, distance to nearest actors, acceleration, jerk, etc. Scenarios in which the vehicle’s risk level exceeds an unsafe threshold are referred to as *safety critical* [19]. Reference [19] provides a comprehensive survey on these methods, although some of them are mentioned here. Since interactions with potentially unpredictable *dynamic* actors is a major challenge, many methods focus on adversarially controlling these actors’ behavior using various approaches, including: evolutionary algorithms [20, 21], Bayesian Optimization [22], multi-agent RL [23], generative modeling [24, 25], and adaptive stress testing [26, 27]. Signal temporal logic and simulated annealing have been proposed to identify faults in machine-learning-based autonomy systems [28, 29], but require direct access to every component being tested. Other methods perturb existing vehicle trajectories from previous datasets [17, 18, 30] to create novel viewpoints for training that are more challenging than the original ones. Another body of work within the realm of testing focuses on estimating the probability that a rare-event occurs. Rare events are events whose probability of occurrence is quite small, and typical thresholds are 0.01 or 0.001. In this context, rare events are

generally considered as those whose risk level exceeds a certain (or unsafe) threshold, and common approaches for finding these probabilities include rare-event simulation [31, 32] which utilize cross-entropy [33], importance sampling (IS) [34], and adaptive multilevel splitting [35] to generate the rare-event samples. Deep learning-based extensions of these methods have been shown to reduce variance when calculating the rare-event probabilities [36] and to increase efficiency [37].

While the aforementioned literature is not exhaustive of all on-road adversarial AV testing, it does show the breadth that exists in this application. In contrast, there only exist two comprehensive open-source datasets [38, 39], a few high-fidelity simulators (e.g., [40–44]), and one approach for evaluating driving policies [42] for off-road AVs. The most recent addition to this list is VertiBench [44], the first-ever, large-scale off-road mobility platform for benchmarking off-road AVs across a wide variety of off-road conditions. Considering the 20+ simulators and 20+ datasets listed in [19], the vast literature survey from [19], along with the works discussed in the previous paragraph, there is a clear need to expand simulation and testing capabilities for off-road AVs.

The lack of any standardized algorithms, procedures, or tools for testing off-road AVs is also of concern and is quite apparent throughout the literature. Some papers do not mention how their test scenarios are generated [2, 45]. Several papers use rather contrived terrain profiles (such as sinusoids with one or fewer frequencies) and/or place a handful of obstacles in contrived locations [46–48]. A few papers use mathematical methods (e.g., Weierstrass-Mandelbrot function, fractals) to generate random terrain profiles with at least a few hills/valleys [49–52]. And a few papers show real experiments validating the proposed novel methods [49, 53]. While this list is not exhaustive and only pertains to trajectory/path planning and control, the message is still clear: the off-road AV community is growing and deserving of tools dedicated to testing AVs in diverse environments to identify the system’s weaknesses and limitations. The same conclusion has also been reached by [54], which goes into greater depth about the challenges of developing and testing off-road AVs.

The discussion presented above highlights two important challenges that affect both the on-road and off-road AV communities:

1. The need for advanced simulation platforms that can create and conduct arbitrary driving

scenarios.

2. The need to develop algorithms that can test an AV’s performance and identify its limitations and weaknesses.

Although these challenges affect both communities in general, the actual approach for addressing these challenges must be different due to the fundamental differences between on-road and off-road AVs. The on-road AV community has clearly made significantly more progress at addressing these two challenges as demonstrated by the larger volume of available literature and software. However, the off-road AV community has only made progress at addressing the first challenge with creating capable simulators, as there is no known literature/tools available for off-road adversarial testing. Furthermore, the on-road AV test methods discussed above are not well-suited for off-road AV testing because: (1) they generally are not designed to create adversarial *static* environments, (2) they generally assume a fixed scenario dimension which limits test variability, and (3) limited off-road datasets make generative/probabilistic methods [24, 25, 31, 32, 36] difficult to apply. Consequently, the second challenge is also a *knowledge gap* for the off-road AV community and it is the central focus of this dissertation.

1.2 Contributions and Dissertation Outline

Following the motivation in Section 1.1, the research problem behind this dissertation is how to efficiently find the weaknesses and performance limits of an off-road AV operating (i.e., navigating) in a static, unstructured environment. Addressing this task in static environments is a critical knowledge gap in the community, and the methodologies in this dissertation are expected to serve as a foundation for when this task is extended to dynamic environments.

To find these weaknesses and limitations, we aim to find and create scenarios that *induce* such weaknesses in the AV’s capabilities. However, a complete solution to this problem must address the following sub-problems:

1. What is a scenario and how do we create them?

2. What do we mean by “weakness”?
3. How do we find these scenarios that induce weaknesses in the AV’s capabilities?

These questions form the basis of the research objectives in this dissertation:

1. Develop a framework to define and describe off-road scenarios.
2. Create a simulation platform to assist in the automated creation of diverse off-road environments and in the oversight of navigation tasks within those environments.
3. Develop an efficient, iterative algorithm that can evaluate an AV’s performance as well as propose and apply modifications to virtual scenarios to incrementally worsen the vehicle’s performance.

The first two research objectives lay the groundwork that allow the third objective to be realized.

The primary novel contribution from this dissertation is the iterative algorithm that addresses the third research objective. The baseline algorithm is called Black-Box Adversarially Compounding Regret Through Evolution (BACRE), and it is an evolutionary algorithm that uses regret as the objective function to maximize. The novelty within BACRE is in the use of a reference trajectory to compare the test system’s trajectory to and in a novel regret metric. An improved version of BACRE, called BACRE-3D, is developed as a more efficient and capable tool for generating 3-dimensional off-road scenarios. BACRE-3D is a novel extension of BACRE (with two novel algorithmic improvements upon BACRE) and is the main contribution of this dissertation. The secondary novel contribution is a custom simulation platform called AutomaticSceneGeneration that addresses the second research objective to a moderate extent. Although this simulation platform is not the focus of this work and can be improved in several areas, an algorithm like BACRE requires the ability to create and manage thousands of simulations autonomously. The AutomaticSceneGeneration platform achieves these goals for the purposes of this work, and it is intended to serve as an example for what the off-road community deserves.

The closest works to BACRE and BACRE-3D in the on-road literature are [20,21,28,29]. Each of these works contain similar elements/goals as in BACRE/BACRE-3D, but none are directly applicable to the off-road AV application. BACRE-3D is most similar to [20] which uses evolution to find feasible scenarios in which the test AV collides with nearby moving vehicles (e.g., on a highway or intersection). Their approach ensures a feasible trajectory exists by predicting a reachable set (on a flat roadway) and using that set in the objective function; this idea is similar to BACRE’s use of a reference agent. They also propose a method to reduce the search space by only modifying parameters of the actors that are near the test vehicle; this idea is similar to BACRE-3D’s ROI mutation method (to be discussed in Section 3.2.1), but specific for the on-road application. Overall, BACRE-3D operates similarly to the approach in [20], but optimizes over a much larger search space, allows for the scenario dimension to change, has an efficient search mechanism specific for off-road environments, *and* has a mechanism to get out of local maxima.

Reference [21] uses a quantum genetic algorithm to maximize a risk-based cost function to force the test vehicle into high-risk situations on a multi-lane road with several other vehicles. References [28, 29] use signal temporal logic and simulated annealing to identify scenarios that falsify strict requirements on component and system-level behavior. BACRE and BACRE-3D are specifically intended to be used with scenario dimensions that can *vary*, unlike [21] which poses a major restriction on the testing capabilities. A quantum genetic algorithm also does not have the ability to get out of a local maxima. The use of signal temporal logic in [28, 29] could possibly be used for off-road AV testing, but the large variation that occurs in off-road environments is expected to make it quite challenging to determine specific requirements on component and system-level behavior. Choosing a specific set of requirements could also restrict the complexity of the environments that can be tested.

The remainder of this dissertation is organized as follows. Chapter 2 addresses each of the three research objectives, with the ultimate goal being the development of the BACRE algorithm. BACRE is the first-ever adversarial algorithm designed specifically to find weaknesses in off-road AVs. However, the version of BACRE presented in Chapter 2.4 is rudimentary, and the experi-

ments and results only demonstrate effectiveness of the algorithm against an AV navigating on a flat ground plane. Chapter 3 discusses several limitations of BACRE, addresses all of them, and presents the BACRE-3D algorithm, the successor to BACRE. BACRE-3D addresses how to generate 3-dimensional terrain, provides an improved regret metric for off-road AVs, and addresses several algorithmic inefficiencies that allow it to be more sample efficient and prevent it from getting stuck in local maxima. The experiments with BACRE-3D are also more extensive and noteworthy as they demonstrate why such an algorithm is needed and how capable it is. Finally, Chapter 4 provides concluding remarks, summarizes the contributions of this dissertation, and presents directions for future work.

CHAPTER 2

A Regret-Informed Evolutionary Approach for Generating Adversarial Scenarios for Black-Box Off-Road Autonomy Systems

2.1 Introduction

The theoretical goal of this dissertation is to develop an efficient method for identifying weaknesses and limitations of black-box off-road autonomy systems. The phrase “weaknesses and limitations” can be interpreted in several ways. On-road AV test methods generally focus on finding high-risk scenarios, where *risk* commonly denotes a metric quantifying a vehicle’s level of failure pertaining to quantities like time-to-collision, distance to nearest actors, and acceleration [19]. Due to the difficulty of driving in unstructured, open environments, this dissertation is more interested in finding scenarios that are feasible (i.e., there exists a known, safe route to reach the goal for which a human would likely identify and follow) but difficult for the autonomy system to navigate and result in unexpected, undesirable, and/or unsafe behaviors. These scenarios can be considered as high-regret, in which *regret* refers to the difference between what happened and the optimal outcome under the same setting [55]. While high-risk and high-regret scenarios may overlap, they are *not* equivalent; some situations may be challenging/risky even for an optimal agent.

As part of our approach, we care about testing the system as a whole (for an AV, this would ideally constitute the physical vehicle along with its hardware, sensors, computers, and autonomy algorithms). This contrasts with methods that focus on finding adversarial attacks for a *single*

component of the autonomy system, such as introducing perturbations into the inputs of a deep neural network-based vision system [56, 57]. Although the goal is test the whole vehicle, all experiments take place in simulated environments because this is the easiest way to conduct many different scenarios. In the experiments conducted in this dissertation, a number of simplifying assumptions are also made about the test vehicle: hardware components are not modeled (and are ignored), sensors are not modeled after specific or real devices, and the vehicle dynamics may be simplified depending on the simulator used. The theoretical work proposed in this dissertation can be used with more accurate testing apparatuses that involve hardware-in-the-loop or vehicle-in-the-loop testing, but doing so can be costly, difficult to implement, and is beyond the scope of this work.

Because the adversarial testing literature in the on-road AV community does not apply to our specific application (as discussed at the end of Section 1.1), it would be wise to consider ideas from other related fields for guidance. Fortunately, there is no shortage of ideas in the autonomy field due to its heightened popularity in the modern robotics and artificial intelligence (AI) communities. Specifically, within the reinforcement learning (RL) community, there is a group of work that focuses on training generally-capable RL agents by means of creating a curriculum of environments tailored to the agent’s current skill level. This is effectively a two-step process that involves: (1) *finding* scenarios that are on the edge of the agent’s current capabilities, and (2) *retraining* the agent in those challenging environments to improve its overall capability. The current approaches utilize evolutionary strategies [58–60], curation strategies [61, 62], RL [63], or any combination thereof [62, 64]. Addressing step 1 is exactly the type of problem we wish to solve, but in the context of off-road AVs. Fortunately, there are several works under this RL topic with promising and relevant ideas.

The specific works of interest are the POET [59], PAIRED [63], and ACCEL [64] algorithms. POET [59] demonstrates the idea of creating a curriculum of environments that push the limits of the agent’s current capabilities. The authors use an evolutionary mechanism to create different environments, but use hard-coded reward ranges to determine if an environment is too easy or too

difficult and worth training on [59]. The authors in [63] formalize a framework called Unsupervised Environment Design (UED) for learning a distribution of the free parameters in an RL context that can achieve a similar goal. The authors develop PAIRED [63] as an approach to solving UED, and one of their key additions is the use of regret as an agnostic measure of the agent’s performance (avoiding the hard-coded approach taken by POET). They then show that maximizing regret incentivizes the adversary to create the simplest tasks that are “challenging but feasible” [63]. Finally, ACCEL [64] is a combined version of POET and PAIRED; it proposes a method for the UED framework using an evolutionary algorithm (inspired by [59]) with a regret-based curator (inspired by [61–63]).

The above three works inspire the design of the proposed algorithm Black-Box Adversarially Compounding Regret Through Evolution (BACRE), which is the algorithmic contribution of this dissertation. BACRE is an evolutionary algorithm that operates similarly to ACCEL, but is agnostic to the entire autonomy system by assuming it to be a black-box. The test system is further assumed to be fixed since the goal is to identify weaknesses of the existing system. These assumptions make BACRE applicable to testing a broader range of diverse and complex systems (which need not be RL-based), such as off-road AVs. The inspiration for evolution follows the rationale and effectiveness of it in POET and ACCEL to gradually increase environment complexity over time; this process thus allows us to observe how the vehicle’s performance degrades with successive mutations (assuming this mechanism can vary the number of obstacles). The latter is only possible with an evolutionary mechanism. The inspiration for regret comes from PAIRED. However, due to BACRE’s black-box assumption, evaluating regret cannot rely on access to the test agent’s internal “thoughts” like in [62–64]. Existing regret formulations in path planning literature (e.g., [65, 66]) are also not ideal because they assume the vehicle reaches the goal. An off-road vehicle may never reach the goal because it can get stuck, roll over, or get shot down (in a military context). Thus, a more general metric is needed.

This entire chapter is dedicated to laying the groundwork for the initial development and testing of BACRE. Despite the recent release of the Verti-Bench [44] platform for benchmarking off-road

AVs, it is still important to discuss the necessary ingredients to perform virtual off-road AV testing (especially *adversarial* testing). The required ingredients for this are: a framework for defining and describing off-road scenarios, a capable simulation platform, and the adversarial testing algorithm. Section 2.2 discusses the components and attributes that make up off-road scenarios. Section 2.3 discusses a custom simulation platform based on Unreal Engine 4 that is designed to assist in the automated creation of diverse off-road environments and in the oversight of navigation tasks within those environments. Section 2.4 describes the BACRE algorithm. Various numerical experiments along with their results and a discussion are presented in Section 2.5. Finally, Section 2.6 concludes the chapter with a summary of what was accomplished, the main findings, and presents some direction for future work.

2.2 Off-Road Scenario Decomposition

2.2.1 What is a Scenario?

The terms *scene*, *environment*, and *scenario* are important and related concepts that deserve standardized definitions because they describe the various conditions in which something occurs. In the context of testing robots (e.g., wheeled, aerial, underwater, etc.), it is important to understand the conditions in which the robot can perform satisfactorily as well as the conditions in which it cannot. To further complicate the matter, for example, it is possible that only changing the start and/or goal locations (without changing the surrounding environment) can affect the agent’s performance/behavior.

Existing literature may define one or more of these terms in the appropriate context of the application, but sometimes these definitions are simply implied. Reference [67] attempts to create a “new” standardized set of definitions for terms like *scene*, *situation*, and *scenario* as it points out that these terms have different, and even conflicting, definitions in existing literature. Since AV testing is an incredibly important challenge and task, the German government created the PEGASUS project [68] as a way to create a standardized set of testing and experimentation procedures

for simulated and real AVs. The definitions used by the PEGASUS project are based on those from [67], such as in [69], however, these definitions tend to focus on use-cases for on-road AVs. It is unclear if these definitions still make sense in the off-road use-case.

In this dissertation, we will not be concerned with following exact terminology from these sources because our ultimate goal is to create an adversarial testing algorithm. The goal of this section is to raise awareness for the need for consistent and encompassing terminology in the off-road AV context, but we will instead use the definitions given in the next paragraph.

Since a robot is typically given a task to complete in an environment with static and/or dynamic elements, it would make sense to decompose this “scenario” into its natural constituents. Specifically, the *scene* refers to the physical static operating environment as well as any relevant fluid mediums. For example, for an off-road vehicle the scene would consist of the terrain/landscape along with its various textures (e.g., grass, mud, sand), and obstacles (e.g., hills, trees, rocks, mountains). For an underwater robot, this would consist of the water (the fluid medium) and any static components like the sea floor, coral reefs, and trenches. The *environment* describes both the static and dynamic elements of the operating environment. The dynamic components included here consist of weather conditions and the surrounding actors (sometimes called non-player characters) whose behavior can be controlled. The *scenario* combines the static and dynamic environment descriptions with the task assigned to the vehicle/agent and any constraints that may be imposed on its behavior. Each of these entities (scene, environment, and scenario) is described by a set of various *attributes*, and these definitions create the following relationship between them:

$$\text{scene} \subset \text{environment} \subset \text{scenario}. \quad (2.1)$$

2.2.2 Off-Road Scenario Attributes

Currently, the only known framework for defining scenarios and their corresponding attributes for wheeled vehicles pertains to on-road vehicles. In 2018, the U.S. National Highway Traffic Safety Administration (NHTSA) developed the operational design domain (ODD) [70] 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 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 [71], this framework does provide a good starting point for defining off-road scenarios. The ODD framework proposes six high-level categories (physical infrastructure, operational constraints, objects, connectivity, environmental conditions, and zones), each with several subcategories, to define the relevant attributes in on-road scenarios [70]. While there is overlap between some of these categories with off-road driving, in lieu of the proposed scene-environment-scenario breakdown, it would make more sense to develop a set of broader categories (influenced by the ODD categories) that can be applied to each of the scenario components as required. Specifically, this work proposes three primary attribute categories for defining a scenario: *structural attributes*, *textural attributes*, and *operational attributes*. The level of detail required to define the associated attributes is case-dependent. The paragraphs below describe each category in the context of off-road scenarios and provide some examples. These categories will be revisited throughout this work to specify the appropriate level of detail for the given experiments.

Structural attributes define the geometry of all the structural objects that compose the scene. This includes the ground surface, static obstacles (e.g., trees, bushes, rocks), water bodies, etc. In a general manner, static obstacles are referred to 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 surface).

Textural attributes enhance the visual realism by describing visual effects of the environment. These attributes include describing the intensities of various weather/lighting conditions, as well as modifying the textural appearance of the ground surface (e.g., specifying which regions are grassy, muddy, etc., as well as the intensity of that texture) or even the SSAs.

Operational attributes describe operational constraints on any vehicle's behavior within the

environment (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).

2.3 Off-Road Simulation Platform

Adversarial test methods require running many different scenarios, hence they are best paired with automated simulation platforms that can: (1) quickly create arbitrary scenarios, and (2) monitor and analyze the according navigation task. We developed a platform called AutomaticSceneGeneration, consisting of a UE4 [72] plugin¹ and a ROS2 interface², to provide these capabilities for the off-road community. Since presenting a beta version in [73], the automation features and user interface were improved, both Python and C++ ROS2 vehicle nodes (which make up the autonomy system) are now supported, and networking features were upgraded to enhance the multi-computer simulation workflow. We also extended the UE4 plugin ROSIntegration from [74] to provide ROS2 support. While the platform only supports placing arbitrary static actors (e.g., trees, bushes) on a flat ground plane, such open field environments are still useful for studying the behavior of an autonomy system in general navigation tasks where multiple good/acceptable paths may exist. The next two subsections discuss this simulation platform.

2.3.1 Platform Overview

This section describes the general operation of the custom simulation platform. The base platform for simulation is UE4 [72], for which a custom plugin called AutomaticSceneGeneration along with a ROS2 interface are added to automate the entire scenario creation, execution, and analysis pipeline.

Figure 2.1 shows the typical workflow for interacting with this platform. There are three main components: (1) Unreal Engine (which includes the plugin), (2) the autonomy system under test

¹UE4 Plugin: <https://github.com/tsender/AutomaticSceneGeneration>

²ROS2 Interface: https://github.com/tsender/auto_scene_gen

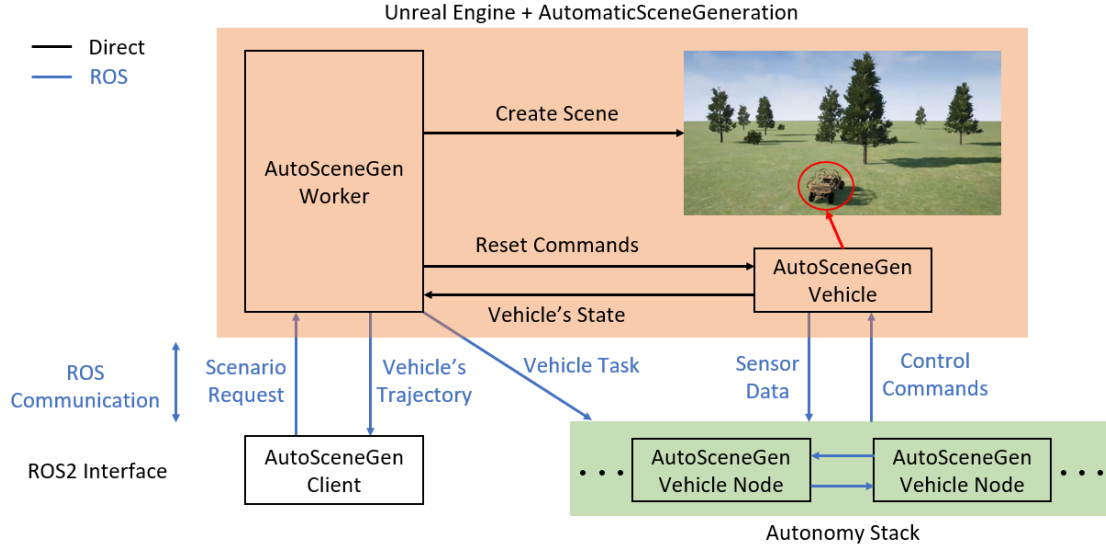


Figure 2.1: Workflow for interacting with the AutomaticSceneGeneration plugin and ROS2 interface.

(consisting of ROS2 nodes), and (3) an external client (another ROS2 node). The platform is configured to work as a server-client model with the AutoSceneGenWorker as the server (also referred to as a “worker”) and the AutoSceneGenClient as the client. The general process is as follows:

1. The client sends a RunScenario request to the worker describing all of the scenario’s attributes (including the environment and navigation task).
2. The worker processes the RunScenario request, creates the scene, places the vehicle (an AutoSceneGenVehicle actor) at the start pose, and relays the navigation task (goal location) to the autonomy stack.
3. The autonomy stack, consisting of one or more AutoSceneGenVehicleNodes, processes sensor data and sends control commands to the vehicle to reach the goal location.
4. Once the vehicle receives its first control command, the worker continuously monitors the vehicle’s progress. Once a termination condition is triggered, the worker will reset the vehicle and send an AnalyzeScenario request to the client containing information about the

vehicle’s trajectory and performance.

5. The client processes the AnalyzeScenario request, and can then initiate subsequent scenarios as desired by following the same procedure until it is done running tests.

2.3.2 Scenario Configuration

This section describes the supported scenario configurations for the AutomaticSceneGeneration plugin using the off-road scenario decomposition framework formalized in Section 2.2.

Vehicle failure is defined and enforced to prevent long-running simulations for vehicles navigating in open-field environments. Vehicle failure is said to occur if:

- It fails to reach the specified goal radius within time T_{sim} .
- It is *idling* (commanding near-zero velocity while undergoing near-zero velocity) for more time than T_{idle} .
- It is *stuck* (commanding non-zero velocity while undergoing near-zero velocity) for more time than T_{stuck} .
- It’s roll (or pitch) exceeds an angle θ_{maxroll} (or θ_{maxpitch}).

Otherwise, the vehicle is considered to have succeeded.

The following list organizes all available scenario parameters according to attribute type:

- **Structural Attributes:** The landscape that obstacles can be placed in is $L \times L$. There can be at most N_{class} instances per obstacle class, where “class” refers to any specific obstacle type (e.g., tree or bush). For the i th obstacle instance, the location $p_i^{\text{class}} \in \mathbb{R}^2$, yaw angle $\psi_i^{\text{class}} \in \mathbb{R}$, and mesh scale factor $s_i^{\text{class}} > 0$ can be set. Last, obstacle-free radii of $r_{\text{obs-free}}^S$ and $r_{\text{obs-free}}^G$ are imposed around p_S and p_G .
- **Textural Attributes:** The ground plane can have a user-defined texture throughout. The only offered weather condition is a clear sky with the sun at any position by setting the yaw angle $\psi^{\text{sun}} \in \mathbb{R}$ and inclination angle $\theta^{\text{sun}} \in \mathbb{R}$.

- **Operational Attributes:** The vehicle starts at $p_S \in \mathbb{R}^2$ with initial yaw angle $\phi_S \in \mathbb{R}$ and must reach the goal radius r_{goal} about the goal location $p_G \in \mathbb{R}^2$. Finally, the parameters related to vehicle failure (T_{sim} , T_{idle} , T_{stuck} , θ_{maxroll} , and θ_{maxpitch}) also fall under this category.

2.4 Black-Box Adversarially Compounding Regret Through Evolution

This section presents the BACRE algorithm, an evolutionary method that incrementally maximizes regret against a fixed black-box system to ultimately find scenarios that highlight the system’s weaknesses and/or suboptimality. The primary goals we have with BACRE are to find scenarios that: (1) highlight what the easiest thing is in our autonomy system that can “break”, (2) demonstrate flaws in the system design or inconsistencies in what the system is expected to be able to achieve, and (3) push the system to its performance limits.

Regret requires comparing the agent’s actions to that of an optimal reference. The test agent/vehicle is denoted A_v and the optimal reference agent is denoted A_* . Since the test agent is a black-box, BACRE can only observe the agent’s trajectory in a scenario. Let a trajectory

$$\tau := ((t_0, x_0), \dots, (t_n, x_n)) \in T^d$$

be a finite sequence of $n + 1$ time-state tuples (t_i, x_i) , where $t_i \in \mathbb{R}_+$ (the set of nonnegative real numbers), $x_i \in \mathbb{R}^d$, $t_i < t_j \forall i, j \in \{0, 1, \dots, n\}, i < j$, $n \in \{0, 1, 2, \dots\}$, and T^d is the set of all trajectories with state dimension d . The recommended reference agent is an optimal, idealized agent with perfect knowledge, navigation, and maneuverability. The optimal reference trajectory

for a scenario θ is governed by the numerical solution to the optimization problem

$$\begin{aligned}
\tau_\theta^* &= \underset{\tau \in T^d}{\operatorname{argmin}} && f_c(\tau; \theta) \\
&\text{subject to} && (\operatorname{pos}(x_0), \operatorname{ang}(x_0)) = (p_S, \phi_S) \\
&&& \operatorname{pos}(x_n) = p_G \\
&&& g_{\text{in},i}(\tau; \theta) \leq 0, \quad i = 1, \dots, n_{\text{in}} \\
&&& h_{\text{eq},j}(\tau; \theta) = 0, \quad j = 1, \dots, n_{\text{eq}}.
\end{aligned} \tag{2.2}$$

That is to say that (2.2) governs the behavior of the agent A_* . $f_c \in C^d$ denotes the cost function to minimize, parameterized by θ , where C^d is the set of all cost functions $f : T^d \rightarrow \mathbb{R}_+$ for which $f(\tau') \leq f(\tau) \quad \forall \tau' \subseteq \tau \in T^d$. The first two constraints ensure that τ_θ^* has starting position p_S , starting orientation ϕ_S , and final/goal position p_G as defined in θ (final orientation is ignored in this work), where $\operatorname{pos}(x)$ and $\operatorname{ang}(x)$ extract the linear and angular position. The remaining inequality and equality constraints, $g_{\text{in},i}$ and $h_{\text{eq},j}$, are parameterized by θ and denote any path-planning preferences to make the problem more tractable (for example, shortest-path grid-search constraints). Existence of a solution to (2.2) is generally assumed in this paper, but non-existence is accounted for. Given a trajectory τ and a cost function $f \in C^d$, the corresponding *cost-aware trajectory* (CAT) is $\bar{\tau}_f := ((t_0, c_0, x_0), \dots, (t_n, c_n, x_n))$, which adds a cumulative cost element $c_i = f(\tau_{0:i})$, $i \in \{0, 1, \dots, n\}$ where $\tau_{0:i}$ is the sub-trajectory in τ from time t_0 to t_i . The subscript f on $\bar{\tau}_f$ may be omitted for clarity.

BACRE uses *black-box regret*, denoted $r(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*)$, which is a function of the scenario θ , the vehicle's CAT $\bar{\tau}_\theta^v$, and the idealized optimal CAT $\bar{\tau}_\theta^*$ (conventional regret where $\bar{\tau}_\theta^v$ and $\bar{\tau}_\theta^*$ use the same autonomy model is not possible in the black-box setting). Unfortunately, black-box regret, including the proposed metric in (2.8), can at best capture a performance difference between $\bar{\tau}_\theta^v$ and $\bar{\tau}_\theta^*$. Consequently, care should be taken when formulating (2.2) because high regret generally indicates poor test agent (or reference agent) performance when (2.2) is well-defined (or ill-defined if it generates poor trajectories and/or penalizes good ones). Despite this limitation, the latter case is not expected to occur often, and maximizing $r(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*)$ with (2.2) well-defined should

generally be conducive to finding suboptimalities of the test agent’s autonomy system. Both $\bar{\tau}_\theta^v$ and $\bar{\tau}_\theta^*$ start with zero initial time, and both use f_c from (2.2) when computing the c_i terms since f_c is the optimal reference cost function. Furthermore, in case of non-determinism in the scenario, simulator, and/or the test agent’s autonomy system, the *true* regret, defined as

$$R(\theta, A_v, A_*) = \mathbb{E}[r(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*)], \quad (2.3)$$

will be unknown. Consequently, BACRE records the estimated regret

$$\hat{R}(\theta, A_v, A_*) = \frac{1}{n_{\text{trial}}} \sum_{i=1}^{n_{\text{trial}}} r(\theta, \bar{\tau}_{\theta,i}^v, \bar{\tau}_\theta^*) \quad (2.4)$$

by averaging the regret metric over $n_{\text{trial}} \geq 1$ trials. For brevity, the shorthand notation $\hat{R}(\theta)$ will generally be used, with the test agent A_v and optimal agent A_* implied.

The mathematical goal of BACRE is to solve for the scenario that maximizes regret via

$$\theta^* = \operatorname{argmax}_{\theta \in \Theta} \hat{R}(\theta), \quad (2.5)$$

where Θ represents the set of all possible scenarios. The maximization process is achieved through sequential mutations that can be traced back to a base scenario in the set Θ_0 . However, from a user’s perspective, the scenario of maximal regret is not necessarily the end goal. The act of maximizing regret is simply performed to find scenarios that represent the system’s weaknesses and limitations. Early stopping criteria may be used for when such scenarios are found (e.g., for when a failure scenario is found, which is defined as a scenario in which a minimum percentage of the vehicle’s attempts result in failures).

Figure 2.2 illustrates the main steps involved in BACRE applied to off-road AV navigation. The next subsections describe a novel generalized regret-metric for black-box navigation performance, the method for identifying “fit” scenarios for mutating, and the full BACRE algorithm.

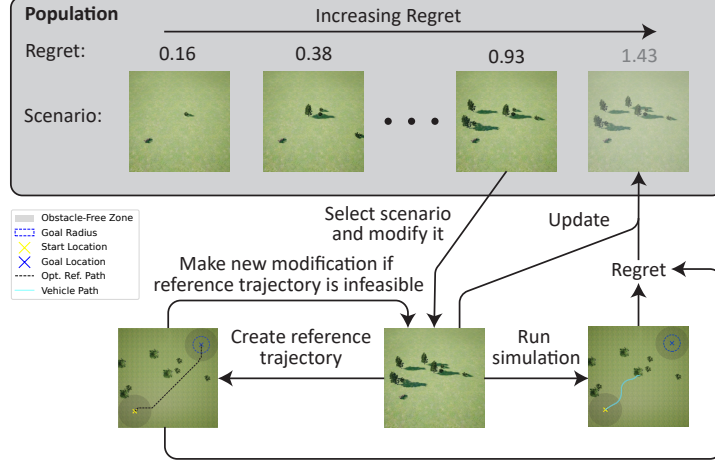


Figure 2.2: Overview of BACRE applied to off-road AVs. In each iteration, a candidate scenario is selected from the current population and one or more attributes are modified while ensuring the reference trajectory is feasible (a new set of modifications must be made to the original scenario until the reference trajectory is valid). Once the prescribed scenario is simulated, its regret is evaluated and the scenario-regret combination is added to the population.

2.4.1 Regret for General Navigation Tasks

This section presents a novel regret-metric for *general navigation* tasks in which the agent may not reach the goal. The errors in the final time, cost, and state between $\bar{\tau}_\theta^v$ and $\bar{\tau}_\theta^*$ in a scenario θ are of most value. For brevity, all final trajectory quantities are denoted with a subscript f (e.g., t_f , c_f , and x_f). Because the differences in final cost and time can be arbitrary in this context, final cost and time errors are denoted with the absolute value differences $|c_{\theta,f}^v - c_{\theta,f}^*|$ and $|t_{\theta,f}^v - t_{\theta,f}^*|$, respectively. At least one more term is needed to account for vehicle position. The term $\|\text{pos}(x_{\theta,f}^v - x_{\theta,f}^*)\|_2$ may seem reasonable and is zero if the agent reaches the goal. However, position error relative to where the optimal agent would be if it accumulated the same cost and/or time as $\bar{\tau}_\theta^v$ is more meaningful in the context of regret (an analogous rotational error, however, would not make sense). These errors are termed equivalent-cost and equivalent-time position error, and they require extracting the respective equivalent-cost and equivalent-time sub-trajectories of $\bar{\tau}_\theta^*$, denoted $\bar{\tau}_\theta^{*c}$ and $\bar{\tau}_\theta^{*t}$.

$\bar{\tau}_\theta^{*c}$ is the sub-trajectory of $\bar{\tau}_\theta^*$, with initial time $t_{\theta,0}^{*c} = t_{\theta,0}^* = 0$ and cost $c_{\theta,0}^{*c} = c_{\theta,0}^*$, such that its

final cost

$$c_{\theta,f}^{*c} = \begin{cases} c_{\theta,f}^v & \text{if } c_{\theta,f}^v < c_{\theta,f}^* \\ c_{\theta,f}^* & \text{if } c_{\theta,f}^v \geq c_{\theta,f}^*. \end{cases} \quad (2.6)$$

Note that $\bar{\tau}_{\theta}^{*c} = \bar{\tau}_{\theta}^*$ if $c_{\theta,f}^v \geq c_{\theta,f}^*$. Similarly, $\bar{\tau}_{\theta}^{*t}$ is the sub-trajectory of $\bar{\tau}_{\theta}^*$, with initial time $t_{\theta,0}^{*t} = t_{\theta,0}^* = 0$ and cost $c_{\theta,0}^{*t} = c_{\theta,0}^*$, such that its final time

$$t_{\theta,f}^{*t} = \begin{cases} t_{\theta,f}^v & \text{if } t_{\theta,f}^v < t_{\theta,f}^* \\ t_{\theta,f}^* & \text{if } t_{\theta,f}^v \geq t_{\theta,f}^*, \end{cases} \quad (2.7)$$

and $\bar{\tau}_{\theta}^{*t} = \bar{\tau}_{\theta}^*$ if $t_{\theta,f}^v \geq t_{\theta,f}^*$. Interpolation can be used to identify the states $x_{\theta,f}^{*c}$, and $x_{\theta,f}^{*t}$.

The proposed regret metric is given by

$$r(\theta, \bar{\tau}_{\theta}^v, \bar{\tau}_{\theta}^*) = \frac{|c_{\theta,f}^v - c_{\theta,f}^*|}{c_{\theta,f}^* - c_{\theta,0}^*} + \frac{\|\text{pos}(x_{\theta,f}^v - x_{\theta,f}^{*c})\|_2}{\|p_S - p_G\|_2} + \frac{|t_{\theta,f}^v - t_{\theta,f}^*|}{t_{\theta,f}^*} + \frac{\|\text{pos}(x_{\theta,f}^v - x_{\theta,f}^{*t})\|_2}{\|p_S - p_G\|_2}, \quad (2.8)$$

which consists of four normalized error terms (in the order written): cost error, equivalent-cost position error, time error, and equivalent-time position error. If time is implicitly included in f_c , then the time error and equivalent-time position error can be omitted. Although each term could be weighted differently, this paper uses an equal weighting of unity. While the position errors may appear to inflate the calculation, the cost/time errors can underestimate regret when they are near zero (depending on the definition of (2.2) and vehicle failure) *and* when the test agent is far from the goal. For a comparison, the regret in [66] is just $c_{\theta,f}^v - c_{\theta,f}^*$. While not considered, a term for final orientation could be added.

2.4.2 Identifying Fit Parents

BACRE selects “fit” scenarios for mutating by sampling from a curated set (instead of some other sampling distribution) to focus on scenarios whose mutations are more likely to quickly increase the maximum regret obtained. Let n_i denote the number of accumulated scenarios at the i th iteration, and let $\theta_{(j)}$ denote the scenario with the j th largest regret for $j \in 1, \dots, n_i$. The goal is to

create a set of candidate parents

$$\Theta_{\text{parents},i} = \{\theta_{(1)}, \dots, \theta_{(\bar{n}_i)}\} \quad (2.9)$$

at the i th iteration by curating the top $\bar{n}_i$ scenarios. Ideally, there should be some form of consistency in the quality of $\bar{n}_i$ for the algorithm to be most effective. A simple requirement is to ensure that the *regret curation ratio* (RCR), defined as

$$\alpha_i = \frac{\hat{R}(\theta_{(1)}) - \hat{R}(\theta_{(\bar{n}_i)})}{\hat{R}(\theta_{(1)}) - \hat{R}(\theta_{(n_i)})}, \quad (2.10)$$

is approximately constant across iterations. Specifically, given a desired RCR of $\alpha \in [0, 1]$, the cutoff $\bar{n}_i$ is the largest integer from 1 to n_i such that $R(\theta_{(\bar{n}_i)})$ lies in the set

$$\left[\hat{R}(\theta_{(1)}) - \alpha \left(\hat{R}(\theta_{(1)}) - \hat{R}(\theta_{(n_i)}) \right), \hat{R}(\theta_{(1)}) \right]. \quad (2.11)$$

α simply determines the upper fraction from the regret range to curate from. Preliminary experiments showed an effective range for α to be 0.05–0.1, and larger values resulted in slower convergence in BACRE.

2.4.3 The BACRE Algorithm

Algorithm 2.1 formally describes the steps in BACRE. The first step is to create a set of feasible base scenarios, which are the simplest possible, non-trivial scenarios the test agent may encounter (i.e., scenarios with at least one type of obstacle/challenge). Scenario feasibility (lines 4 and 17) requires existence of a solution to (2.2) and satisfaction of optional user-defined criteria. Infeasible scenarios are discarded and new ones are generated until all criteria are met as shown in Figure 2.2. In the main loop, scenarios are sampled from a curated set and then perturbed until N scenarios have been created. Mutations (line 17) can take on any form (e.g., adding, removing, and/or modifying attributes), although limiting their range/scope may help to prevent scenario complex-

Algorithm 2.1 BACRE (Sequential Version)

```
1: Input: Test agent, reference agent model for (2.2), RCR curation level  $\alpha$ , num. base scenarios  $N_0$ , num. maximum scenarios  $N$ , base scenario parameter space  $\Theta_0$ , num. trials to run  $n_{\text{trial}}$ , prob. of sampling random parent scenario  $p_{\text{rand}}$ , mutation method.
2: Initialize: Data buffer  $\Lambda$ 
3: for  $i = 1$  to  $N_0$  do ▷ Create base scenarios
4:   Create a feasible base scenario  $\theta_i \in \Theta_0$ 
5:   Compute reference trajectory  $\tau_{\theta}^*$  in  $\theta_i$  using Eq. (2.2)
6:   Run the test agent in scenario  $\theta_i$  for  $n_{\text{trial}}$  trials, compute the estimated regret  $\hat{R}(\theta_i)$  using Eq. (2.4), and store  $(\theta_i, \hat{R}(\theta_i))$  in  $\Lambda$ 
7: end for
8: for  $i = N_0 + 1$  to  $N$  do ▷ Main loop
9:   Sample  $u \sim \mathcal{U}[0, 1]$ 
10:  if  $u < p_{\text{rand}}$  then ▷ Select RCR level
11:     $\bar{\alpha} \leftarrow 1$  ▷ Use entire population
12:  else
13:     $\bar{\alpha} \leftarrow \alpha$  ▷ Use desired regret range
14:  end if
15:  Create  $\Theta_{\text{parents},i}$  with RCR level  $\bar{\alpha}$ 
16:   $\theta_p \leftarrow$  Uniform random sample from  $\Theta_{\text{parents},i}$ 
17:   $\theta_i \leftarrow$  Mutate  $\theta_p$  (ensure feasibility)
18:  Compute reference trajectory  $\tau_{\theta_i}^*$  in  $\theta_i$  using Eq. (2.2)
19:  Run the test agent in scenario  $\theta_i$  for  $n_{\text{trial}}$  trials, compute the estimated regret  $\hat{R}(\theta_i)$  using Eq. (2.4), and store  $(\theta_i, \hat{R}(\theta_i))$  in  $\Lambda$ 
20: end for
```

ity from increasing too quickly. Section 2.5.2 describes the mutations and feasibility criteria as considered in the experiments in Section 2.5. To add an element of randomness with probability $0 \leq p_{\text{rand}} \leq 1$ an RCR level of 1 is used (line 11 vs. line 13). Finally, throughout the entire search process, scenarios are labeled as “failures” or “successes”, with a failure scenario defined as one where at least 50% of the vehicle’s attempts result in failure (otherwise, the scenario is labeled as a success). Other thresholds could have been used, but in this dissertation we use such a high cutoff to ensure the scenario is indicative of a problem with the autonomy system.

Algorithm 2.1 presents the *sequential version* of BACRE because it assumes access to just a single test agent. If independent, identical agents can be simulated in parallel, then the *parallel version* of BACRE should be used in which the two for-loops become while-loops that loop over all available simulators and feed their data into a central data buffer.

Scenario feasibility, which was briefly discussed above, is an important topic that requires a more thorough discussion. We reiterate that the primary goals we have with BACRE are to find scenarios that: (1) highlight what the easiest thing is in our autonomy system that can “break”, (2) demonstrate flaws in the system design or inconsistencies in what the system is expected to be able to achieve, and (3) push the system to its performance limits. BACRE must be ran for numerous iterations to identify scenarios that address one or more of these goals because BACRE’s performance strongly depends on the size of the mutation that is applied at each search iteration *and* the criteria that constitutes scenario feasibility. We do not want to create scenarios that are knowingly impossible or near-impossible to navigate. However, to *truly* test if a scenario is feasible to navigate would require simulating every possible control sequence to determine if the vehicle can reach the goal safely. This is an intractable task and it is why we resort to letting the user specify various reasonable criteria for manual inspection of the scenarios to reduce the chance of creating an impossible scenario. Furthermore, if one is interested in finding scenarios at the vehicle’s performance limits, then some of the feasibility criteria may need to be relaxed; it is possible that the AV’s performance limit is also at the limit of what is physically possible.

2.5 Numerical Experiments and Results

The experiments³ we present in this section demonstrate the utility of BACRE paired with the proposed regret metric in (2.8) for finding (high-regret) adversarial, static scenarios for off-road AVs. The first experiment is an iterative design procedure in which BACRE is used to strategically identify adversarial scenarios to guide the improvements for an example autonomy system. The second experiment is a comparison between several variations of BACRE and a few baseline test methods. Comparing BACRE against the adversarial test methods discussed in Section 1.1 is quite challenging because these methods are primarily intended for controlling the dynamic components. In this section, the baseline comparisons consist of random search and a continuous genetic algorithm (CGA) [75]. Last, all experiments use the AutomaticSceneGeneration platform for demonstration

³Code and implementation details at https://github.com/tsender/bacre_2D

purposes.

2.5.1 Autonomy System

The experimental vehicle model is a virtual Polaris MRZR (with hand-tuned model parameters) with a forward-facing camera (RGB and depth) and a localization sensor (for position and orientation). A photo of a real MRZR and the virtual rendering used in UE4 are shown in Figure 2.3. The perception system is a custom traversability semantic segmentation DNN based on the U-Net architecture [76] that labels camera pixels as belonging to an object that is traversable, non-traversable, or to the sky. The path planner is an A* voting path planner that uses an obstacle grid to create a cost map and a vehicle grid for determining waypoints. If a non-traversable label from the perception system corresponds to a 3D point with a height of less than 2 meters, then the closest obstacle node is given a vote, increasing the cost within the vicinity. The path planner has a one second initial delay and then generates a new path every two seconds. The path follower tracks a look-ahead point 5 meters away on the most recently generated path and uses a PD controller to determine the steering angle while moving at constant speed of 5 m/s. Figure 2.4 provides an illustration of the components for this autonomy system.



Figure 2.3: A real (left) and virtual (right) Polaris MRZR vehicle. **Left:** An MRZR vehicle used by the Automotive Research Center at the University of Michigan. **Right:** A rendering of an MRZR as used in the UE4 simulations.

Table 2.1 shows the details in the U-Net architecture used for semantic segmentation. The labeled images for the training dataset were created using an automatic image labeler from the AutomaticSceneGeneration plugin. The dataset has a total of 4042 images with resolution $256 \times$

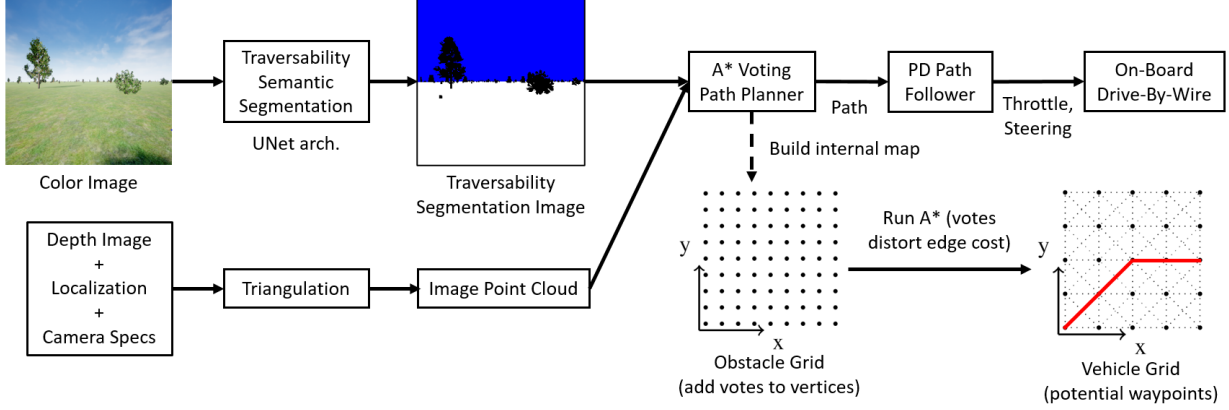


Figure 2.4: A* traversability autonomy stack used in the numerical experiments. The color image and semantic segmentation image shown were generated using the AutomaticSceneGeneration plugin.

256, and the images were collected with the sun placed at various inclination angles (0, 15, 30, 45, 60, 75, and 90 degrees). The dataset was split into 90% for training (3638 images) and 10% for validation (404 images), the learning rate was $\eta = 0.0001$, the batch size was 16, and the network was trained for 4 epochs (for a total of 912 training batches). The network was trained using categorical cross-entropy and achieved a validation loss of 0.0579 (average pixel loss) with a validation accuracy of 0.977 (average pixel accuracy).

The details for the A* voting path planner are described next. By applying A*, a graph search method, to a flat ground plane, the planner's output will be a path $p = (x_0, \dots, x_n)$, where $x_i \in \mathbb{R}^2, i = 0, \dots, n$.

The path planner discretizes the flat landscape of size $L_x \times L_y$ into an obstacle grid and vehicle grid. An optional border L_b can be added to extend the bounds of the grids (if the vehicle is likely to travel outside the main landscape, then adding a border can help the planner continue generating relevant paths to reach the goal). The obstacle grid G_o is defined as the l_o -discretization of the set $[-L_b, L_x + L_b] \times [-L_b, L_y + L_b] \subset \mathbb{R}^2$ for some node-spacing l_o . Every node location $q_i^o \in G_o$ stores the number of votes $v_i \in \mathbb{N}$ at that node, with a maximum of $v_{\max}$ votes per node. The vehicle grid G_v is defined as the l_v -discretization of the set $[-L_b, L_x + L_b] \times [-L_b, L_y + L_b] \subset \mathbb{R}^2$ for some node-spacing l_v . Every node location $q_i^v \in G_v$ is a possible node for A* path planning,

Table 2.1: DNN architecture for semantic segmentation network. All activation functions with Leaky RELU use a leak rate of $\alpha = 0.2$. All convolutional layers use an L2 kernel regularizer with penalty $\lambda = 0.001$, and all weights are initialized according to the He initializer method [1]. There are a total of 587,123 trainable weights in the network. In the “Name” column, the prefix “e” indicates an encoder block and the prefix “d” indicates a decoder block.

Name	Layer	Activation	Output Shape
input	Input Image	-	$256 \times 256 \times 3$
e1	Conv 3×3 , stride 2	Leaky RELU	$128 \times 128 \times 16$
e2	Conv 3×3 , stride 2	Leaky RELU	$64 \times 64 \times 32$
e3	Conv 3×3 , stride 2	Leaky RELU	$32 \times 32 \times 64$
e4	Conv 3×3 , stride 2	Leaky RELU	$16 \times 16 \times 128$
e5	Conv 3×3 , stride 2	Leaky RELU	$8 \times 8 \times 128$
-	Upsample	-	$16 \times 16 \times 128$
d1	Conv 3×3 , stride 1	Leaky RELU	$16 \times 16 \times 128$
-	Concatenate (d1, e4)	-	$16 \times 16 \times 256$
-	Upsample	-	$32 \times 32 \times 256$
d2	Conv 3×3 , stride 1	Leaky RELU	$32 \times 32 \times 64$
-	Concatenate (d2, e3)	-	$32 \times 32 \times 128$
-	Upsample	-	$64 \times 64 \times 128$
d3	Conv 3×3 , stride 1	Leaky RELU	$64 \times 64 \times 32$
-	Concatenate (d3, e2)	-	$64 \times 64 \times 64$
-	Upsample	-	$128 \times 128 \times 64$
d4	Conv 3×3 , stride 1	Leaky RELU	$128 \times 128 \times 16$
-	Concatenate (d4, e1)	-	$128 \times 128 \times 32$
-	Upsample	-	$256 \times 256 \times 32$
output	Conv 3×3 , stride 1	Softmax	$256 \times 256 \times 3$

with diagonal edges allowed between adjacent nodes. The subscript $i \in \mathbb{N}$ on q_i^o and q_i^v indicates the ID of the node in the corresponding grid. When a higher-level module detects an obstacle, it indicates to the planner the location $x_{\text{obs}} \in \mathbb{R}^2$ of the detected obstacle and the number of votes v_{obs} to add. Internally, the path planner determines the closest obstacle node $q_i^o \in G_o$ to x_{obs} , and then updates the corresponding number of obstacle votes to $v_i \leftarrow \max(v_i + v_{\text{obs}}, v_{\text{max}})$.

The cost of traversing any edge in the vehicle grid is equal to the arc-length of that edge times a scaling factor determined by the summation of multiple scaled (unnormalized) Gaussian distributions. Centered at each obstacle node is an unnormalized bivariate Gaussian PDF described with mean $\mu \in \mathbb{R}^2$, covariance $\Sigma \in \mathbf{S}_{++}^2$ (the set of 2×2 positive definite matrices), and a scaling

factor $\alpha \geq 0$ by the function

$$f(x; \mu, \Sigma, \alpha) = \alpha \exp \left((x - \mu)^T \Sigma^{-1} (x - \mu) \right) \quad (2.12)$$

where $x \in \mathbb{R}^2$. The cost for traveling across an arbitrary segment (x_1, x_2) where $x_1, x_2 \in \mathbb{R}^2$ is computed by the formula

$$\text{edgecost}_t(x_1, x_2) = \|x_1 - x_2\|_2 (1 + \gamma(x_1, x_2)), \quad (2.13)$$

where

$$\gamma(x_1, x_2) = \sum_{q_i^o \in G_o} \frac{1}{3} \left[f(x_1; q_i^o, \Sigma, \alpha_i) + f\left(\frac{x_1+x_2}{2}; q_i^o, \Sigma, \alpha_i\right) + f(x_2; q_i^o, \Sigma, \alpha_i) \right] \quad (2.14)$$

is a non-negative scalar that indicates the amount of distortion contributed by the surrounding Gaussian PDFs on the segment (x_1, x_2) . Each of the PDFs is centered at an obstacle node $q_i^o \in G_o$ and uses covariance Σ and the associated scaling factor $\alpha_i = \alpha_v v_i$, where $\alpha_v > 0$ is an additional parameter. Note that (2.14) simply averages the effect of all the PDFs over the two endpoints and the midpoint of edge (x_1, x_2) . A faster, but slightly less accurate, method for estimating $\gamma(x_1, x_2)$ that may be considered only uses the midpoint resulting in

$$\gamma(x_1, x_2) = \sum_{q_i^o \in G_o} f\left(\frac{x_1+x_2}{2}; q_i^o, \Sigma, \alpha_i\right). \quad (2.15)$$

While these two versions for computing the scale factor are not equivalent, observe that (2.14) and (2.15) will converge to the same value as $\|x_1 - x_2\|_2 \rightarrow 0$. Finally, the cost of traversing a path $p = (x_0, x_1, \dots, x_n)$ for $x_i \in \mathbb{R}^2, i = 0, \dots, n$ is given by

$$\text{cost}_t(p) = \sum_{i=0}^{n-1} \text{edgecost}_t(x_i, x_{i+1}). \quad (2.16)$$

Note that (2.16) is a generic cost formula, as the x_i 's in path p need not be constrained to lie on the

vehicle grid G_v , although this will be the case for the A* planner.

The heuristic used by the A* planner to estimate the cost-to-go from a vehicle grid node $q_i^v \in G_v$ to the goal p_G is the Euclidean distance $h(q_i^v) = \|q_i^v - p_G\|_2$. Furthermore, the starting vertex in each replanning step is the closest vehicle grid vertex to the vehicle's current location.

The above-described A* planner is the baseline version that is considered. Because graph-search planners are not well-suited for controlling a vehicle locally, it is expected that some subsequent paths may request infeasible changes in vehicle heading. To minimize this effect, the use of a directional cost is proposed in addition to (2.13) by comparing the vehicle's current heading to the individual path edge angles. Although this extra penalty could be applied everywhere, this is not necessary. Since the planner is assumed to replan frequently, this penalty need only be applied within a specified radius of the vehicle's current location so the beginning of the newly created path (which will be closest to the vehicle) is more dynamically feasible for the vehicle to follow.

Specifically, assume the edge in question is from the vertices $q_{i_1}^v$ to $q_{i_2}^v$, let $e_{12} = q_{i_2}^v - q_{i_1}^v$, let $e^v = (\cos(\psi^v), \sin(\psi^v))$ denote the vehicle's current heading vector with yaw angle $\psi^v \in [0, 2\pi]$, let $q_{i_v}^v \in G_v$ denote the closest vertex to the vehicle's location in the vehicle grid, and let $\theta_{v12} \geq 0$ denote the absolute value angle difference between the vehicle's heading and the vector e_{12} . The directional cost is given by

$$\text{edgecost}_d(q_{i_1}^v, q_{i_2}^v) = \begin{cases} 10^6, & \text{if } 0 \leq \|q_{i_v}^v - q_{i_2}^v\|_2 \leq r_1 \text{ and } \theta_{v12} > \theta_1 \\ 10^6, & \text{if } r_1 < \|q_{i_v}^v - q_{i_2}^v\|_2 \leq r_2 \text{ and } \theta_{v12} > \theta_2 \\ 0, & \text{otherwise,} \end{cases} \quad (2.17)$$

where $0 < r_1 \leq r_2$ are radius thresholds and $0 < \theta_1 \leq \theta_2$ are angle thresholds set by the user. *Two* radius and angular thresholds are chosen, instead of one, such that the affected region in the search space can more closely follow the kinematic turning radius of wheeled vehicles. The directional cost penalty effectively removes the affected edges from the search process. However, in case the vehicle happens to drive outside the bounds of the A* grids, removing edges could prevent A* from finding a valid path to bring the vehicle back on track, and adding a large value to the cost is a

fair compromise. The total cost of traversing a path $p = (x_0, x_1, \dots, x_n)$ for $x_i \in \mathbb{R}^2, i = 0, \dots, n$ is then given by

$$\text{cost}_{td}(p) = \sum_{i=0}^{n-1} \text{edgecost}_t(x_i, x_{i+1}) + \text{edgecost}_d(x_i, x_{i+1}). \quad (2.18)$$

The remaining item to describe is the path follower. The path follower acts as the controller within the autonomy stack and operates at 20 Hz. First, using the most recently generated A* path, a look-ahead point five meters away from the vehicle's current location is calculated. If such a point on the A* path does not exist, then the path follower will use the closest point on the path to the vehicle's location (this could be a path node or a point linearly interpolated between two path nodes). Let $x_k^* \in \mathbb{R}^2$ denote the look-ahead point and $x_k^v \in \mathbb{R}^2$ denote the vehicle's current location at the k th controller timestep for $k \in \mathbb{N}$. The desired yaw angle ψ_k^* is the angle pointing in the direction of $x_k^* - x_k^v$. The yaw angle error is $e_k = \psi_k^* - \psi_k^v$, where ψ_k^v is the vehicle's current heading, and the error is constrained to lie in the bounds $[-\pi, \pi]$ by remapping the angle appropriately (e.g., for $0 < \epsilon < 2\pi$, $\pi + \epsilon$ gets remapped to $-\pi + \epsilon$ and $-\pi - \epsilon$ gets remapped to $\pi - \epsilon$). Finally, the output steering angle is computed by the discretized PD control law

$$u_k^{\text{steer}} = K_p e_k + K_d \left(\frac{e_k - e_{k-1}}{\Delta t} \right), \quad (2.19)$$

where Δt is the period at which the controller runs, which in this case is 0.05 seconds. In the implementation, e_k has units of radians, u_k^{steer} has units of degrees (since UE4 uses degrees), and so K_p has units degree/radian and K_d has units (degree · second)/radian. The vehicle is set to travel at a constant forward speed of 5 m/s.

2.5.2 Experiment Parameters and BACRE Configuration

The following list contains the specific parameters for those described in Section 2.3:

- **Structural Attributes:** $L = 60$ meters; there can be either trees and/or bushes in the simulation with at most $N_{\text{tree}} = N_{\text{bush}} = 6$, $\psi_i^{\text{tree}} = \psi_i^{\text{bush}} = 0$ degrees and $s_i^{\text{tree}} = s_i^{\text{bush}} = 1$; and $r_{\text{obs-free}}^S =$



Figure 2.5: A rendering of a Barberry bush and Juniper tree as used in the UE4 simulations in this chapter.

$r_{\text{obs-free}}^S = 10$ meters. The type of tree and bush used in the simulations are a Juniper tree and Barberry bush. A rendering of these obstacles in UE4 is shown in Figure 2.5.

- **Textural Attributes:** The ground plane texture is grassy, and we fix $\psi^{\text{sun}} = 0$ degrees and $\theta^{\text{sun}} = 30$ degrees.
- **Operational Attributes:** $p_S = (10, 10)$ meters and $p_G = (50, 50)$ meters; $\phi_S = 45$ degrees; $r_{\text{goal}} = 5$ meters; $T_{\text{sim}} = 17$ seconds, $T_{\text{idle}} = 5$ seconds, $T_{\text{stuck}} = 5$ seconds, and $\theta_{\text{maxroll}} = \theta_{\text{maxpitch}} = 70$ degrees.

BACRE can control/mutate certain structural and textural attributes and will choose a category with equal probability:

- **Changing a Structural Attribute:** The possible actions are to add, remove, or modify an obstacle. Each action is randomly chosen according to the nominal probabilities p_{add} , p_{rem} , and p_{mod} , respectively (see Table 2.2 for the probabilities for all possible cases). All decisions required to perform an action are also conducted randomly. For example, when adding an obstacle, the obstacle type and all attribute values are chosen randomly.
- **Changing a Textural Attribute:** The possible actions are to modify ψ^{sun} or θ^{sun} with equal probability.

Table 2.2: Probability of each possible action when modifying a structural attribute of a parent scenario, assuming an obstacle class “class” has already been chosen.

Num. Obstacle Instances in $\theta, n_{\text{class}}(\theta)$	Prob. Add Obstacle	Prob. Remove Obstacle	Prob. Modify Obstacle
1	$\frac{p_{\text{add}}}{p_{\text{add}} + p_{\text{mod}}}$	0	$\frac{p_{\text{mod}}}{p_{\text{add}} + p_{\text{mod}}}$
$1 < n_{\text{class}}(\theta) < N_{\text{class}}$	p_{add}	p_{rem}	p_{mod}
N_{class}	0	$\frac{p_{\text{rem}}}{p_{\text{rem}} + p_{\text{mod}}}$	$\frac{p_{\text{mod}}}{p_{\text{rem}} + p_{\text{mod}}}$

Actions that instantiate/modify attribute values do so according to a uniform distribution over predefined ranges.

The reference agent for BACRE is used to assist in enforcing feasible scenarios and in the regret computation. Following the discussion at the end of Section 2.4, the criteria listed here is expected to significantly reduce the chance that an impossible scenario is created (assuming appropriate parameters are selected), though it may not entirely prevent it from happening.

- **Enforcing Feasible Scenarios:** Several criteria must be verified for a scenario θ to qualify as feasible, in addition to existence of a solution τ_{θ}^* to (2.2). If any criteria fails, then new action values are applied to the parent scenario until all criteria are met.

1. *Time Feasibility:* τ_{θ}^* must have final time $t_{\theta,f}^* \leq T_{\text{sim}}$.

2. *Minimum Obstacle Path Proximity:* For each $(t_{\theta,i}^*, x_{\theta,i}^*) \in \tau_{\theta}^*$, $\text{pos}(x_{\theta,i}^*)$ must be a minimum distance of $r_{\text{proximity}}$ from every obstacle in θ .

3. *Obstacle-Free Zones:* All obstacles must lie outside the obstacle-free radii placed around p_S and p_G .

- **Computing the Regret:** Evaluating $r(\theta, \bar{\tau}_{\theta}^v, \bar{\tau}_{\theta}^*)$ for a feasible scenario θ requires $\bar{\tau}_{\theta}^*$ and $\bar{\tau}_{\theta}^v$. We describe how to obtain τ_{θ}^v from the simulation trajectory $\tau_{\theta}^{\text{sim}}$ ($\bar{\tau}_{\theta}^v$ follows from the definition of a CAT).

1. *Vehicle Failed:* $\tau_{\theta}^v = \tau_{\theta}^{\text{sim}}$.

2. *Vehicle Succeeded*: $\text{pos}(x_{\theta,f}^{\text{sim}})$ is at a distance r_{goal} from p_G . Since $\text{pos}(x_{\theta,f}^*) = p_G$, an artificial trajectory $\tau_{\theta}^{\text{sim}2}$ from $\text{pos}(x_{\theta,f}^{\text{sim}})$ to p_G is appended to $\tau_{\theta}^{\text{sim}}$, creating $\tau_{\theta}^v = (\tau_{\theta}^{\text{sim}}, \tau_{\theta}^{\text{sim}2})$, such that $\text{pos}(x_{\theta,f}^v) = p_G$. In this chapter, $\tau_{\theta}^{\text{sim}2}$ is a straight-line trajectory with constant speed v and 100 time-state tuples. Although this $\tau_{\theta}^{\text{sim}2}$ is unlikely to flow seamlessly from $\tau_{\theta}^{\text{sim}}$, adding this short artificial trajectory should not considerably affect the calculation of $r(\theta, \bar{\tau}_{\theta}^v, \bar{\tau}_{\theta}^*)$ if $r_{\text{goal}} < r_{\text{obs-free}}^G$.

For these experiments, BACRE’s reference agent solves (2.2) in two steps. It first solves for the optimal path in $\mathbb{R}^2$ from the start to the goal using the A* voting path planner as described in Section 2.5.1, but with more precise and accurate knowledge of the environment than the test vehicle’s planner. The reference agent’s heading is assumed to be tangential to this optimal path, thus, the state’s dimension in τ_{θ}^* is $d = 3$. Then, the time component of τ_{θ}^* is determined by assuming the vehicle travels at a constant speed throughout the path. The reason for using A* instead of a more accurate trajectory planner like rapidly exploring random trees (RRT) or model predictive control (MPC) is because: (1) the xy path is thought to be more valuable than having a dynamically feasible trajectory for evaluating if the vehicle is roughly taking the path we expect it to (since there are infinitely many paths), and (2) the reference trajectory needs to be computed for *every* scenario, and thus, we want a (relatively) computationally fast method to solve for this trajectory.

The reference A* agent uses a grid size of $L_x = L_y = L = 60$ meters with obstacle grid spacing $l_o = 0.25$ meters and vehicle grid spacing $l_v = 0.5$ meters. The internal obstacle grid is exact to within the grid discretization, and obstacles are added to the closest obstacle node using the maximum number of allowed votes of $v_{\text{max}} = 50$. When computing the travel cost, the edge distortion scale factor is computed according to (2.14) with the parameters $\Sigma = 2.5I_{2 \times 2}$ and $\alpha_v = 1/v_{\text{max}}$. The planner also uses the directional constraints to prevent sharp turns in the initial generated path segments with the parameters $r_1 = 3$ meters, $r_2 = 6$ meters, $\theta_1 = 45$ degrees, and $\theta_2 = 72.5$ degrees. Thus, the planner uses the cost formula $\text{cost}_{td}(\cdot)$ from (2.18) to compute the optimal path (note, the heading is *not* considered in this calculation since A* is a global path

planner).

All experiments, unless stated otherwise, set $N_0 = 200$, $N = 5000$, $n_{\text{trial}} = 3$, and use the parallel version of BACRE with 12 workers. Base scenarios consist of a single obstacle. In the mutation step, BACRE applies a single mutation to the structural attributes with the probabilities $p_{\text{add}} = 0.3$, $p_{\text{rem}} = 0.3$, and $p_{\text{mod}} = 0.4$. The parent selection step uses $\alpha = 0.05$ and $p_{\text{rand}} = 0.1$. Finally, when evaluating the cost of the actual vehicle’s trajectory τ_{θ}^v , like with the A* reference agent, only the xy path component of τ_{θ}^v is used, but we instead use a different cost function. The reference agent uses (2.18) as the cost function f_c when solving (2.2), but when evaluating the cost of τ_{θ}^v , we instead use (2.16). This difference is just due to an implementation of the directional constraint (for the reference agent, we alternatively could have ignored those edges with actual constraints and then used (2.16) as the cost function). BACRE also uses $r_{\text{proximity}} = 4$ meters when checking for scenario feasibility and $v = 5$ m/s when creating $\tau_{\theta}^{\text{sim}2}$.

In all experiments (and within every scenario trial) the test vehicle does not have any prior knowledge about the environment - it assumes a flat environment without obstacles. The vehicle processes its sensor data for one second to create an initial map before it begins to move. The vehicle then continues to predict where obstacles are as it navigates the environment.

The entire simulation platform is split up across two Windows computers with an i9 CPU and 3090 GPU running UE4 and two Ubuntu computers for the ROS2-based autonomy system, with six workers per computer. On average, each experiment in this setup (of running BACRE for 5000 search iterations) took about 12 hours.

2.5.3 Iterative Design Procedure: Results and Discussion

The first experiment is an iterative design procedure in which BACRE is used to strategically identify adversarial scenarios to guide the improvements for a custom autonomy system. BACRE is ran for 5000 iterations, the problems in the maximum regret scenario are identified and fixed, and this process is repeated for a total of three iterations. This experiment intentionally begins with a poorly-tuned autonomy system, denoted AV 1. Table 2.3 shows a summary of the results from

Table 2.3: Results from the iterative design experiment. Shown are the Base AV and its problems, the Test AV and its improvements, the max regret scenario for the Base AV, and the associated regret learning curves. The max regret scenario shows a front-aerial view of the scene (upper-left), a third-person view from behind the starting position (upper-right), and an orthographic-aerial view (lower) overlaid with the paths from both the Base AV and the improved Test AV. In the regret learning curves, the blue curves for regret have been smoothed using a moving average window of size 100.

	Iteration 1	Iteration 2	Iteration 3
Base AV and Discovered Problems	<p>AV 1</p> <ol style="list-style-type: none"> 1) A* paths often request initial infeasible heading change (e.g., > 90 deg.). 2) Long computational burden by A*, resulting in a path far away from vehicle's current location. 	<p>AV 2</p> <ol style="list-style-type: none"> 1) Inefficient, nonvectorized pixel-by-pixel camera processing. 2) Steering controller slow to respond. 	<p>AV 3</p> <ol style="list-style-type: none"> 1) A* paths can alternate on how to avoid upcoming obstacles (left vs. right), occasionally resulting in collision.
Test AV with Improvements to Base AV	<p>AV 2</p> <ol style="list-style-type: none"> 1) Incorporate directional cost penalty on A* paths. 2) Use faster, but less accurate, A* edge cost calculation, and reduce assumed obstacle size. 	<p>AV 3</p> <ol style="list-style-type: none"> 1) Vectorize camera processing code, and separate mapping and planning operations. 2) Double steering controller P gain. 	<p>AV 4</p> <ol style="list-style-type: none"> 1) Only update A* path beginning from a look-ahead point 5 meters away from vehicle's current location.
Max Regret Scenario for Base AV (Dimensions in meters)			
Regret Learning Curve			

this experiment which will be referred to when discussing each iteration.

In AV 1, the A* planner used a grid size of $L_x = L_y = 60$ meters, with a border of size $L_b = 50$ meters, and the grids use a spacing of $l_o = 1$ meter for the obstacle grid and $l_v = 2$ meters in the vehicle grid. The planner uses (2.16) to compute the path cost and uses (2.14) for computing the edge distortion factors with $\Sigma = 2.5I_{2 \times 2}$, $\alpha_v = 1/v_{\max}$, and $v_{\max} = 50$. The path follower uses the control gains of $K_p = 10$ degree/radian and $K_d = 5(\text{degree} \cdot \text{second})/\text{radian}$ and uses a

look-ahead distance of 5 meters. This autonomy stack is implemented using two ROS nodes with the AutomaticSceneGeneration plugin and ROS2 interface. One node is dedicated to perception and navigation in which the perception system is merged with the A* planner. The second node is dedicated to control and consists of the path follower.

The parameters chosen for AV 1 are known to result in poor performance of the vehicle, such as the controller being slow to respond and the A* planner possibly requiring too long to compute an optimal path. The goal is to start with an “easy” system to break and demonstrate that BACRE can continue to find adversarial scenarios after the system is improved. BACRE was easily able to identify an adversarial scenario for AV 1 within the 5000 search iterations. The column labeled “Iteration 1” in Table 2.3 shows this maximum regret scenario for AV 1 (the base AV for this iteration) along with BACRE’s regret learning curves (the table also summarizes the problems with AV 1 and the improvements that are made, but these will be discussed shortly in greater detail). For this particular maximum regret scenario, the light blue curves denote the paths taken by AV 1. We observe that the vehicle took a circuitous route to reach the goal and failed to reach the goal because the simulation timeout expired.

One of BACRE’s primary limitations is that because it is a black-box algorithm, it cannot explain the vehicle’s behavior that results in a high-regret or failure scenario. Thus, understanding what leads to the vehicle’s poor performance requires manual inspection/analysis. Figure 2.6 shows three separate snapshots of AV 1 path planner’s internal state for its maximum regret scenario. Two observations can be drawn from this figure: (1) the newly created paths are impossible for a vehicle to follow reasonably well since the initial portion of the trajectories makes at least a 90-degree angle relative to the vehicle’s current heading, and (2) due to the large computational burden of at least 1.5 seconds for the A* planner to generate a path, the vehicle is no longer near the starting point of the path, and attempting to follow such a path forces the vehicle on a long and windy journey. These problems point towards a poorly-tuned autonomy stack (which is expected).

Multiple improvements are made to address these shortcomings. To prevent infeasible paths from being generated, the A* planner should use the cost function (2.18) to generate the optimal

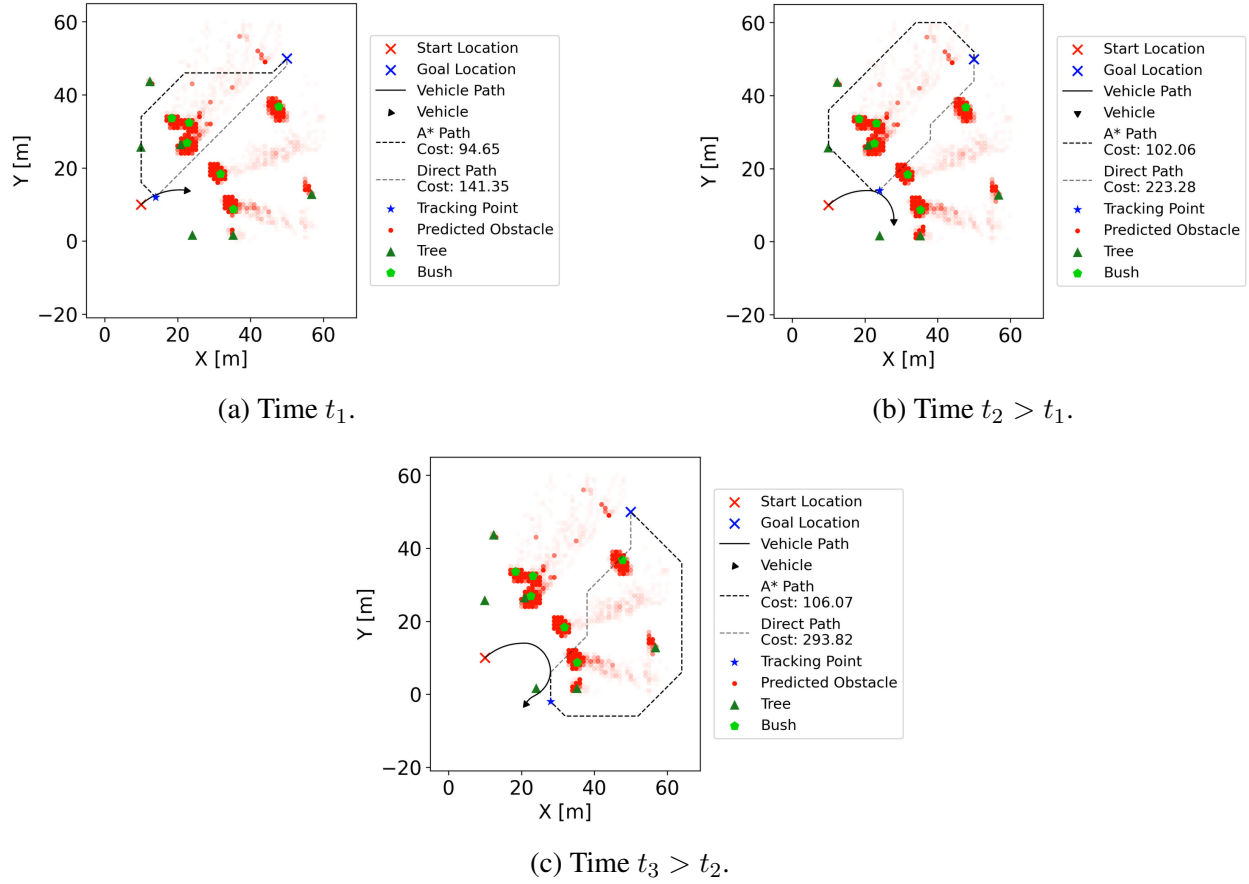
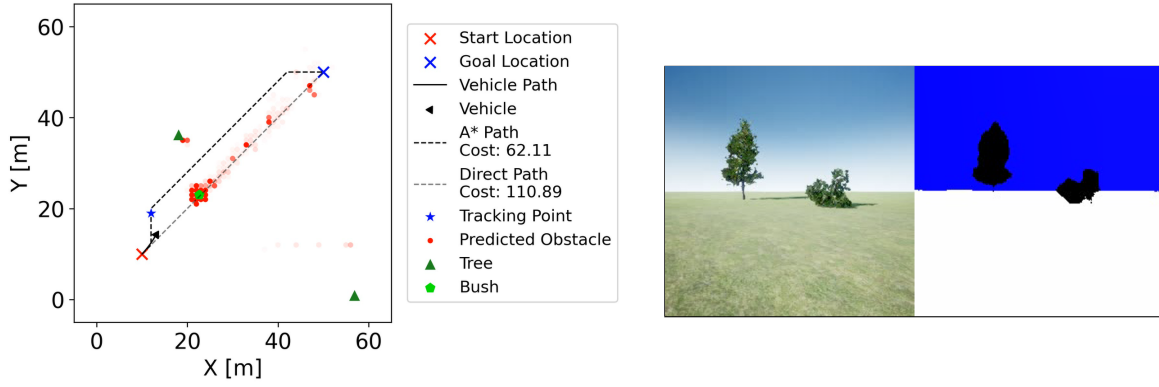


Figure 2.6: Three separate snapshots of AV 1 path planner’s state for the maximum regret scenario in iteration 1 with BACRE. The vehicle and all obstacles are indicated by point markers (their actual size is not shown). The red dots coincide with the A* planner’s obstacle grid, and the intensity of the red indicates how many votes (as a fraction of $v_{\max}$) are assigned at that vertex. At each of these time instances, the most recently generated path that the vehicle should begin following is shown. A direct path and its associated cost is also shown for validation that the A* path is reasonable.

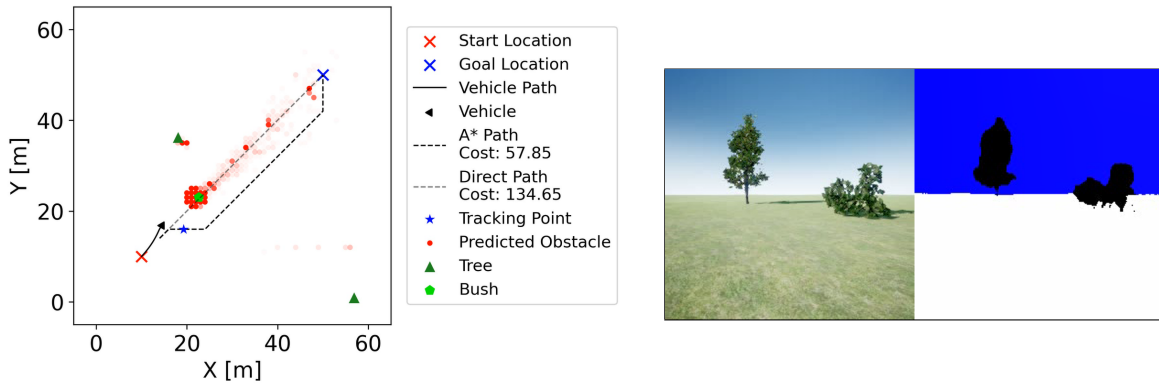
path as it incorporates the directional penalty, and its associated parameters are set to $r_1 = 3$ meters, $r_2 = 6$ meters, $\theta_1 = 45$ degrees, and $\theta_2 = 72.5$ degrees. Reducing the computational cost in A* can be accomplished by using the alternate edge distortion formula (2.15) and by assuming a smaller obstacle “width” by setting $\Sigma = 1.5I_{2 \times 2}$. This improved vehicle is denoted AV 2, and its performance is validated on the maximum regret scenario from iteration 1 in Table 2.3. The paths for AV 2 are shown in red (AV 2 is referred to as the test vehicle at this point) and demonstrate the vehicle can reach the goal without problems.

BACRE can now be pitted against AV 2. After running BACRE for another 5000 iterations, BACRE was yet again able to identify an adversarial scenario for vehicle. The column labeled “Iteration 2” in Table 2.3 shows this maximum regret scenario for AV 2 (now the base AV for iteration 2) along with BACRE’s regret learning curves. In this maximum regret scenario, the vehicle collides into a bush far from the goal radius, and since the vehicle can only move forward, either the stuck timeout or simulation timeout eventually expires ending the run. Some of the vehicle’s internal decisions that lead up to this collision are shown in Figure 2.7, which show chronological snapshots of the path planner and perception systems’ states. Moving from Figure 2.7a to Figure 2.7b indicates a change in the general direction the A* planner wants the vehicle to avoid the upcoming bush (initially left and then right). While addressing this A* switching behavior might be needed in future iterations, it may not be required just yet. Knowledge of the vehicle’s turning radius suggests that the vehicle could have avoided the obstacle if it made a sharper turn than what was made as it approached the state shown in Figure 2.7c. The path follower can easily be adjusted to follow a path more quickly, and the chosen improvement is to increase the gain K_p from 10 to 20 degree/radian.

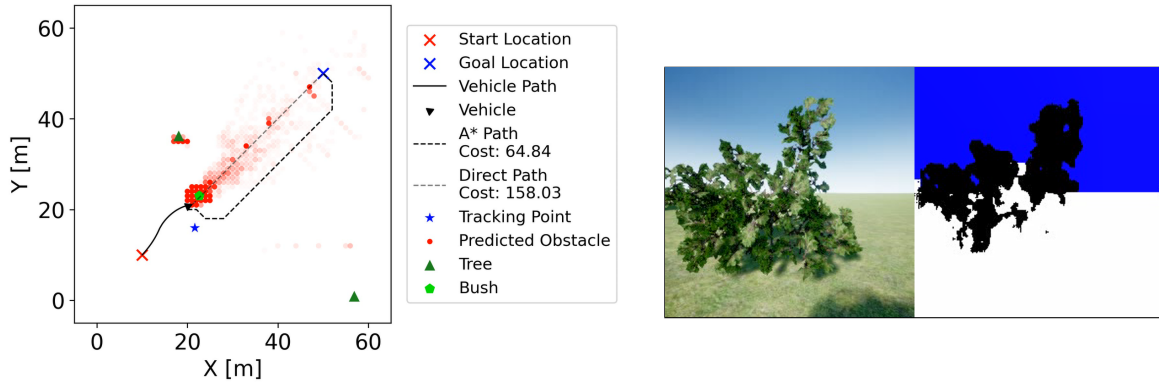
A second and more concerning problem that cannot be easily shown through static figures, is that a video replay of the perception system’s states indicate a bottleneck in the perception pipeline. It was not a trivial effort to identify the source of this problem. The predicted semantic segmentation image in Figure 2.7c shows that a considerable fraction of the image is classified as a non-traversable obstacle. Since this region is almost entirely under the two meter height threshold, each of these pixels corresponds to a vote that will be added to the A* planner’s internal map. The original implementation (from AV 1) uses a basic for-loop approach in Python to add the votes to the A* map pixel-by-pixel, which turns out to be very slow (in some cases, processing a single frame can take at least 1-3 seconds). Fortunately, this operation can be vectorized and made much faster (by a factor of about 1000 on average). Furthermore, to remove any potential future conflict with the A* mapping and planning operations, which were originally implemented as a single ROS node in AV 1, these two operations are given *separate* dedicated ROS nodes.



(a) Time t_1 .



(b) Time $t_2 > t_1$.



(c) Time $t_3 > t_2$.

Figure 2.7: Three separate snapshots of the internal state for the path planner and perception system in AV 2 for the the maximum regret scenario in iteration 2 with BACRE. Time increases from top to bottom. The images on the left show the path planner state. The images on the right show the raw camera image and the predicted semantic segmentation image with the labels traversable (white), non-traversable (black), and sky (blue).

Let AV 3 denote the improved vehicle from AV 2. AV 3 must first be validated against AV 2's maximum regret scenario. As desired, the red paths in Table 2.3 for the iteration 2 test AV (which is AV 3) indicate the vehicle can successfully avoid colliding with the bush and reach the goal. BACRE can now be pitted against AV 3 for the third and final design iteration in this experiment. The column labeled "Iteration 3" in Table 2.3 shows this maximum regret scenario for AV 3 (now the base AV for iteration 3) along with BACRE's regret learning curves. Figure 2.8 shows three chronological snapshots of the path planner's state, and the same A* switching that occurred in iteration 2 occurred again. Figure 2.8a to Figure 2.8b shows the path planner chose to switch the general direction of travel, and as the vehicle attempts to follow the new path, it collides with the tree as shown in Figure 2.8c. Although improving the controller's response rate worked in iteration 2, the fact that BACRE identified another scenario highlighting this same problem indicates that A*'s switching behavior is the next easiest flaw to take advantage of and should be addressed.

Recall that the version of A*, as described in Section 2.5.1, creates an entirely new path every two seconds. While the vehicle's perception of the *overall* environment will change over time, it is unlikely that its perception of its *local* environment (especially the region directly in front of the vehicle) is unlikely to change significantly before the next planning step. Accordingly, a reasonable approach to reduce the occurrence of A*'s switching behavior is to let the vehicle continue following the initial segments of the previously generated path (up to a certain point) and then have A* create a new path beginning from this old segment. The hypothesis is that this can reduce the extent to which the vehicle's local behavior will need to change significantly.

To make this idea mathematically concrete, let k denote the current replanning step, $x_k^v \in \mathbb{R}^2$ denote the vehicle's current location, $\psi_k^v \in \mathbb{R}$ denote the vehicle's current yaw angle, and $p_{k-1} = (q_{k-1,i_1}^v, \dots, q_{k-1,i_n}^v)$ denote the A* path from the previous replanning step with $q_{k-1,i}^v \in G_v$. Let n^v denote the element of the closest point on the path p_{k-1} to x_k^v . Let $x_k^{v*} \in \mathbb{R}^2$ be a look-ahead point on path p_{k-1} that is a distance $d_{\text{keep}} > 0$ from x_k^v , and let n^{v*} denote the element of the closest point on the path p_{k-1} to x_k^{v*} . The A* path $p_k = (p_{k1}, p_{k2})$ is then comprised of two subpaths, p_{k1} and p_{k2} , with subpath $p_{k1} = (q_{k-1,i_{n^v}}^v, \dots, q_{k-1,i_{n^{v*}-1}}^v)$ (subpath p_{k1} stops at index $n^{v*} - 1$ because

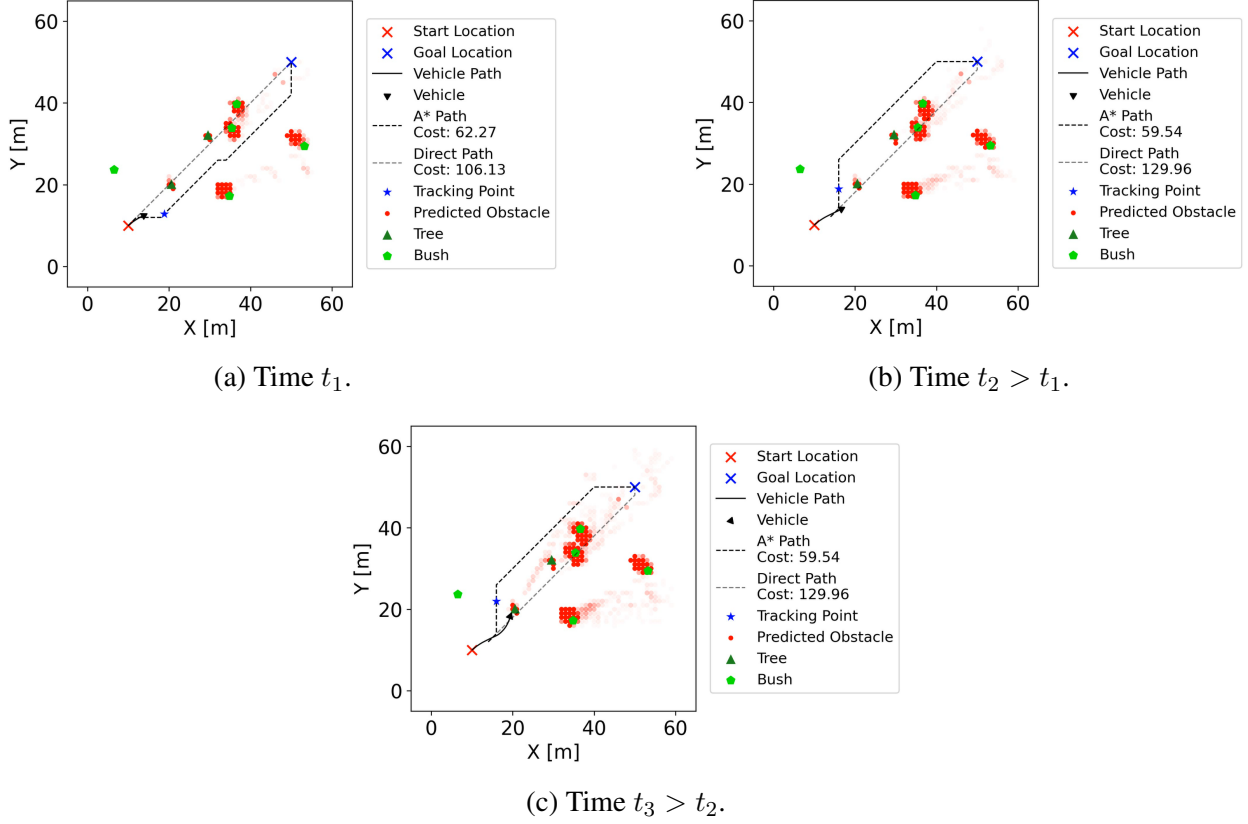


Figure 2.8: Three separate snapshots of AV 3 path planner’s state for the maximum regret scenario in iteration 3 with BACRE.

subpath p_{k2} starts at the point $q_{k-1,i_{nv*}}^v$). Subpath p_{k2} is determined from running the A* planner with the same parameters set in AV 3, with the exception that evaluating (2.17) uses the parameters $q_{i_v}^v = q_{k-1,i_{nv*}}^v$ and $e^v = q_{k-1,i_{nv*+1}}^v - q_{k-1,i_{nv*}}^v$. AV 4 denotes this improved vehicle model with the parameter $d_{\text{keep}} = 5$ meters. AV 4 is then validated against AV 3’s maximum regret scenario, and as desired, the red paths in Table 2.3 for the iteration 3 test AV (which is AV 4) indicate the vehicle can successfully avoid colliding with the bush and reach the goal.

The last thing to mention from this experiment is that even though BACRE cannot find *specific* weaknesses, any high-regret scenario’s family tree, generated from the successive mutations, can *show* which perturbations degrade the vehicle’s performance. Figure 2.9 shows a subset of the family tree from the maximum regret scenario in iteration 1. BACRE found an adversarial scenario in 2381 iterations (the level 12 scenario in Figure 2.9), but creating the maximum regret scenario

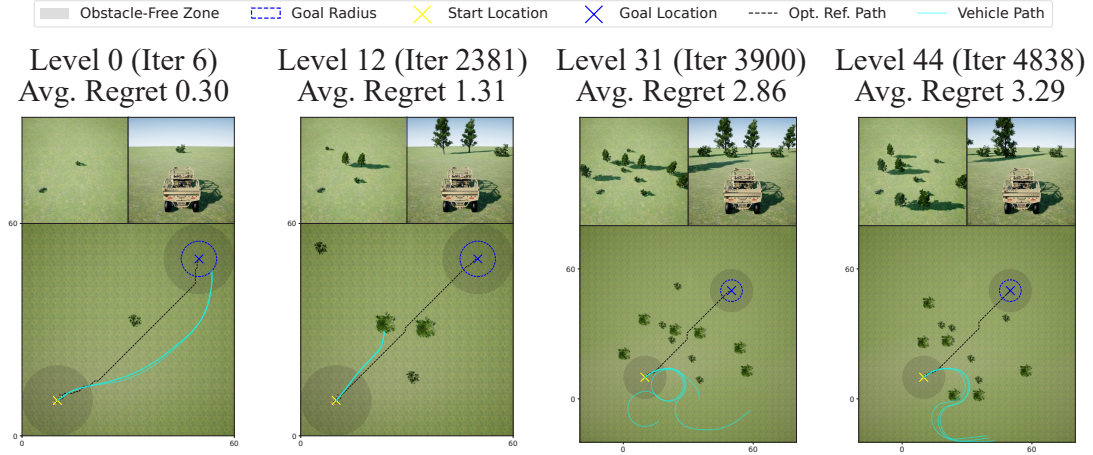


Figure 2.9: Partial sub-tree for the max regret scenario in iteration 1 in Table 2.3. “Level” refers to the tree level and “Iter” refers to the overall iteration that generated the scenario. Dimensions are in meters.

took an additional 2400+ iterations.

While only a few design iterations are performed for demonstration purposes, the results support the hypothesis that BACRE can identify adversarial scenarios even after making successive autonomy improvements.

2.5.4 BACRE Comparison: Results and Discussion

In the comparison experiment, all methods are ran for five trials and use AV 4 (from the prior experiment) as the test vehicle. Figure 2.10 summarizes the regret data, Figure 2.11 shows the maximum regret scenarios, and Table 2.4 summarizes the fraction of scenarios whose regret exceeds 0.75 (a qualitative analysis suggests this was a good threshold for indicating an adversarial scenario), and the fraction of scenarios where the vehicle failed to reach the goal. As expected, random search (in which the obstacle count, type, and locations are chosen randomly) does not exhibit any learning, as shown in Figure 2.10a (left), and it achieves the lowest rates in Table 2.4. However, Figure 2.10a (right) shows that, on average, random search was able to generate higher regret scenarios than the vanilla BACRE. BACRE’s sub-par performance is likely due to getting stuck in local optima and insufficient exploration. To mitigate these problems, a variation of BACRE is tested by linearly an-

nealing α from 0.5 to 0.05 and p_{rand} from 0.5 to 0.1 over the first 1000 iterations after generating the base scenarios. Figure 2.10a shows that annealing enables a steadier learning curve with a lower standard error than its original version. BACRE with annealing also briefly surpasses the average maximum regret than that of random search. A version of BACRE that applies three mutations (instead of one) is also tested following the ACCEL paper. Figure 2.10a (right) shows that applying more mutations enables BACRE to more quickly maximize regret than the other variations, but the left plot shows no improvement to the average regret. Among these variations, BACRE with annealing appears to be better at finding high regret scenarios and failure cases, as supported by Figure 2.10a (left) and Table 2.4.

The CGA tested is a variation of the method described in [75] that uses interpolation at the crossover point, but each element in the chromosome (in this case, the x and y coordinates for each obstacle) can be mutated to a new random value with probability 20%. The population size is 200, new populations are created from the top 10 pairs of samples, and a total of 25 populations are created (so that up to 5000 scenarios can be created). Since a CGA requires a fixed chromosome dimension, two settings are tested, using 6 and 12 obstacles (split evenly between trees and bushes). Figure 2.10b shows the average and maximum population regret. As expected, both studies show fairly quick convergence of the maximum regret, with convergence occurring sooner with a larger obstacle density. The CGA with 12 obstacles also attains the highest regret over all methods tested.

The last BACRE variation, labeled as “BACRE w/Annealing + NEET”, uses annealing and a modified regret metric without the equivalent error terms (EETs) proposed in (2.8) (this is the closest regret formulation to the one in [66]). Interestingly, the average regret is quite similar to the versions of BACRE without annealing. And unexpectedly, this version attains the highest rates in Table 2.4. Nevertheless, as stated at the end of Section 2.4.1, a quantity like $|c_{\theta,f}^v - c_{\theta,f}^*|$ alone can underestimate the regret making it inferior to (2.8) for off-road general navigation tasks.

Two final items are worth discussing. First, despite BACRE’s apparent success, the use of setting attributes uniformly at random is generally inefficient and likely inflated the values in Table 2.4 (such modifications to high-regret scenarios are unlikely to affect vehicle performance). And

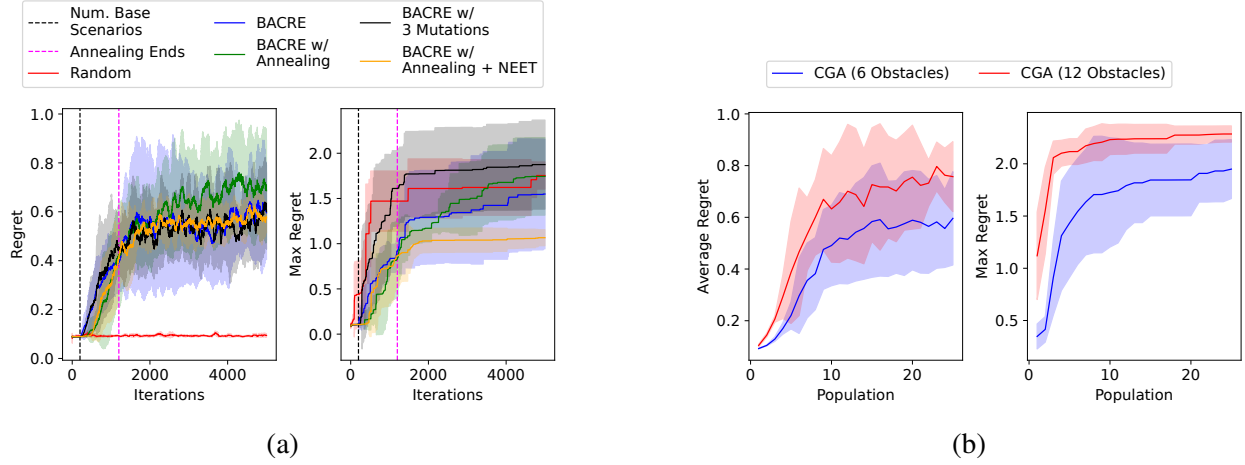


Figure 2.10: Regret learning curves from the comparison experiment. Shown are the mean and standard error over five trials. Smoothing is used in the left plot of (a) with a moving average window of size 100.

Table 2.4: Additional metrics from the comparison experiment.

Method	Fraction of Scenarios with $R(\theta) \geq 0.75$	Vehicle Failure Rate
Random Search	0.0013	0.0015
BACRE	0.223	0.184
BACRE w/Annealing	0.282	0.247
BACRE w/3 Mutations	0.211	0.204
BACRE w/Annealing + NEET	0.333	0.305
CGA (6 Obstacles)	0.147	0.143
CGA (12 Obstacles)	0.196	0.182

second, despite using a low RCR of 5%, there is a considerable gap between the average and maximum regret throughout most learning curves. Perhaps enough random mutations resulted in easier scenarios and/or finding adversarial scenarios truly is that difficult in open-field environments.

2.6 Conclusions

This chapter introduced the foundational tools needed to provide targeted adversarial testing for an off-road AV navigating in a static, unstructured environment. We first introduced some key terminology and a custom simulation platform that highlights all of the necessary features re-

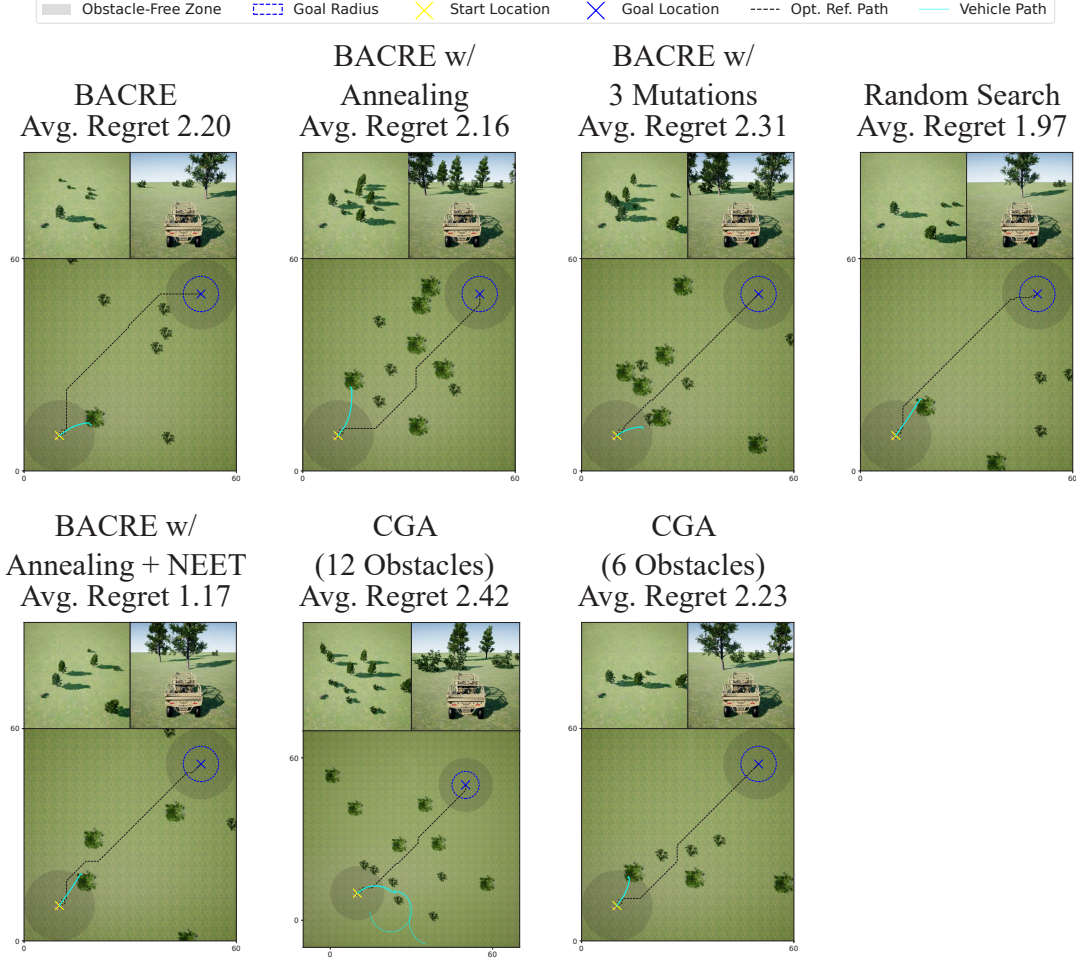


Figure 2.11: Maximum regret scenarios from the comparison experiment.

quired to perform automated off-road AV navigation simulations. We then proposed a new method called BACRE as a regret-informed evolutionary algorithm for generating adversarial scenarios for black-box autonomy systems, along with a novel regret-based metric was proposed for evaluating general navigation performance. BACRE was applied to a mildly complex custom off-road AV and experiments demonstrated that: (1) BACRE’s curating and evolutionary features allow it to design increasingly difficult scenarios for the test AVs, and (2) BACRE can accelerate the vehicle development process by identifying weaknesses even after improving the test system. BACRE also showed that adversarial scenarios on a flat, open field can be constructed by just a handful of strategically placed obstacles in configurations that are non-intuitive for a human to conceive.

Despite the perceived success of BACRE from the experiments that were presented, there are

several directions for future work that would increase BACRE’s potential impact for the off-road community. Future work should focus on extending the algorithm and experiments to modify the terrain (elevation, textures, etc.) and weather conditions, as well as pitting BACRE against far more sophisticated off-road autonomy systems than the example system presented. Doing so may also require improving BACRE’s search process and the regret metric. Further, a secondary goal is to extend BACRE for handling environments with dynamic components.

The next chapter focuses on the next most important upgrade: being able to generate 3-dimensional (3D) adversarial terrains for sophisticated off-road trajectory planners/controllers.

CHAPTER 3

An Evolutionary Approach for Generating 3-Dimensional Adversarial Scenarios for Black-Box Off-Road Autonomy Systems

3.1 Introduction

Although the results in Chapter 2 show potential for BACRE to find weaknesses in off-road AVs, there are several concerns and questions that must be addressed if BACRE were tasked with finding failure scenarios that better represent realistic off-road environments:

1. Terrain that is uneven, rugged, lacking in defined paths, and is difficult to navigate are hallmark features of off-road environments. However, Chapter 2 does not provide any guidance on how to mutate 3D terrain to create such features. The experiments and results only focus on rearranging obstacles on a flat, open field. A methodology for creating 3D mutations is required.
2. It is unclear if the proposed regret metric is sufficient for evaluating an off-road AV's performance in realistic off-road scenarios. The reference agent in BACRE is a global planner, namely A*, which can only account for vehicle position along waypoints. If the actual vehicle undergoes jerky linear or rotational motions, the regret metric with an A* reference agent will not evaluate such unsafe/undesirable motions with higher regret. Even if a more complete and realistic reference trajectory were computed (which could be cost-prohibitive), the regret metric still lacks clarity on how to account for such jerky motions.

3. BACRE does not have a methodical/strategic mutation strategy. BACRE uses a naive uniform random sample to change the value of a scenario’s attribute. Allowing BACRE to modify the environment’s structural geometry at a uniformly random location from within the entire landscape can result in choosing a location that is far from the ideal location and/or is too far to alter the vehicle’s behavior. Fig. 3.1 provides an explicit example of this phenomena. This behavior occurs quite frequently with BACRE and considerably increases the number of iterations needed to find a high-regret scenario. If either the landscape size and/or the number of structural mutations BACRE can control increased, then this problem would be exacerbated.
4. Finally, BACRE lacks a mechanism to avoid getting stuck in local maxima. BACRE creates and maintains a single, ever-growing population. If BACRE were to get stuck (i.e., unable to increase regret any further and find a failure scenario), then the population would saturate with scenarios too closely related to the local maxima (due to curation), thus making it much less likely to create a higher regret scenario, which would need to be different from the local maxima.

The first two limitations are implementation concerns, and the latter two are algorithmic concerns. Section 3.2 addresses the algorithmic limitations, and then Section 3.3 addresses the implementation concerns. Section 3.4 then discusses numerical experiments that show these improvements are effective and that BACRE-3D is the most effective tool in the BACRE family. Finally, Section 3.5 concludes the chapter.

3.2 BACRE-3D

This section describes the BACRE-3D algorithm, an extension of BACRE tailored for identifying weaknesses in systems that operate in 3D environments, such as off-road AVs. The full algorithm is outlined in Algorithm 3.1, with the inputs listed in Algorithm 3.2 due to space, and the core functions from the original BACRE method are outlined in Algorithm 3.3. The following subsections describe the two main algorithmic improvements per the limitations raised in Section 3.1:

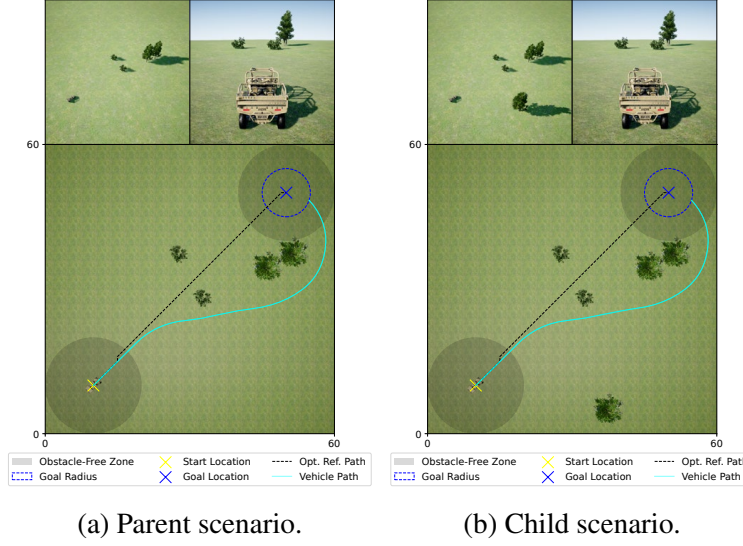


Figure 3.1: A parent-child scenario combination from one of the experiments in Section 2.5. The child scenario has an added tree roughly 30 meters to the right of the start location which does not alter the vehicle’s behavior. Each figure shows a front-aerial view of the scene (upper-left), a third-person view from behind the starting position (upper-right), and an orthographic-aerial view (lower) overlaid with additional information.

improving the search process (Section 3.2.1, addresses limitation 3) and avoiding local maxima (Section 3.2.2, addresses limitation 4). Fig. 3.2 provides an illustration comparing how the different BACRE variations construct the population. This figure and these algorithms are discussed in more detail in the following subsections.

3.2.1 Improving Search Efficiency

Mutations are essential in BACRE as they are the primary gateway for creating scenarios of higher regret. This mechanism should be as optimal as possible because it contributes significantly to the algorithm’s overall efficiency. This section addresses BACRE’s inefficient mutation approach.

From a general perspective, one may first wonder what the *optimal* mutation is. Let $a : \Theta \times S_a \rightarrow \Theta$ be an action function that operates on the scenario space Θ and the action’s (bounded) parameter space S_a (which could be a Cartesian product of multiple sets) to produce a mutated scenario. For example, if the action is to add an obstacle, then S_a would describe parameters such as the object’s location $(x, y) \in \mathbb{R}^2$ and perhaps other attributes like orientation and size. A

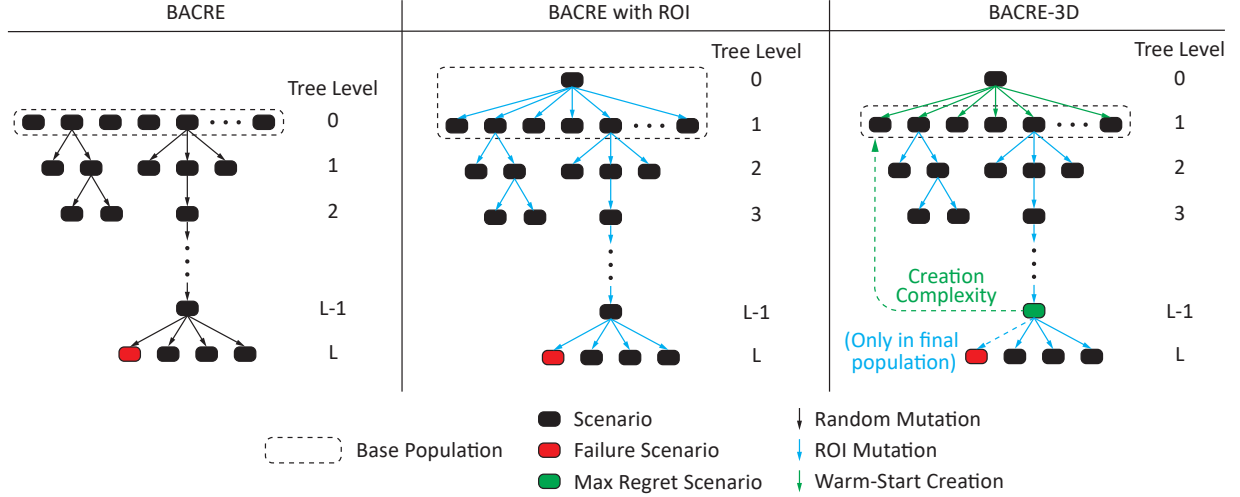


Figure 3.2: A comparison of how the BACRE variations construct the scenario population, represented as a tree, using the optimizations presented in this chapter. The maximum regret scenario is not shown in the left or middle diagrams because the goal is to find a failure scenario. In the BACRE-3D diagram, the failure scenario created by the dashed blue line is only present in the final population. Furthermore, the green scenario shown may or may not correspond to the maximum regret scenario in that final population.

mutation is then described by the tuple (a, s_a) to denote an action a and an associated parameter vector $s_a \in S_a$. The optimal mutation for a scenario θ is then the best action-parameter pair over all possible actions and their associated parameter spaces. Assuming n action functions, the optimal parameter vector for action $a_i \in \{a_1, \dots, a_n\}$ in scenario θ is

$$\begin{aligned}
 s_{a_i}^* &= \underset{s \in S_{a_i}}{\operatorname{argmax}} \left[\hat{R}(a_i(\theta, s)) - \hat{R}(\theta) \right] \\
 &\text{subject to } \left[\hat{R}(a_i(\theta, s)) - \hat{R}(\theta) \right] \geq 0,
 \end{aligned} \tag{3.1}$$

and the optimal mutation is the pair

$$(a^*, s_a^*) = \underset{(a, s_a) \in \{(a_i, s_{a_i}^*)\}_{i=1}^n}{\operatorname{argmax}} \left[\hat{R}(a(\theta, s_a)) - \hat{R}(\theta) \right]. \tag{3.2}$$

The subscript θ is omitted from a^* and s^* to improve readability but its dependence is implied.

Solving (3.1) is generally intractable due to searching over S_{a_i} . However, the structure of (3.1)

Algorithm 3.1 BACRE-3D (Sequential Version)

Input: Listed in Algorithm 3.2 due to space.

- 1: $m_{\text{base}} \leftarrow 1$ ▷ Initial creation complexity
- 2: $\mathcal{P} \leftarrow \text{List}$ ▷ Population data buffer
- 3: $\mathcal{D} \leftarrow \text{Dictionary}$ ▷ Distance dictionary
- 4: $\mathcal{E} \leftarrow \text{List}$ ▷ List of ROI estimators
- 5: **for** $i = 1$ **to** n **do**
- 6: $E_{a_i} \leftarrow \text{CreateROIEstimator}(r_i^0)$
- 7: Append E_{a_i} to $\mathcal{E}$
- 8: **end for**
- 9: $\tau_{\theta_{\text{root}}}^* \leftarrow \text{Opt. ref. trajectory for } \theta_{\text{root}} \text{ with } A_*$
- 10: $T_{\theta_{\text{root}}}^v = (\tau_{\theta_{\text{root}}, i}^v)_{i=1}^{n_{\text{trial}}} \leftarrow \text{Run } A_v \text{ in } \theta_{\text{root}} \text{ for } n_{\text{trial}} \text{ trials}$
- 11: **RecordDistanceFromRef**($\theta_{\text{root}}, T_{\theta_{\text{root}}}^v, \tau_{\theta_{\text{root}}}^*, \mathcal{D}$)
- 12: **while** **IsStopCriteriaMet**($\mathcal{P}$) = False **do**
- 13: **while** **IsStopCriteriaMet**($\mathcal{P}$) = False and **Size**($\mathcal{P}$) < N_0 **do**
- 14: $k \leftarrow \text{Random sample from } \{0, 1, 2, \dots, m_{\text{extra}}\}$
- 15: $\theta, \tau_{\theta}^* \leftarrow \text{CreateWarmStartScenario}(\theta_{\text{root}}, \tau_{\theta_{\text{root}}}^*, m_{\text{base}} + k, A_*, \mathcal{D}, \mathcal{A}, \mathcal{S}, \mathcal{E})$
- 16: **UpdatePopulation**($\mathcal{P}, A_v, \theta, \tau_{\theta}^*, n_{\text{trial}}$)
- 17: $T_{\theta}^v \leftarrow \text{Get vehicle trajectories in } \theta \text{ from } \mathcal{P}$
- 18: **RecordDistanceFromRef**($\theta, T_{\theta}^v, \tau_{\theta}^*, \mathcal{D}$)
- 19: **end while**
- 20: **while** **IsStopCriteriaMet**($\mathcal{P}$) = False and **IsPopResetCriteriaMet**($\mathcal{P}$) = False **do**
- 21: $\theta_p \leftarrow \text{GetCuratedScenario}(\mathcal{P}, \alpha, p_{\text{rand}})$
- 22: $T_{\theta_p}^v \leftarrow \text{Get vehicle trajectories in } \theta_p \text{ from } \mathcal{P}$
- 23: $\theta, \tau_{\theta}^*, d, E \leftarrow \text{CreateROIChildScenario}(\theta_p, T_{\theta_p}^v, A_*, \mathcal{A}, \mathcal{S}, \mathcal{E}, 0)$
- 24: $\hat{R}(\theta) \leftarrow \text{UpdatePopulation}(\mathcal{P}, A_v, \theta, \tau_{\theta}^*, n_{\text{trial}})$
- 25: $T_{\theta}^v \leftarrow \text{Get vehicle trajectories in } \theta \text{ from } \mathcal{P}$
- 26: **RecordDistanceFromRef**($\theta, T_{\theta}^v, \tau_{\theta}^*, \mathcal{D}$)
- 27: $\hat{R}(\theta_p) \leftarrow \text{Get parent regret from } \mathcal{P}$
- 28: **if** $|\hat{R}(\theta) - \hat{R}(\theta_p)| \geq \epsilon$ **then**
- 29: Append d to $E.d$ ▷ Record action distance
- 30: **end if**
- 31: **UpdateROIEstimators**($\mathcal{E}, L$)
- 32: **end while**
- 33: **if** **IsStopCriteriaMet**($\mathcal{P}$) = False and **IsPopResetCriteriaMet**($\mathcal{P}$) = True **then**
- 34: $\theta_{\text{max}} \leftarrow \text{Max regret scenario from } \mathcal{P}$
- 35: $m_{\text{base}} \leftarrow \text{Creation complexity of } \theta_{\text{max}}$
- 36: Reset $\mathcal{P}$
- 37: **end if**
- 38: **end while**

and (3.2) begs a relaxed and related question: Is it possible to at least approximate and sample from a subset of an action's parameter space such that the mutation increases the regret by at least

Algorithm 3.2 BACRE-3D Inputs

Input: Test agent A_v , reference agent model A_* , RCR level α , root scenario θ_{root} , num. base scenarios N_0 , stopping criteria, num. trials to run n_{trial} , prob. of sampling random parent scenario p_{rand} , action functions $\mathcal{A} = (a_1, \dots, a_n)$, action parameter spaces $\mathcal{S} = (S_{a_1}, \dots, S_{a_n})$, ROI initial search radii $r_1^0, \dots, r_n^0$, ROI learning period L , ROI regret threshold ϵ , num. of extra warm-start mutations m_{extra} , population reset criteria.

Algorithm 3.3 Core BACRE Functions

Input: Population $\mathcal{P}$, test agent A_v , scenario θ , opt. ref. trajectory τ_θ^* , num. trials n_{trial}

Output: The estimated regret $\hat{R}(\theta)$

- 1: **function** **UpdatePopulation**($\mathcal{P}, A_v, \theta, \tau_\theta^*, n_{\text{trial}}$)
 - 2: $T_\theta^v = (\tau_{\theta,i}^v)_{i=1}^{n_{\text{trial}}} \leftarrow \text{Run } A_v \text{ in } \theta \text{ for } n_{\text{trial}} \text{ trials}$
 - 3: Estimate regret $\hat{R}(\theta)$ using Eq. (2.4)
 - 4: Append $(\theta, \tau_\theta^*, T_\theta^v, \hat{R}(\theta))$ to $\mathcal{P}$
 - 5: **return** $\hat{R}(\theta)$
 - 6: **end function**
-

Input: Population $\mathcal{P}$, desired RCR level α , prob. of sampling random parent scenario p_{rand}

Output: Curated scenario θ

- 1: **function** **GetCuratedScenario**($\mathcal{P}, \alpha, p_{\text{rand}}$)
 - 2: Sample $u \sim \mathcal{U}[0, 1]$
 - 3: **if** $u < p_{\text{rand}}$ **then** ▷ Select RCR level
 - 4: $\bar{\alpha} \leftarrow 1$ ▷ Use entire population
 - 5: **else**
 - 6: $\bar{\alpha} \leftarrow \alpha$ ▷ Use desired regret range
 - 7: **end if**
 - 8: $\bar{n} \leftarrow \text{Cutoff for RCR level } \bar{\alpha}$
 - 9: $\Theta_{\text{parents}} \leftarrow \{\theta_{(1)}, \dots, \theta_{(\bar{n})}\}$ from $\mathcal{P}$
 - 10: **return** Uniform random sample $\theta \in \Theta_{\text{parents}}$
 - 11: **end function**
-

some threshold? We first consider the set of action parameters for which an action a in a scenario θ changes the regret by at least some threshold $\epsilon > 0$, which is described by

$$S_{\text{ROI}}(\theta, a, \epsilon) = \left\{ s \in S_a : \left| \hat{R}(a(\theta, s)) - \hat{R}(\theta) \right| \geq \epsilon \right\}. \quad (3.3)$$

The set S_{ROI} describes a regret *region of influence* (ROI), and more specifically, it is the ϵ -ROI. Dependence on the test agent and reference agent is implied, though not explicitly written in (3.3)

to reduce clutter. Since the goal in BACRE is to *increase* regret, of more interest is the positive ϵ -ROI

$$S_{\text{ROI}+}(\theta, a, \epsilon) = \left\{ s \in S_a : \hat{R}(a(\theta, s)) - \hat{R}(\theta) \geq \epsilon \right\}. \quad (3.4)$$

While the set S_a may have multiple dimensions, the *location* at which a structural mutation is applied contributes significantly to the scenario’s structural diversity and difficulty for the test agent. Moving forward, only structural actions (i.e., actions that require a location parameter) are considered in the rest of this chapter. For any structural action a , we can decompose the action parameter space $S_a = S_a^{\text{xy}} \times S'_a$, where $S_a^{\text{xy}} \subset \mathbb{R}^2$ denotes the 2D Cartesian component and S'_a denotes the remainder of the parameter space (if applicable). We then define the Cartesian ϵ -ROI as

$$S_{\text{ROI}}^{\text{xy}}(\theta, a, \epsilon) = \{x \in S_a^{\text{xy}} : (x, s'_a) \in S_{\text{ROI}}(\theta, a, \epsilon)\} \quad (3.5)$$

which is a projection of S_{ROI} onto the XY plane. Similarly, the positive Cartesian ϵ -ROI is

$$S_{\text{ROI}+}^{\text{xy}}(\theta, a, \epsilon) = \{x \in S_a^{\text{xy}} : (x, s'_a) \in S_{\text{ROI}+}(\theta, a, \epsilon)\}. \quad (3.6)$$

Existence of S_{ROI} (and any of its variations) for any (θ, a, ϵ) tuple is not guaranteed, but if it does exist, it would be intractable to compute. Nevertheless, we believe that there exists a practical approximation of $S_{\text{ROI}+}^{\text{xy}}$. The general idea is to *exercise* the autonomy system with local perturbations. The ROI hypothesis is that applying mutations to the scenario within a neighborhood of the vehicle’s trajectory (instead of anywhere randomly) will be more likely to increase the vehicle’s regret and consequently more efficient at discovering the system’s weaknesses. The underlying reasoning is that the more frequently the agent has to “think” in locally different environments, the more likely the agent’s limitations will be exposed. The use of perturbations has already been successfully applied in the on-road literature to create perturbed trajectories from datasets [30] and perturbed sensor viewpoints to augment training data [17, 18], and it is a component in adaptive stress testing [26, 27].

Let $T_\theta^v = (\tau_{\theta,i}^v)_{i=1}^{n_{\text{trial}}}$ denote the vehicle trajectories from the test agent in scenario θ . Let $\mathcal{A} =$

$(a_1, \dots, a_n)$ denote a list of structural action functions and $\mathcal{S} = (S_{a_1}, \dots, S_{a_n})$ be the associated list of the bounded parameter spaces. Given a radius $r \geq 0$, we define the set

$$U(T_\theta^v, r) = \bigcup_{\tau \in T_\theta^v} \bigcup_{x \in \tau} \{p \in \mathbb{R}^2 : \|p - \text{pos}^{\text{xy}}(x)\|_2 \leq r\}, \quad (3.7)$$

where $\text{pos}^{\text{xy}}(x) \in \mathbb{R}^2$ is the 2D position from a trajectory state x . The set U is just the union of circles of radius r in the XY plane centered along each point from the vehicle's trajectories. But more importantly, U is intended to be a practical guide for sampling from $S_{\text{ROI}+}^{\text{xy}}$. If $S_{\text{ROI}+}^{\text{xy}}(\theta, a_i, \epsilon)$ exists for an action $a_i \in \mathcal{A}$ and some $\epsilon > 0$, then it is bounded because $S_{a_i}^{\text{xy}}$ is bounded. Thus, there exists an $r_{\max} > 0$ such that $S_{\text{ROI}+}^{\text{xy}}(\theta, a_i, \epsilon) \subseteq U(T_\theta^v, r_{\max})$. However, $S_{\text{ROI}+}^{\text{xy}}$ will most likely be an irregular shape and/or a union of disjointed sets, and the intersection-over-self (IOS) ratio

$$\rho(U(T_\theta^v, r), S_{\text{ROI}+}^{\text{xy}}(\theta, a_i, \epsilon)) \quad (3.8)$$

is likely to be rather small when $r \geq r_{\max}$. The IOS ratio is defined as

$$\rho(A, B) = \frac{\text{Area}(A \cap B)}{\text{Area}(A)} \quad (3.9)$$

where $A, B \subset \mathbb{R}^2$. The goal is to find the radius that maximizes (3.8).

While this is an intractable task, we can leverage two ideas that will make it more manageable. First, we impose a requirement that mutations be *reversible* (for any two scenarios θ_1 and θ_2 , there exists a mutation that can transform θ_1 to θ_2 and vice versa). Consequently, a mutation that *decreases* the regret by at least $\epsilon > 0$ is just as useful as one that *increases* the regret by at least ϵ . If we also consider the fact that

$$S_{\text{ROI}+}^{\text{xy}} \subseteq S_{\text{ROI}}^{\text{xy}} \implies \text{Area}(S_{\text{ROI}+}^{\text{xy}}) \leq \text{Area}(S_{\text{ROI}}^{\text{xy}}),$$

then we observe that a random location $x \in S_a^{\text{xy}}$ is more likely to lie in $S_{\text{ROI}}^{\text{xy}}$ than $S_{\text{ROI}+}^{\text{xy}}$. When

a scenario θ is chosen for mutation, the proposed ROI mutation mechanism constrains the action locations to lie in $U(T_\theta^v, r)$, where r is termed the controlling search radius (CSR). Whenever a mutation changes the regret by at least $\epsilon > 0$, the distance between the action's location and T_θ^v is recorded, and the CSR is periodically updated. The goal is to effectively learn a CSR online such that $U(T_\theta^v, r)$ *in general* overlaps the part of $S_{\text{ROI}}^{\text{xy}}$ nearest T_θ^v , which can indirectly increase the chances of sampling from $S_{\text{ROI}+}^{\text{xy}}$. While in some cases $\text{Area}(S_{\text{ROI}+}^{\text{xy}}) \ll \text{Area}(S_{\text{ROI}}^{\text{xy}})$ and the true IOS ratio in (3.8) will vary as θ changes, the perceived benefits of this approach are expected to outweigh the risks.

Algorithm 3.4 and Algorithm 3.5 describes the core ROI mutation functions, such as **CreateROIChildScenario** and **UpdateROIEstimators**, which perform the above task. The middle diagram in Fig. 3.2 depicts an improved version of BACRE, called BACRE with ROI, which illustrates how this mechanism alone is utilized. The main differences between BACRE with ROI and the original version are that:

1. Every action has an estimator object associated with it.
2. The base population contains a root scenario from which the remaining base scenarios are created from. This root scenario, however, is detached from the base population in BACRE-3D.
3. The mutation method for creating child scenarios is guided by the online-approximated ROI.

The **CreateROIChildScenario** function outlines the process for creating a child scenario using the ROI approximation for the chosen action. Several steps require some commentary:

1. Sampling a search radius (lines 8-9): A radius r is sampled from $\mathcal{N}(E_a.r, E_a.r/3)$ to determine if a better CSR exists, where E_a is the action's estimator object and $E_a.r$ is the estimator's CSR. The standard deviation of $E_a.r/3$ is subjectively chosen.
2. Sampling from $U(T_\theta^v, r)$ (lines 10-12): In line 10, **SelectControlPoint**(θ, T_θ^v) returns a random 2D Cartesian point c_1 from T_θ^v that lies outside the obstacle-free radii in θ . In line 11,

r_{ext} refers to the extension radius and is discussed in Section 3.2.2. For now, $c_2 = c_1$ because $r_{\text{ext}} = 0$. In line 12, the set $\bar{B}(c, r) = \{x \in \mathbb{R}^2 : \|x - c\|_2 \leq r\}$ is the closed Euclidean ball in $\mathbb{R}^2$ with center $c \in \mathbb{R}^2$ and radius $r \geq 0$. Although c_3 is not *uniformly* randomly sampled from $U(T_\theta^v, r)$, this sampling method is the simplest to perform.

3. Ensuring feasibility (lines 6, 15): **IsFeasible**(θ', A_*) returns a flag indicating if the scenario θ' is feasible and the corresponding optimal reference trajectory $\tau_{\theta'}^*$. If θ' is None, then flag is False.
4. The action distance (line 17): In order to update the ROI approximation, the action distance d is required. For most elementary actions, the location of the mutation $(a, (c_3, s'_a))$ will be equal to the action location c_3 , but for more advanced/stochastic actions this may not be true.

Fig. 3.3 provides an illustration of the key steps in **CreateROIChildScenario**.

To perform the ROI approximation update (lines 28-31 of Algorithm 3.1), the child scenario's regret is compared to that of the parent, and if the difference exceeds the threshold ϵ , then the action distance is recorded by the associated estimator. Finally, the function **UpdateROIEstimators** is called to update the estimator's CSR every L iterations to control how quickly the estimate is updated.

A few modifications are made to BACRE-3D if the environment can have discrete physical obstacles, such as trees or bushes. When it comes to adding, moving, or removing such discrete obstacles, all three of these actions will *share* the same ROI estimator object. In the case of removing an obstacle, lines 8-12 of **CreateROIChildScenario** are ignored.

Finally, one limitation of the ROI mutation mechanism is that only *one* mutation can be applied when creating child scenarios. If multiple mutations were applied and the regret changed by at least ϵ , then it would be very difficult to determine which mutation was responsible. A way around this, though, is to define an “advanced” action that performs multiple elementary mutations within a local region (the red noise sculpt tool in Section 3.3.1 is one such example).

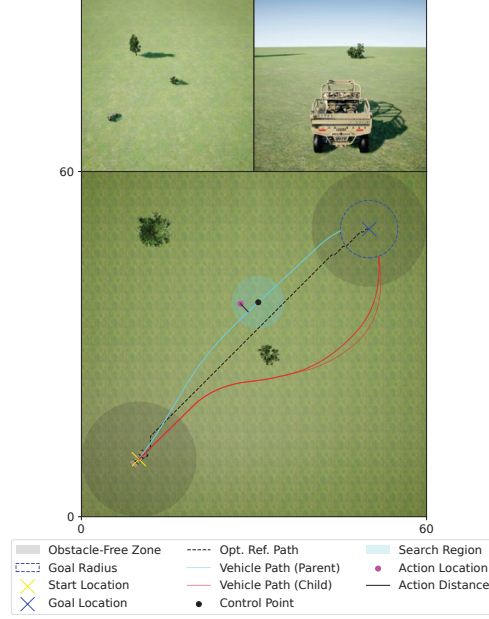


Figure 3.3: An illustration of the ROI mutation mechanism. In this example, the action is to add a bush to the parent scenario shown. To create a mutation, a control point c_1 (black dot) is first selected from part of the parent vehicle path (cyan) that lies outside the obstacle-free radii. The cyan circle centered at c_1 shows the search region from which an action location can be chosen from. The magenta dot shows the selected action location c_2 , and the black line from c_2 to the parent vehicle path represents the action distance. The vehicle paths from the resulting child scenario are shown in red.

3.2.2 Avoiding Local Maxima

Like with any optimization algorithm, BACRE can get trapped in local maxima. The original design in BACRE makes a trade-off between exploration (via random mutations and occasionally choosing uniformly random parents) and exploitation (by curating the highest regret scenarios). While the ROI mutation mechanism focuses on improving BACRE’s exploratory behavior, there is no guarantee that a mutation will increase the regret. It might even take *many* mutations before the regret increases. This could certainly occur if the base population is too simple for the test system and/or if the “size” of the mutation is rather small. This problem must be addressed if BACRE is to generate adversarial scenarios in large search spaces.

The desire with BACRE is to find scenarios that are representative of the *simplest* adversarial scenarios for the test system. From a graph theory perspective, we can represent the set of all

Algorithm 3.4 ROI Mutation Functions (Part 1)

Input: Initial search radius r_0 **Output:** Initialized ROI estimator E

```
1: function CreateROIEstimator( $r_0$ )
2:   Create estimator object  $E$ 
3:    $E.i \leftarrow 0$  ▷ Iteration counter
4:    $E.d \leftarrow \text{List}$  ▷ List of action distances
5:   Append  $r_0$  to  $E.d$ 
6:    $E.r \leftarrow r_0$  ▷ Controlling search radius
7:   return  $E$ 
8: end function
```

Input: ROI Estimators $\mathcal{E}$, learning period L

```
1: function UpdateROIEstimators( $\mathcal{E}, L$ )
2:   for each  $E \in \mathcal{E}$  do
3:      $E.i \leftarrow E.i + 1$ 
4:     if  $E.i \% L = 0$  then ▷ Modulo operator
5:        $E.r \leftarrow \text{Mean}(E.d)$  ▷ Take average
6:     end if
7:   end for
8: end function
```

possible scenarios with an infinite tree in which every scenario (the nodes in the tree) has an infinite number of children, and so forth. This is like the trees in Fig. 3.2, except infinite. The simplest failure scenario could be found with a breadth-first search approach, but the infinite number of children poses a challenge. On the other hand, while a depth-first search approach is impossible (due to infinite depth), starting the search from a higher depth (i.e., a more complex scenario) can increase the chance that a mutation will increase the regret. But choosing an appropriate start depth is challenging because the test system is a black-box. The proposed solution to these problems is to periodically adapt BACRE's behavior to the test system's behavior to maintain a sufficiently complex and diverse population. This can be done by improving how BACRE handles both exploration *and* exploitation. These improvements must also work in tandem with the ROI mutation mechanism.

The first improvement is to *vary* the number of mutations when creating the base population to improve its diversity. The original method creates a base population by applying a *single* mutation

Algorithm 3.5 ROI Mutation Functions (Part 2)

Input: Parent scenario θ , vehicle trajectories T_θ^v , opt. ref. agent A_* , action functions $\mathcal{A}$, action parameter spaces $\mathcal{S}$, ROI Estimators $\mathcal{E}$, extension radius r_{ext}

Output: Child scenario θ' , opt. ref. trajectory $\tau_{\theta'}^*$, action distance d , associated ROI estimator E_a

```
1: function CreateROIChildScenario( $\theta, T_\theta^v, A_*, \mathcal{A}, \mathcal{S}, \mathcal{E}, r_{\text{ext}}$ )
2:    $a \leftarrow$  Select an action from  $\mathcal{A}$ 
3:    $S_a^{\text{xy}}, S'_a \leftarrow$  Parameter spaces for  $a$  from  $\mathcal{S}$ 
4:    $E_a \leftarrow$  ROI estimator for  $a$  from  $\mathcal{E}$ 
5:    $\theta' \leftarrow$  None
6:    $\text{flag}, \tau_{\theta'}^* \leftarrow \text{IsFeasible}(\theta', A_*)$ 
7:   while flag is False do
8:     Sample  $r \sim \mathcal{N}(E_a.r, E_a.r/3)$ 
9:      $r \leftarrow \text{Max}(r, 0)$  ▷ Must be nonnegative
10:     $c_1 \leftarrow \text{SelectControlPoint}(\theta, T_\theta^v)$ 
11:     $c_2 \leftarrow$  Random point from  $\partial \bar{B}(c_1, r_{\text{ext}})$ 
12:     $c_3 \leftarrow$  Random point from  $\bar{B}(c_2, r)$ 
13:     $s'_a \leftarrow$  Sample random parameters from  $S'_a$ 
14:     $\theta' \leftarrow a(\theta, (c_3, s'_a))$  ▷ Create child scenario
15:     $\text{flag}, \tau_{\theta'}^* \leftarrow \text{IsFeasible}(\theta', A_*)$ 
16:  end while
17:   $d \leftarrow$  Distance between the location of the mutation  $(a, (c_3, s'_a))$  and  $T_\theta^v$ 
18:  return  $\theta', \tau_{\theta'}^*, d, E_a$ 
19: end function
```

to a “blank” scenario. However, if the test agent is quite capable, then applying mutations to this “simplistic” base population is unlikely to affect the vehicle’s performance. Increasing the initial diversity is expected to help remedy this. The integer parameter $m_{\text{extra}} \geq 0$ controls this added diversity in Algorithm 3.1.

The second improvement is to periodically *reset* the entire population to prevent it from saturating with scenarios that are closely related to each other and do not increase the vehicle’s regret. If various criteria are met, then the population is entirely reset (i.e., thrown out) and a new base population is created. This new base population will contain carefully crafted scenarios generated with the same *creation complexity* as that of the previous population’s maximum regret scenario. Creation complexity refers to the number of mutations used to create a given scenario from the root scenario. The goal is to start afresh with a more complex base population, but without any

direct connection to the previous population’s highest regret scenarios to avoid getting trapped in similar local maxima. This is in contrast to a genetic algorithm in which each population carries over the best data points [75]. The function **IsPopResetCriteriaMet** determines if the population should be reset. The following criteria is used in BACRE-3D:

1. If the population reaches a maximum size $N_{\max}$.
2. If the maximum regret has not changed in N_{idle} iterations.
3. If a tree level contains at least $N_{\max}^{\text{lev1}}$ scenarios.
4. If two neighboring tree levels contain at least $N_{\max}^{\text{lev2}}$ scenarios combined.

Algorithm 3.6 describes the functions **CreateWarmStartScenario** and **RecordDistanceFromRef** which are specific to this population reset mechanism.

The base scenarios in BACRE-3D are created with the **CreateWarmStartScenario** function whose goal is to quickly create a base scenario with m mutations from the root scenario. While the ROI mutation method is the preferred approach for applying mutations, it would require running m (expensive) vehicle simulations, one per mutation. A way around this is to use a surrogate trajectory that is cheaper to compute, namely the optimal reference trajectory τ_{θ}^* , plus a model mismatch correction. The **CreateROIChildScenario** function first finds a control point c_1 on τ_{θ}^* . Then, a secondary control point c_2 is found by sampling from a circle of radius r_{ext} to account for the discrepancy between τ_{θ}^* and (unknown) τ_{θ}^v . This extension radius r_{ext} is a learned value that measures the average distance τ_{θ}^v is from τ_{θ}^* in the XY plane for a given creation complexity. In line 11 of **CreateROIChildScenario**, the set $\partial \bar{B}(c_1, r_{\text{ext}}) = \{x \in \mathbb{R}^2 : \|x - c_1\|_2 = r_{\text{ext}}\}$ denotes the boundary of a 2D Euclidean ball (or the singleton $\{c_1\}$ in the case $r_{\text{ext}} = 0$). The if-block in **CreateWarmStartScenario** selects the most appropriate value for r_{ext} from the look-up table (the dictionary $\mathcal{D}$).

The function **RecordDistanceFromRef** computes and stores the above path distances from τ_{θ}^v to τ_{θ}^* in the dictionary $\mathcal{D}$. Let $p = (x_1, \dots, x_n)$ denote a path of ordered points $x_i \in \mathbb{R}^d$ (in this

case $d = 2$). Given two paths p_1 and p_2 with n_1 and n_2 points, respectively, the function

$$\text{pathdist}(p_1, p_2) = \frac{1}{n_1} \sum_{x \in p_1} \left[\min_{y \in p_2} \|x - y\|_2 \right] \quad (3.10)$$

computes the average distance of a point $x \in p_1$ from p_2 . Note that in general, $\text{pathdist}(p_1, p_2) \neq \text{pathdist}(p_2, p_1)$.

One subtlety of BACRE-3D is that the root scenario is omitted from the base population, unlike in BACRE with ROI. The root scenario was originally included in the base population, however, in preliminary experiments the maximum regret scenario was not consistently related to the warm-start base scenarios because the warm-start scenarios occasionally had *lower* regret than that of the root scenario. This allowed for an additional branch from the root scenario to be created, which then resulted in numerous fluctuations of m_{base} between populations hindering the algorithm's performance. To ensure that m_{base} is non-decreasing between populations, the root scenario is excluded from the base population.

The right-most diagram in Fig. 3.2 provides an illustration of how BACRE-3D operates. The feedback loop with the ROI mutation mechanism and population reset mechanism positions BACRE-3D as the most advanced and capable version of BACRE compared to its predecessors.

3.3 Methodologies for Defining Mutations and Regret for Off-Road AVs

This section discusses specific methodologies in the application of BACRE-3D to off-road AVs. Section 3.3.1 discusses how to generate diverse 3D terrains through mutations, and Section 3.3.2 discusses an improved regret metric for evaluating AV performance in off-road navigation scenarios.

3.3.1 Generating 3D Terrain Mutations

The creation of 3D terrains is entirely accomplished through actions that modify the environment's heightmap. These actions are called sculpting tools, and the proposed tools only modify the en-

Algorithm 3.6 Population Reset Functions

Input: Root scenario θ_{root} , opt. ref. trajectory $\tau_{\theta_{\text{root}}}^*$, num. mutations to apply $m \geq 1$, opt. ref. agent A_* , distance dictionary $\mathcal{D}$, action functions $\mathcal{A}$, action parameter spaces $\mathcal{S}$, ROI Estimators $\mathcal{E}$

Output: Child scenario θ , opt. ref. trajectory τ_θ^*

```
1: function CreateWarmStartScenario( $\theta_{\text{root}}, \tau_{\theta_{\text{root}}}^*, m, A_*, \mathcal{D}, \mathcal{A}, \mathcal{S}, \mathcal{E}$ )
2:    $\theta \leftarrow \theta_{\text{root}}$ 
3:    $\tau_\theta^* \leftarrow \tau_{\theta_{\text{root}}}^*$ 
4:    $r_{\text{ext}} \leftarrow 0$ 
5:   for  $i = 0$  to  $m - 1$  do
6:     if  $i$  is in  $\mathcal{D}.\text{Keys}()$  then
7:        $r_{\text{ext}} \leftarrow \text{Mean}(\mathcal{D}.\text{Value}(i))$  ▷ Take average
8:     end if
9:      $\theta, \tau_\theta^*, d, E \leftarrow \text{CreateROIChildScenario}(\theta, (\tau_\theta^*), A_*, \mathcal{A}, \mathcal{S}, \mathcal{E}, r_{\text{ext}})$ 
10:  end for
11:  return  $\theta, \tau_\theta^*$ 
12: end function
```

Input: Scenario θ , vehicle trajectories T_θ^v , opt. ref. trajectory τ_θ^* , distance dictionary $\mathcal{D}$

```
1: function RecordDistanceFromRef( $\theta, T_\theta^v, \tau_\theta^*, \mathcal{D}$ )
2:    $d \leftarrow \text{List}$ 
3:    $m \leftarrow \text{Creation complexity of } \theta$ 
4:    $p^* \leftarrow \text{2D Cartesian path from } \tau_\theta^*$ 
5:   for each  $\tau \in T_\theta^v$  do
6:      $p^v \leftarrow \text{2D Cartesian path from } \tau$ 
7:     Append pathdist( $p^v, p^*$ ) to  $d$  using Eq. (3.10)
8:   end for
9:    $\bar{d} \leftarrow \text{Mean}(d)$ 
10:  if  $m$  not in  $\mathcal{D}.\text{Keys}()$  then
11:     $\mathcal{D}.\text{Insert}(m, \text{List}(\bar{d}))$  ▷ Insert new key-value pair
12:  else
13:    Append  $\bar{d}$  to the list at  $\mathcal{D}.\text{Value}(m)$ 
14:  end if
15: end function
```

vironment *locally* following the idea of local perturbations presented in Section 3.2.1. Repeated application of these tools in different locations and with varied parameters results in the creation of diverse/complex 3D environments. This is in contrast to [49–52] which define the geometry of the *global* environment, typically using fractals.

Three elementary sculpting tools are proposed. Each tool has a center point or axis that controls

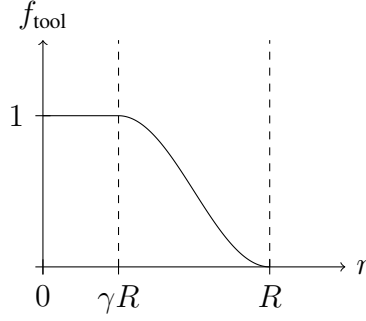


Figure 3.4: A visualization of the tool’s local strength function $f_{\text{tool}}(r; \gamma, R)$ for some $R > 0$ and $0 < \gamma < 1$.

how the environment changes within a radius/distance of $R > 0$. Each tool also has a radial strength parameter $0 \leq \gamma \leq 1$ that controls where the tool has “full strength” or maximal intensity. The tool’s local strength at a given distance r from the center point/axis is determined by the function $f_{\text{tool}} : \mathbb{R}_+ \rightarrow [0, 1]$ defined by

$$f_{\text{tool}}(r; \gamma, R) = \begin{cases} 1, & 0 \leq r \leq \gamma R \\ 0.5 \cos \left(\pi \frac{r - \gamma R}{R(1 - \gamma)} \right) + 0.5, & \gamma R < r < R \\ 0, & r \geq R, \end{cases} \quad (3.11)$$

with parameters γ and R , and is visualized in Fig. 3.4.

There are two circular-based sculpting tools. The simplest circular tool is the circular sculpt tool, shown in Fig. 3.5a, which modifies the environment’s heightmap within a circle of radius $r > 0$ centered at $c \in \mathbb{R}^2$. The height at any location $p \in \mathbb{R}^2$ in the terrain is given the perturbation

$$\Delta z_{\text{circ}}(p) = f_{\text{tool}}(\|p - c\|_2; \gamma, r) \times h, \quad (3.12)$$

where $h \neq 0$ is the maximum height change at the center.

The second tool is the (circular) red noise tool, shown in Fig. 3.5b. To simulate red noise, the tool applies a Gaussian kernel to smooth a grid of white noise and then interpolates the z coordinate change. Specifically, the tool first creates the smallest grid with a noise spatial resolution (i.e., grid

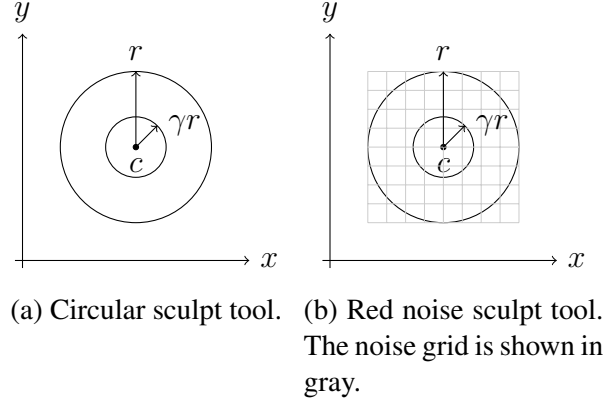


Figure 3.5: The circular-based elementary sculpting tools.

spacing) $d > 0$ that circumscribes the circle at center c with radius r . Each point in this grid is assigned a value $u \sim \mathcal{U}[-1, 1]$. Then, a Gaussian kernel of (odd) size k and kernel sigma $\sigma > 0$ are applied over this grid n times, with a renormalization step after each filter pass to keep the values between -1 and 1. The height at any location $p \in \mathbb{R}^2$ in the terrain is then given the perturbation

$$\Delta z_{\text{red}}(p) = f_{\text{tool}}(\|p - c\|_2; \gamma, r) \times h \times \text{filt}(p), \quad (3.13)$$

where $h > 0$ is the scaling factor in the z direction and $\text{filt}(p)$ denotes the value from the filtered noise grid using bilinear interpolation.

The third and final elementary sculpting tool is the rectangular sculpt tool shown in Fig. 3.6. This tool modifies a rectangular region of the environment defined by a rectangle with center line from a to b , where $a, b \in \mathbb{R}^3$ and width $w > 0$. The height at any point $p \in \mathbb{R}^2$ that lies between a and b in the rectangular region (shown in red in Fig. 3.6) is given the perturbation

$$\Delta z_{\text{rect,main}}(p) = f_{\text{tool}}(\bar{y}_p; \gamma, w/2) \times h_{ab}(p), \quad (3.14)$$

where

$$h_{ab}(p) = a_z + \frac{\bar{x}_p(b_z - a_z)}{\|(a_x, a_y) - (b_x, b_y)\|_2}, \quad (3.15)$$

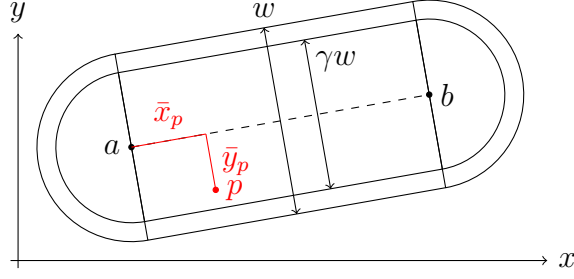


Figure 3.6: Elementary rectangular sculpt tool. The points a and b have a z -component which is not shown. The endcaps are optional.

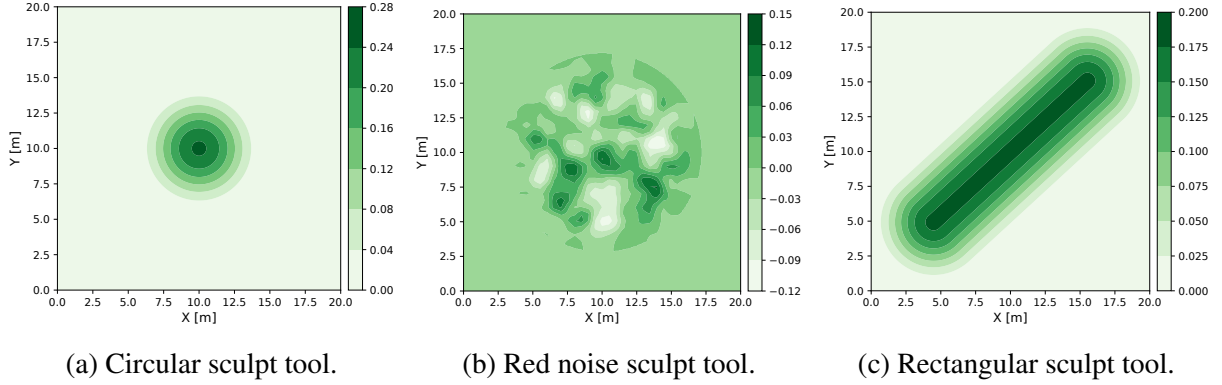


Figure 3.7: Sample mutations from each type of elementary sculpt tool applied to a 20 meter $\times$ 20 meter area. The color bar denotes the height perturbation in meters.

and $\bar{x}_p$ and $\bar{y}_p$ are the longitudinal and lateral distances, respectively, from a to b as shown in the figure. The two endcaps (semi-circles) are optional. If included, then the height at any location $p \in \mathbb{R}^2$ in endcap a is given the perturbation

$$\Delta z_{\text{rect},a}(p) = f_{\text{tool}}(\|p - (a_x, a_y)\|_2; \gamma, w/2) \times a_z, \quad (3.16)$$

and similarly for endcap b

$$\Delta z_{\text{rect},b}(p) = f_{\text{tool}}(\|p - (b_x, b_y)\|_2; \gamma, w/2) \times b_z. \quad (3.17)$$

Fig. 3.7 provides a sample mutation from each of these elementary sculpt tools. To increase the diversity of the scenarios that can be created, multiple sets of parameter spaces can be defined

for each tool, with each set defining different ranges of values for the required parameters. Furthermore, more advanced sculpting tools can be created through repeated application of a specific elementary sculpt tool with parameters determined from given ranges and the center points being chosen randomly from within a given circle in the environment.

Because the goal is to create realistic-looking 3D terrains, BACRE-3D must be constrained to ensure it does not create unrealistic spikes in the terrain (in either the positive or negative z direction). The simplest approach is to add a constraint to the **IsFeasible** function. Since any given terrain could have rough/uneven areas, simply searching for a local or global optima in the terrain's elevation will not necessarily indicate an abnormal spike. A better indicator is if the elevation at a particular point is vastly different from points that are a certain distance away. Such points are considered *pseudo-local optima* (see Definition 3.3.1). Fig. 3.8 provides an illustration for a pseudo-local maxima for the 1D case.

Definition 3.3.1 (Pseudo-Local Maxima/Minima) *Let $f : A \rightarrow \mathbb{R}$ define a surface over $A \subseteq \mathbb{R}^n$. Consider a point $p \in A$, radius $r > 0$, and height threshold $h > 0$. Let*

$$\mathcal{C}(p) = \{x \in A : \|x - p\|_2 = r\},$$

and let $\bar{f}$ be the average value of f over the set $\mathcal{C}(p)$. The point p is a pseudo-local maxima of radius r and height threshold h if $f(p) > f(x) \forall x \in \mathcal{C}(p)$ and $f(p) - \bar{f} \geq h$. Similarly, the point p is a pseudo-local minima of radius r and height threshold h if $f(p) < f(x) \forall x \in \mathcal{C}(p)$ and $\bar{f} - f(p) \geq h$.

It is recommended to add constraints that search for pseudo-local optima with multiple radii and height thresholds to best filter out different sized unrealistic spiky areas. If any pseudo-local optima are identified, then the scenario is rejected and a new one must be created.

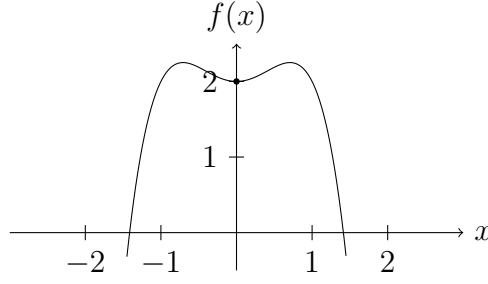


Figure 3.8: An illustration of a pseudo-local maxima (1D case). Consider the function $f(x) = -x^4 + x^2 + 2$ with zeros at $x = \pm\sqrt{2}$. Although $(0, 2)$ is a local minima, it can be considered a pseudo-local maxima with $r = \sqrt{2}$ and $h = 2$ (this is just one of many (r, h) pairs that work for this point).

3.3.2 Regret

Finding failure scenarios is generally a very difficult task. Repeatedly perturbing scenarios to slowly push a vehicle to its limits (as done in BACRE) is a good strategy, but this only works if the evaluation metric accounts for the vehicle being pushed to its limit (or to failure). Knowing *which* scenarios to perturb is just as important as knowing *where* to perturb them.

Because BACRE uses a single metric (regret) to identify candidate parents for mutating, this metric should be as rich in information as possible. In the case of navigation in unstructured 3D terrain, vehicle failure often occurs from either rollover, getting stuck, or from obstacle collisions. However, the regret metric (2.8) will not necessarily assign a high regret to scenarios for which the vehicle comes *close* to failures arising from its orientation. This is partially due to the use of a cost difference with respect to the reference agent’s trajectory, and it also depends on how sophisticated the reference agent model is. For instance, the A* reference agent used in Section 2.5 only finds a reference path of 2D waypoints and cannot account for vehicle orientation, which renders it useless for identifying scenarios that induce rollover.

While BACRE was not intended to include any risk-based terms, the inclusion of various *risk of failure* metrics is expected to improve the effectiveness of the fitness function. This idea is borrowed from the widespread use of risk in the on-road AV literature [19]. In the case of off-road AVs, the proposed risk of failure metrics that should be added are: roll risk, angular acceleration

risk (x,y, and z), and a linear z-acceleration risk. Each of these terms is computed by dividing the quantity in question by a reference value (and possibly followed with exponentiation). These terms are considered most important for assessing the risk for a given vehicle trajectory. The roll risk for a vehicle trajectory τ_θ^v is defined as

$$r_{\text{roll}}(\tau_\theta^v) = \left(\frac{|\text{MaxRoll}(\tau_\theta^v)|}{\pi/2} \right)^{1.7} \quad (3.18)$$

because the maximum possible roll angle is $\pi/2$ (though, the critical angle at which rollover occurs for the test system could also be used). The exponent of 1.7 is subjectively chosen because the lower roll angle range (0-20 degrees) need not hold equal weighting as the upper roll range (70-90 degrees). The reference values for each acceleration risk of failure term is determined by the maximum acceleration from a reference sinusoid. If the reference sinusoid takes the form of $x(t) = A \sin(\omega t)$ with amplitude $A > 0$ and angular frequency $\omega > 0$, the maximum acceleration of $x(t)$ is $A\omega^2$. Thus, the angular x-acceleration risk is defined as

$$r_{\text{angx}}(\tau_\theta^v) = \frac{|\text{MaxAngAccel}_x(\tau_\theta^v)|}{A_{\text{angx}}\omega_{\text{angx}}^2}, \quad (3.19)$$

the angular y-acceleration risk is defined as

$$r_{\text{angy}}(\tau_\theta^v) = \frac{|\text{MaxAngAccel}_y(\tau_\theta^v)|}{A_{\text{angy}}\omega_{\text{angy}}^2}, \quad (3.20)$$

the angular z-acceleration risk is defined as

$$r_{\text{angz}}(\tau_\theta^v) = \frac{|\text{MaxAngAccel}_z(\tau_\theta^v)|}{A_{\text{angz}}\omega_{\text{angz}}^2}, \quad (3.21)$$

and the linear z-acceleration risk is defined as

$$r_{\text{linz}}(\tau_\theta^v) = \frac{|\text{MaxLinAccel}_z(\tau_\theta^v)|}{A_{\text{linz}}\omega_{\text{linz}}^2}. \quad (3.22)$$

The final proposed regret for off-road AVs driving in unstructured environments is the sum of the original regret metric (2.8), which will be denoted $r_{\text{orig}}(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*)$, plus the risk metrics defined above:

$$r_{\text{aug}}(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*) = r_{\text{orig}}(\theta, \bar{\tau}_\theta^v, \bar{\tau}_\theta^*) + r_{\text{roll}}(\tau_\theta^v) + r_{\text{angx}}(\tau_\theta^v) + r_{\text{angy}}(\tau_\theta^v) + r_{\text{angz}}(\tau_\theta^v) + r_{\text{linz}}(\tau_\theta^v). \quad (3.23)$$

This final metric may be referred to as an *augmented* regret metric.

3.4 Numerical Experiments and Results

Various numerical experiments are presented to demonstrate that BACRE-3D is the most effective tool in the BACRE family for finding adversarial scenarios for an off-road AV. The computer hardware used in these experiments consists of two Windows 10 computers (with an Intel i9 CPU and a GeForce RTX 3090 GPU) and two Ubuntu computers (with an Intel Xeon W-2275 CPU and a Quadro RTX 8000 GPU), though not all four computers were needed for each experiment.

3.4.1 Region of Influence: Validation and a Comparison

This section details two experiments¹ with regards to the ROI mutation mechanism: a validation experiment and a comparison experiment. Both experiments utilize the exact same simulation platform and parameters, test vehicle, and BACRE configuration from the comparison experiments in Section 2.5.

The goal of the validation experiment is to validate the hypothesis from Section 3.2.1 that applying mutations within a neighborhood of the vehicle’s trajectory is more likely to increase the vehicle’s regret than applying mutations anywhere. The procedure involves using brute-force to find the positive Cartesian ϵ -ROI $S_{\text{ROI}+}^{\text{xy}}$, defined in (3.6), for various scenarios for an action a of adding a tree. The test scenarios are three scenarios from the same maximum regret family tree from one of the comparison experiment trials in Section 2.5. While the true parameter space for

¹Code and implementation details will be added to https://github.com/tsender/bacre_2D once the journal paper on this chapter is published (expected late 2025).

this action is the continuous set $S_a = S_a^{\text{xy}} = [0, L]^2 \subset \mathbb{R}^2$, where $L = 60$ meters is the landscape size, a discretized subset of S_a^{xy} is used for evaluating the action at strategic locations to reduce the computational cost. This discretized search space for a scenario θ is given by

$$S_a^{\text{xy}}(d, r, \theta) = S_a^{\text{xy}}(d) \cap U(T_\theta^v, r) \subset S_a^{\text{xy}}, \quad (3.24)$$

where $S_a^{\text{xy}}(d)$ denotes the discretization of S_a^{xy} with grid spacing $d > 0$, and $U(T_\theta^v, r)$ is defined in (3.7) with radius $r > 0$. In the experiment, $d = 0.5$ meters and $r = 10$ meters.

Fig. 3.9 depicts the three test scenarios (top) and various components from the regret ROI (bottom) from this validation experiment. In the right figures, the expression $\Delta R = \hat{R}(a(s, \theta)) - \hat{R}(\theta)$ for a scenario θ with $s \in S_a^{\text{xy}}(0.5, 10, \theta)$. Thus, the region labeled $\Delta R \geq 0.1$ (light green) corresponds to the discrete set $\hat{S}_{\text{ROI}+}^{\text{xy}}$ with $\epsilon = 0.1$ using (3.6), where the $\hat{\cdot}$ indicates usage of $S_a^{\text{xy}}(0.5, 10, \theta)$ instead of S_a^{xy} . The two regions $\Delta R \geq 0.1$ and $\Delta R \leq -0.1$ together form $\hat{S}_{\text{ROI}}^{\text{xy}}$ with $\epsilon = 0.1$ using (3.5). Note that $\hat{S}_{\text{ROI}+}^{\text{xy}}$ and $\hat{S}_{\text{ROI}}^{\text{xy}}$ are subsets of their true counterpart. In all three examples, the points from $\hat{S}_{\text{ROI}+}^{\text{xy}}$ lie rather close to the vehicle trajectories from the original scenario. Fig. 3.10 plots the average distance that points in $\hat{S}_{\text{ROI}+}^{\text{xy}}$ and $\hat{S}_{\text{ROI}}^{\text{xy}}$ are to the vehicle trajectories T_θ^v for several regret thresholds ϵ as a function of the search radius r . The convergence of the distance curves associated with $\hat{S}_{\text{ROI}+}^{\text{xy}}$ (left plots), with the exception of $\epsilon = 0.05$ for scenario 2, indicate this distance is less than 2 meters. Since 2 meters is shorter than the vehicle's length, these results support the ROI hypothesis. The lack of convergence in the distance curves associated with $\hat{S}_{\text{ROI}}^{\text{xy}}$ in Fig. 3.10 (right) is irrelevant here.

Although Fig. 3.10 shows that $\hat{S}_{\text{ROI}+}^{\text{xy}}$ lies quite close to the vehicle trajectories, this plot does not indicate the difficulty of *sampling* from the true $S_{\text{ROI}+}^{\text{xy}}$. Fig. 3.11 shows just how difficult it is to sample from $\hat{S}_{\text{ROI}+}^{\text{xy}}$ using various search radii and how comparatively easy it is to instead sample from $\hat{S}_{\text{ROI}}^{\text{xy}}$. This plot shows several important trends: (1) the probability of sampling from the ROI sets decreases as the search radius increases, (2) this probability decreases as the regret threshold ϵ increases, and (3) the probability of sampling from $\hat{S}_{\text{ROI}+}^{\text{xy}}$ appears to attain its maximum value at

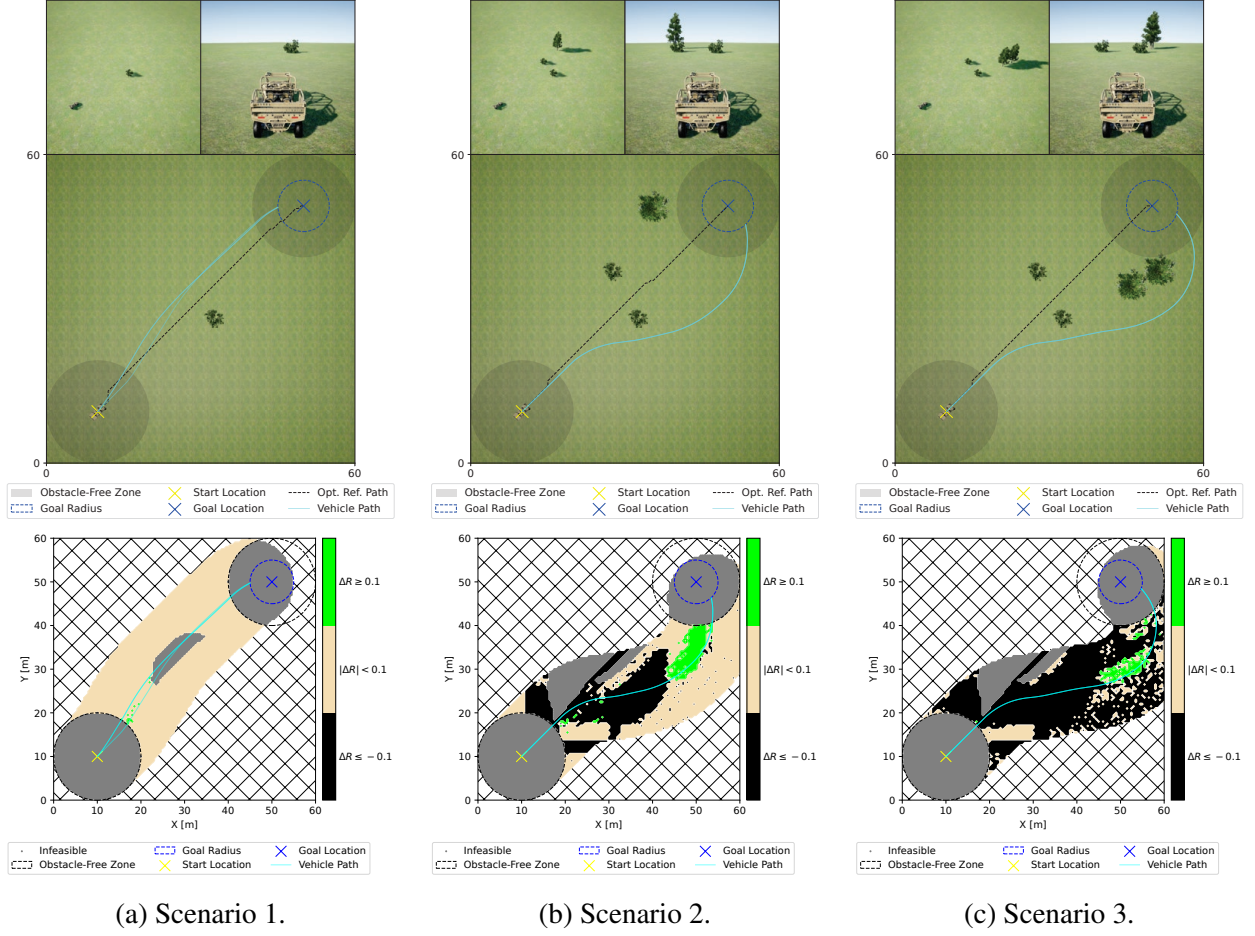
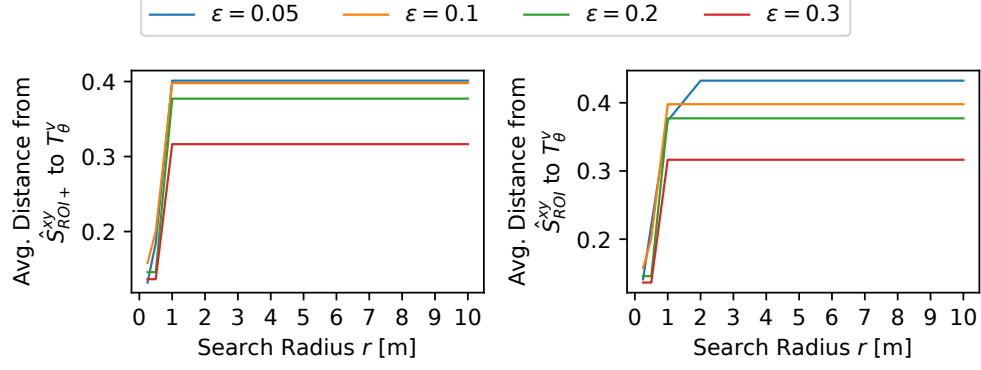
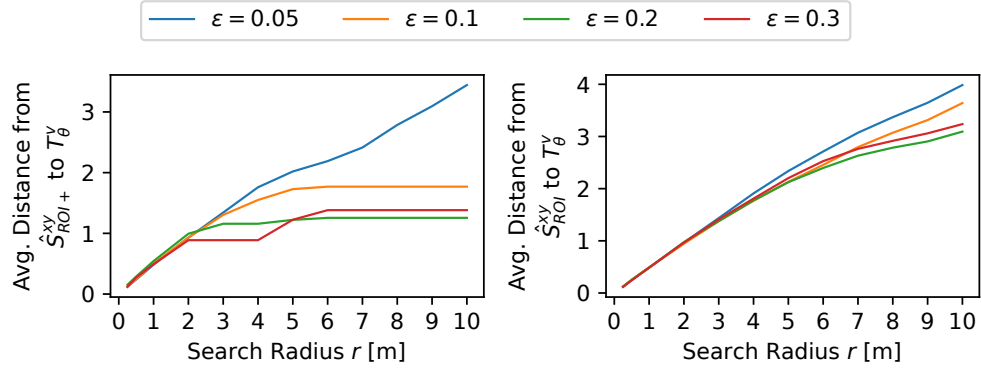


Figure 3.9: Shown are the three scenarios from the ROI validation experiment and several visualizations of how adding a single tree to each of these scenarios changes the regret based on the tree’s location. **Top:** The original scenario with the vehicle trajectories. **Bottom:** Shows a partial brute-force visualization of how the regret changes when a tree is added within the set $S_a^{xy}(0.5, 10, \theta)$. The hatched white region denotes the “unknown”, as this area was not tested. The gray areas represent an infeasible placement of the tree due to the feasibility constraints. In the colorbar, $\Delta R = \hat{R}(a(\theta, s)) - \hat{R}(\theta)$ indicates the regret difference between the child scenario $a(\theta, s)$ and the original/parent scenario θ . The region labeled $\Delta R \geq 0.1$ represents $\hat{S}_{ROI+}^{xy}$ with $\epsilon = 0.1$, and the two regions $\Delta R \geq 0.1$ and $\Delta R \leq -0.1$ together form $\hat{S}_{ROI}^{xy}$ with $\epsilon = 0.1$.

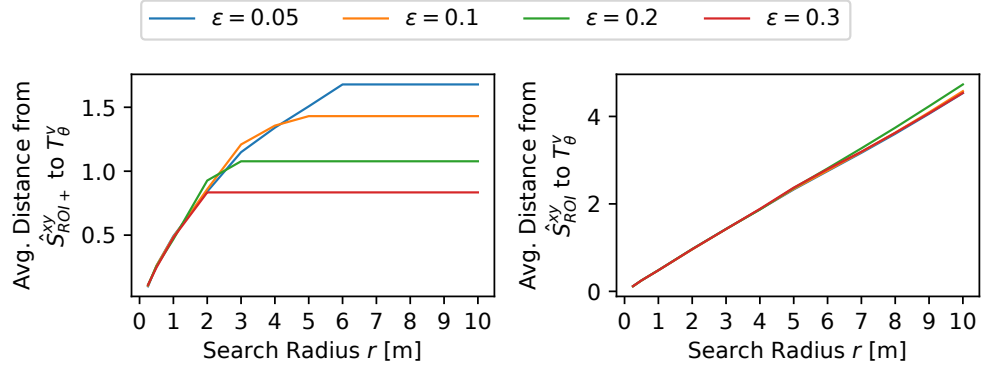
a small but positive search radius (though, the data within a search radius of <1 meters is limited due to the 0.5 meter discretization, thus, we cannot make any claims to the true behavior in this region without additional testing). While these trends are based on a very small sample size and on a discretized action space due to runtime cost, they are still considered representative and support the rationale behind the ROI mutation mechanism.



(a) Scenario 1.

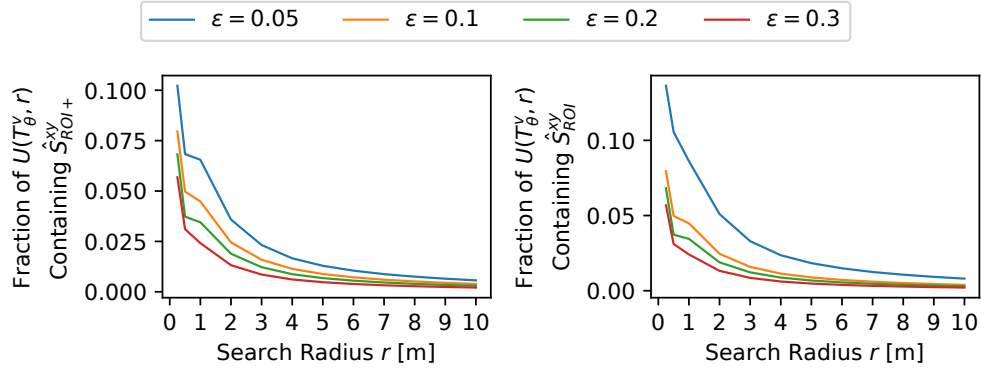


(b) Scenario 2.

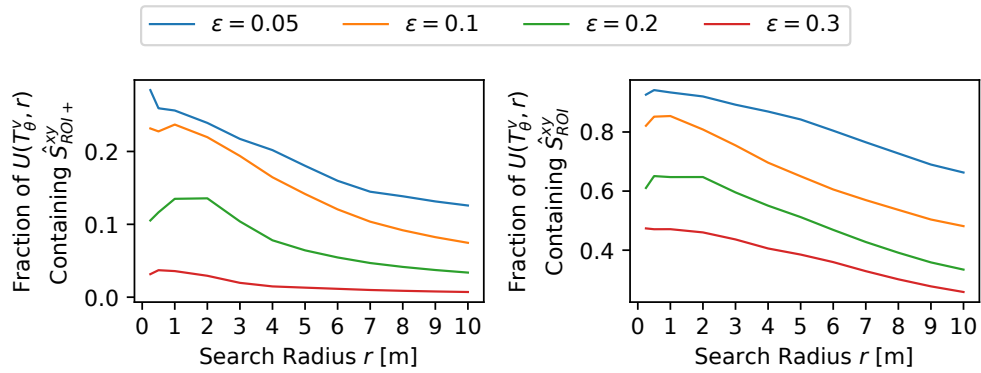


(c) Scenario 3.

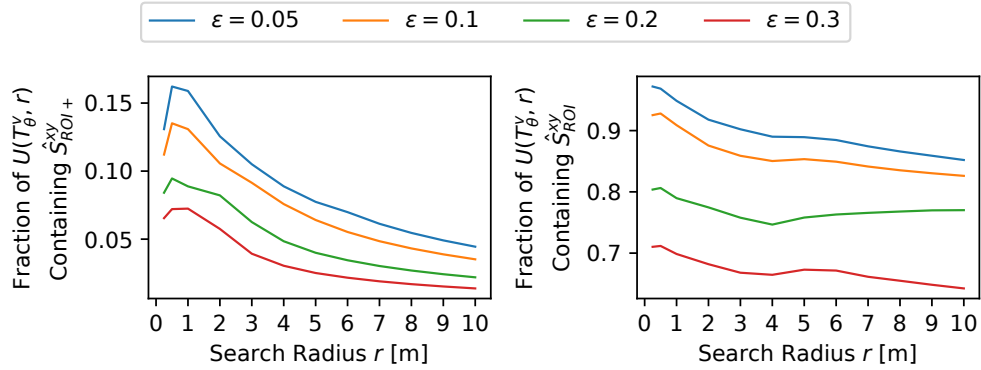
Figure 3.10: These plots show the average distance a point from $\hat{S}_{ROI+}^{xy}$ (left) and $\hat{S}_{ROI}^{xy}$ (right) is from the vehicle trajectories T_θ^v for the three scenarios in the ROI validation experiment. Distances are shown as a function of search radius for various regret thresholds of ϵ , which are used to construct the two ROI sets.



(a) Scenario 1.



(b) Scenario 2.

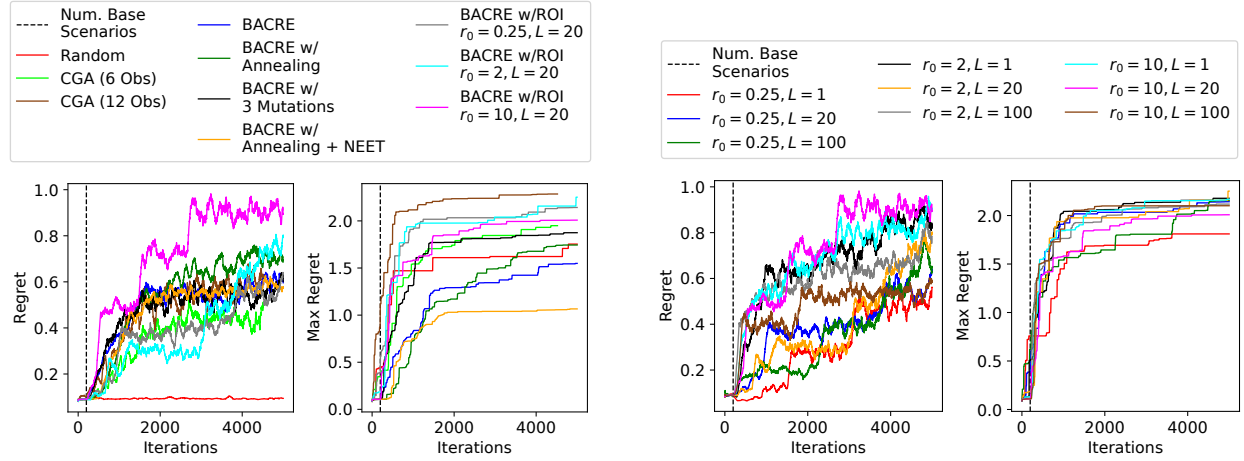


(c) Scenario 3.

Figure 3.11: These plots show the fraction of points within the *feasible* subset of the conservative ROI approximation $U(T_{\theta}^v, r)$ that belong to $\hat{S}_{ROI+}^{xy}$ (left) and $\hat{S}_{ROI}^{xy}$ (right) for the three scenarios in the ROI validation experiment. The feasible subset of $U(T_{\theta}^v, r)$ is shown by the green, beige, and black points combined in the bottom figures in Fig. 3.9, and the infeasible subset is shown in gray.

A few numerical experiments were then performed to demonstrate the efficiency boost of BACRE with ROI than without. When testing BACRE with ROI, the ROI mutation method was added on top of only the vanilla BACRE from Section 2.4 (the use of annealing or multiple mutations are *not* tested here). Multiple parameter combinations of BACRE with ROI were tested: All possible (r_0, L) pairs for an initial guess radius $r_0 \in \{0.25, 2, 10\}$ meters and learning period $L \in \{1, 20, 100\}$ iterations with a regret threshold $\epsilon = 0.3$. The root scenario was an “empty” scenario with a flat ground and no obstacles. Finally, all obstacles shared the same ROI estimator as opposed to each type having its own estimator.

Fig. 3.12 shows the regret learning curves from all methods tested in Section 2.5.4 (random search, a continuous genetic algorithm, and several BACRE variations) along with all of the versions of BACRE with ROI, and Fig. 3.13 shows how quickly each of these methods found a failure scenario for the test vehicle. The average regret in Fig. 3.12 (left) is not of concern since all methods (except random search) show an increasing trend; this data is only provided to be consistent with the figures from Section 2.5.4. For readability purposes, Fig. 3.12a shows how only a few of the BACRE with ROI variations compare to the methods from Section 2.5.4, and all of the BACRE with ROI variations are compared in Fig. 3.12b. The continuous genetic algorithm (CGA) with 12 obstacles performed the best as it was quickest to maximize regret, and accordingly, to find a failure scenario. However, it remains unreasonable to assume a fixed number of obstacles in a practical setting, and genetic algorithms do not scale well to high dimensions (thus, it would be a poor choice to use them to create 3D terrain). The BACRE with ROI variations were (generally) able to both maximize regret and find a failure scenario faster than all of the other methods and do so at similar rates, with a few exceptions. The variation with the smallest initial search radius and learning period ($r_0 = 0.25$ meters and $L = 1$) performed the worst out of all of the tested BACRE with ROI combinations regarding how much slower it both maximized regret and found a failure scenario (the next paragraph discusses a possible explanation). Additionally, random search was able to find a high-regret (and failure) scenario sooner than expected, as mentioned in Section 2.5.4. This “luck” from random search is likely attributed to the autonomy system not being so-



(a) All methods from Section 2.5.4 and a few versions of BACRE with ROI.

(b) All parameter combinations of BACRE with ROI.

Figure 3.12: Regret learning curves from Section 2.5.4 plus the versions of BACRE with ROI. Shown are the mean values over five trials. Smoothing is used in the left plots (regret vs. iterations) with a moving average window of size 100. The standard error is omitted to improve readability,

phisticated enough to break, or because collision is easier to induce with a higher obstacle density (as demonstrated by the CGA with 12 obstacles). The main takeaway is that BACRE with ROI was able to find a failure scenario within 250-500 search iterations, whereas the BACRE methods from Section 2.5.4 required 600-1200 iterations.

Fig. 3.14 and 3.15 provide a look under the hood of how the CSR evolves throughout the search process in BACRE with ROI. Fig. 3.14 shows how the CSR evolves for the nine parameter combinations tested. The average CSRs for $r_0 = 2$ meters and $r_0 = 10$ meters nearly approach the same value, while the CSRs for $r_0 = 0.25$ meters do not. Though, only the case with the smallest (r_0, L) combination resulted in a performance drop. Our best explanation for this is that when the initial ROI search radius is too small, the learning period needs to be sufficiently large so the ROI learning mechanism has time to identify a larger, more appropriate CSR. This theory is supported by Fig. 3.14 and Fig. 3.13; when $r_0 = 0.25$ meters and L is 20 or 100, the CSR converged to a higher radius, and the algorithm found a failure scenario more easily. The sampling mechanism when choosing the maximum search radius in line 8 of **CreateROIChildScenario** might also play a role here. These plots, along with Fig. 3.12 and Fig. 3.13, suggest that so long as r_0 and L are

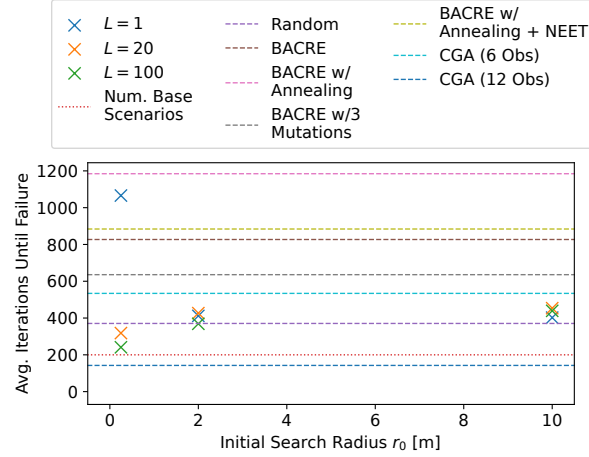


Figure 3.13: Shown are the average iterations needed for the methods in the comparison experiment to create a failure scenario. The x-axis only pertains to the BACRE with ROI variations, which are denoted by the colored X's, to help identify the (r_0, L) parameter combination. The methods tested in Section 2.5.4 are denoted with horizontal lines.

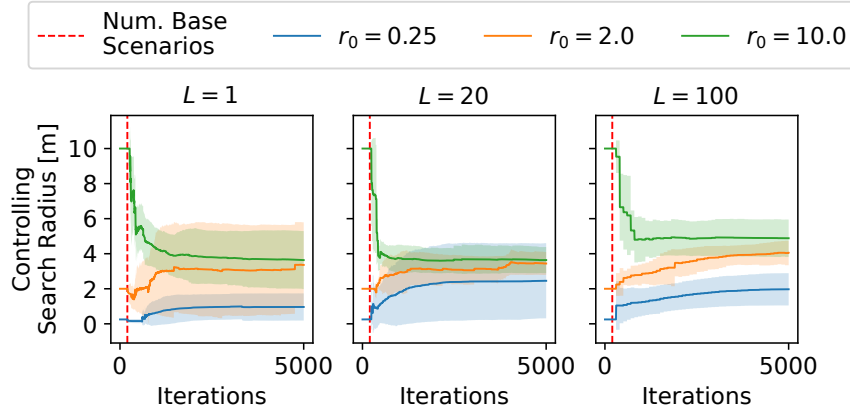
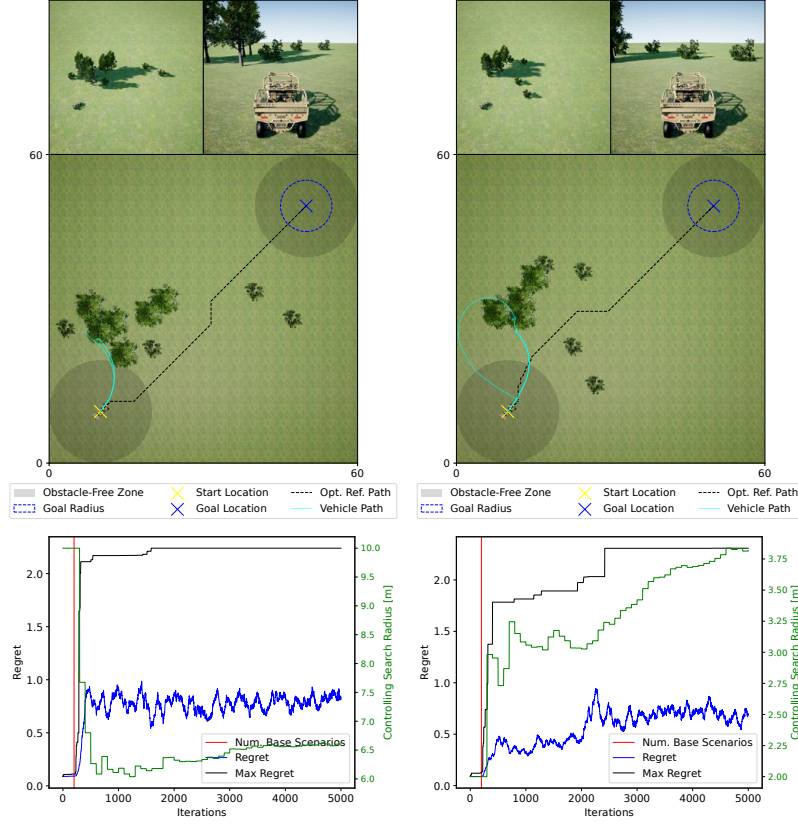


Figure 3.14: This plot shows how the controlling search radius changed over time for the nine different (r_0, L) combinations. The solid line represents the mean value over the five trials, with the shaded region depicting the standard error.

not *both* too small, then BACRE with ROI will likely work quite well.

The maximum regret scenario and corresponding regret learning curve from two BACRE with ROI variations are shown in Fig. 3.15. For both scenarios, the first rapid change in the CSR coincides with finding the first (relatively) high-regret scenario, as would be expected.

The experiments in this section demonstrate the validity of the ROI hypothesis and show that the addition of the ROI mutation mechanism considerably improves BACRE's search efficiency.



(a) $r_0 = 10$ meters and $L = 100$. (b) $r_0 = 2$ meters and $L = 100$.

Figure 3.15: Shown are the maximum regret scenario and corresponding regret learning curve from two of the BACRE with ROI variations that were tested. **Top:** Maximum regret scenario. **Bottom:** Corresponding regret learning curve. The (green) axis on the right of the plot shows how the controlling search radius (also in green) changed over time.

Despite the intractability of the true ROI, the proposed simple and light-weight mechanism offers significant improvement, not just over the vanilla BACRE, but also over annealing and applying multiple mutations.

3.4.2 BACRE-3D vs. TPP

These next two subsections showcase the ability of BACRE-3D² to find failure scenarios for two different state-of-the-art off-road trajectory planners and controllers [52, 77] for a Polaris MRZR navigating on rugged 3D terrain. We are grateful for the authors of these papers to share their

²Code and implementation details can be found at https://github.com/tsender/bacre_3D once the journal paper on this chapter is published (expected late 2025).

full code and simulation setup so we could perform these experiments. The criteria for what constitutes a failure scenario mostly follows that from Section 2.3.2, but the simulation setups provided by each paper use slightly different definitions for when certain events occur (e.g., the threshold angle for rollover). A scenario is deemed a failure if at least half of the vehicle attempts result in failure. We first aim to demonstrate that BACRE with ROI (with the ability to sculpt the terrain using multiple instances of the tools mentioned in Section 3.3.1) is insufficient at finding failure scenarios and that it needs the other proposed improvements (to avoid local maxima and to augment the regret with risk of failure metrics). The actual landscape resolution is set to 5 cm, but the resolution given to the autonomy system is coarser than this to represent environment uncertainty. Perception systems were not utilized with these autonomy systems, but could be used in future work. All experiments consisted of running five independent trials.

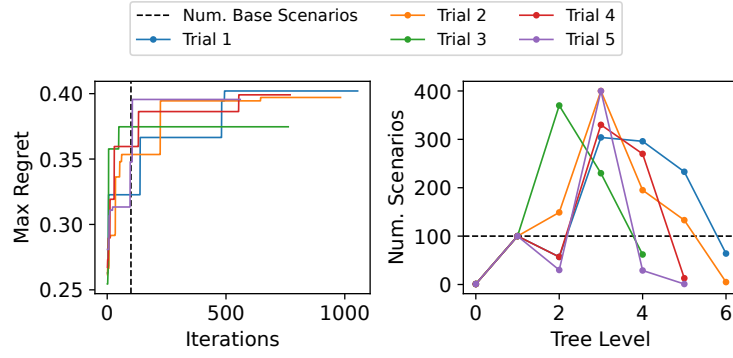
The base population has size $N_0 = 100$, all scenarios are ran for $n_{\text{trial}} = 5$ trials, and scenarios are created in batches with a batch size of 10 for parallel simulations. The ROI parameters used were $r_0 \in \{2, 5\}$ meters (depending on the sculpt tool parameters), $\epsilon = 0.3$, and $L = 20$ iterations. When using BACRE-3D, the population reset parameters were $N_{\text{max}} = 1000$, $N_{\text{idle}} = 400$, $N_{\text{max}}^{\text{lev1}} = 400$, and $N_{\text{max}}^{\text{lev2}} = 600$. Both versions of BACRE use the same sculpt tools and select a tool uniformly randomly from the tool set. Eight tools in total are created (three circular, one rectangular, and four red noise), each tool specifies a range of values allowed for each parameter, and whenever a tool is selected, a new set of parameter values is chosen randomly. Both versions of BACRE also use the same A* reference agent that finds the shortest 3D path tangential to the terrain from the start to the goal (using a 0.25-meter resolution of the terrain). The A* agent has constraints to ensure each path edge has an elevation angle and tangential roll angle of at most 20 degrees and is at least 2 meters from the map boundary.

The trajectory planner/controller (i.e., the autonomy system) tested in this subsection is the Three-Phase Planner (TPP) developed in [52]. The simulation setup consists of precomputing the global trajectory via TPP in Ubuntu, and then running a simulation with Project Chrono [40] (version 9.0 running on Windows 10) in which a ROS node [78] (Noetic running on Ubuntu) sends

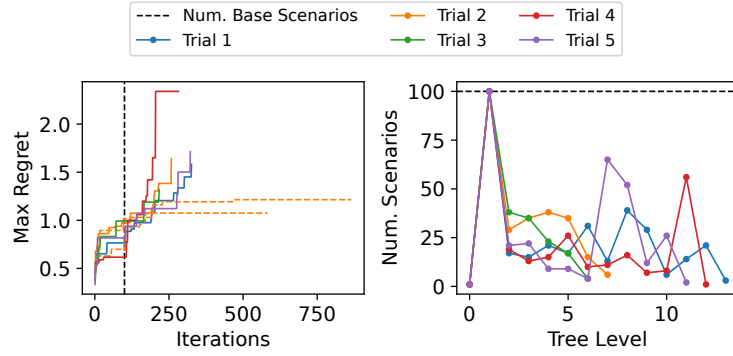
control commands to the simulator to track the reference trajectory. Numerous improvements were made to the original setup to run parallel simulations. We used the same model and simulation parameters as in [52] with a few exceptions. Since the terrain in our experiments is rectangular, instead of square, we manually adjusted TPP’s A* planner to use a spacing of 0.5 meters, which is approximately what it translated to in the paper. We also configured Chrono to use deformable terrain, instead of rigid terrain as used in [52], with the Bekker-Wong parameters for sandy loam $n = 1$, $k_c = 5.7 \text{ kN/m}^2$, and $k_\phi = 2293 \text{ kN/m}^3$ taken from [79].

The TPP vehicle’s top speed was set to 9 m/s and it was provided with a 0.5-meter resolution of the environment for planning because traveling at high speeds with imperfect knowledge is expected to be easiest to induce failure. When BACRE with ROI was pitted against the TPP vehicle, it failed to find any failure scenarios, and as expected, the tree levels within the population saturated. Fig. 3.16a shows a breakdown of the population tree levels and the evolution of the maximum regret attained by BACRE with ROI. Due to limited compute resources, various criteria was used for early termination of the experiments when it was clear that BACRE was stuck in a local maxima. The maximum regret scenarios from two different trials are shown in Fig. 3.17. With a batch size of 10, each experiment trial took about 15-20 hours.

The next step was to pit BACRE-3D (with all features from Section 3.3) against the same TPP vehicle. Unfortunately, this resulted in an unexpected problem. In preliminary testing, we encountered a bug with Chrono in which one of the vehicle’s front tires rotated well beyond its mechanical limit. The Chrono team suggested using deformable terrain instead of rigid terrain (we initially used rigid terrain following the experiments in [52]). We followed their recommendations, and for additional safety, we added a feasibility check so that if this issue occurred again, BACRE would reject the scenario and replace it with another. In the experiment of BACRE with ROI vs. TPP, only a handful of scenarios were rejected. However, once we began using BACRE-3D, at some point, most of the scenarios created resulted in this same problem. The frequency of the scenario rejection halted BACRE-3D from continuing in its search process. Due to limited compute resources, we chose to disable the feasibility check with BACRE-3D.



(a) BACRE with ROI vs. TPP.



(b) BACRE-3D vs. TPP.

Figure 3.16: Summary plots from the experiments with TPP. **Left:** The evolution of the maximum regret attained vs. the number of search iterations. **Right:** A breakdown of how many scenarios were in each tree level in the latest population. (b) Data from previous populations are shown in dashed lines, but these are only shown in the left plot.

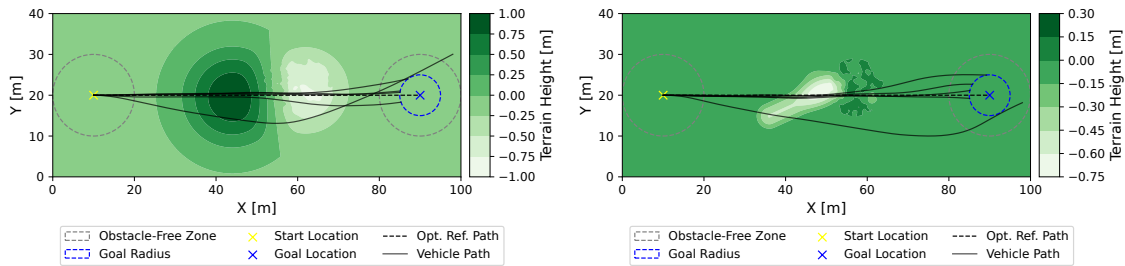


Figure 3.17: The maximum regret scenarios from two of the BACRE with ROI vs. TPP trials with the vehicle and reference trajectories overlaid.

In four of the five trials of BACRE-3D vs. TPP, the failure scenarios found were solely due to a lack of steering control caused by the tire issue discussed above (the fifth failure was due to rollover, but the tire issue presented just before the vehicle rolled over). Fig. 3.18 shows three of

the failure scenarios caused by this tire issue and the actual tire data at the relevant point in time. The tire data in the top row is most telling: In all three cases shown (and in the other two trials not shown), when driving over a rough region, the right wheel experiences zero tire force while a mysterious non-zero spindle Z moment causes the wheel to rotate beyond its mechanical limit. One of the weird cases is from trial 5 (Fig. 3.18c) in which the left tire is rotated to its maximum angle while the right tire rotates in the opposite direction beyond its limit. Fig. 3.19 shows two snapshots from trial 2 showing this weird behavior with the right tire.

Fig. 3.16b shows a breakdown of the population tree levels and the evolution of the maximum regret attained by BACRE-3D for the five trials. When compared to BACRE with ROI's performance, BACRE-3D was (generally) able to find a failure scenario in less than 350 iterations (with the exception of trial 2 which required two population resets), whereas BACRE with ROI was unable to find any failure scenarios in 500-1000 iterations.

We have an outstanding ticket³ detailing the tire issue with the Project Chrono team. At this point, we could not continue further experimentation with BACRE-3D and TPP. While the initial goal was to demonstrate that BACRE-3D can find scenarios that push an off-road AV to its limits, we instead showed that BACRE-3D is more generally a tool that can push an entire simulation platform (which includes the physics engine, vehicle model, and virtual autonomy system) to its limit. In this case, BACRE-3D found a flaw in a popular open-source simulation engine.

A final remark about BACRE-3D is warranted. Since BACRE is a black-box algorithm, it cannot indicate *why* a particular scenario induces vehicle failure. Manual analysis is required to determine the underlying source of the problem and subsequently the fix. Identifying the problem that Chrono is unable to properly model tire collisions in not-so-challenging terrain took a considerable amount of work. Numerous data logs had to be recorded and inspected before identifying the root of the problem. While BACRE-3D appears to be quite capable, this limitation must be considered when using the algorithm.

³[Click for details.](#)

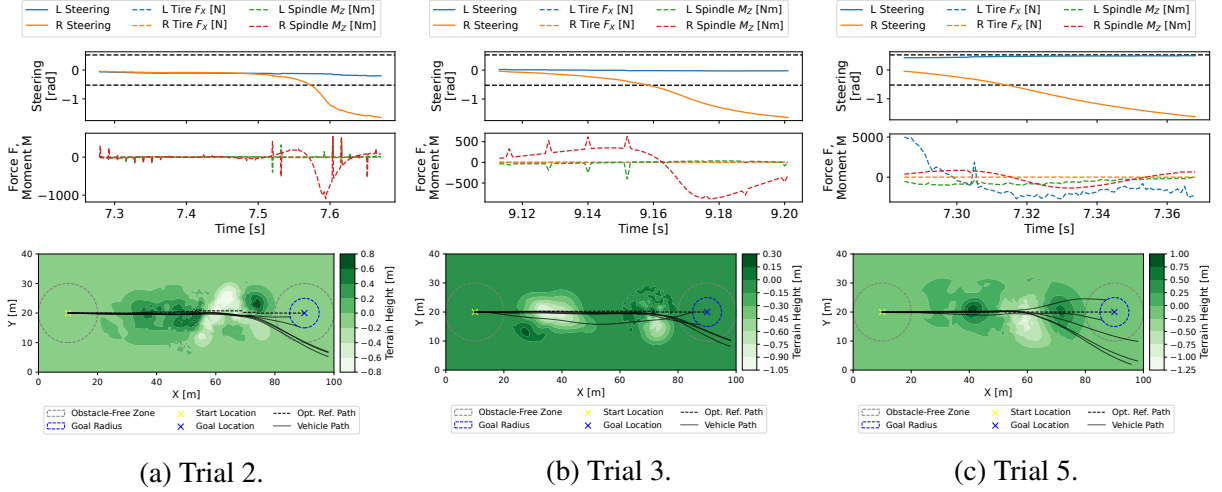


Figure 3.18: Shown are the failure scenarios and the corresponding tire steering/force/moment data from three of the BACRE-3D vs. TPP trials. **Top:** Various data about the tires are shown as the front right tire rotates from 2% to 75% of its final angle. The top plot shows the steering angles, with the horizontal dashed lines marking the steering range, and the bottom plot shows the tire force F_x in the local x-axis and the moment M_z about the spindle’s local z-axis. In the legend, “L” and “R” denote the left and right tires, respectively. **Bottom:** The failure scenario with the vehicle and reference trajectories overlaid.

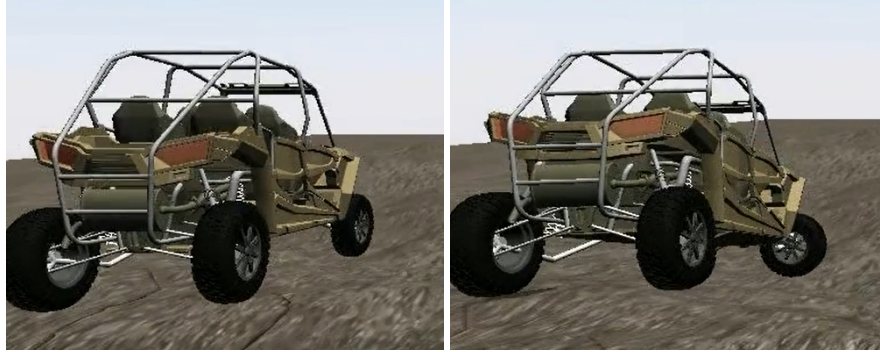


Figure 3.19: Renderings at two snapshots of the vehicle’s right side from trial 2 in the BACRE-3D vs. TPP experiment. These snapshots occur around the time 7.6 seconds in the tire plot in Fig. 3.18a, and the front right tire has clearly rotated about 90 degrees at time t_2 .

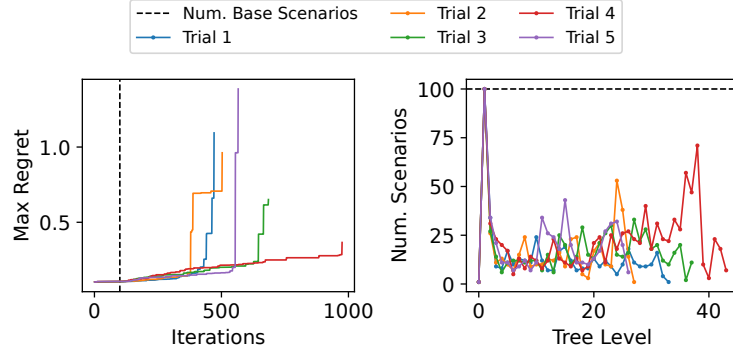
3.4.3 BACRE-3D vs. MPC

The algorithm tested in this subsection is a state-of-the-art local trajectory planner/controller for AGVs driving on rugged 3D terrain [77]. The planner uses a nonlinear model predictive controller

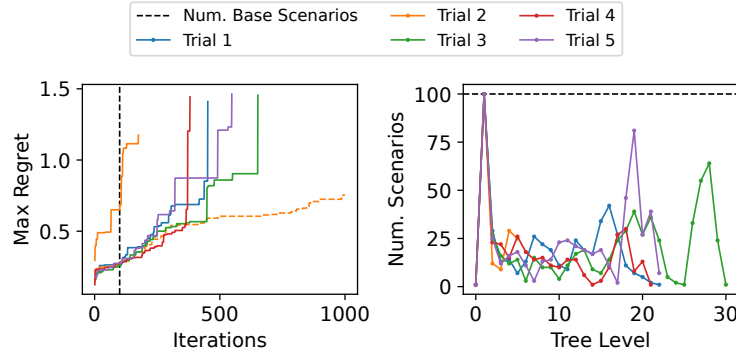
and a 6.5-degree-of-freedom vehicle dynamics model that can predict tire lift-off and rollover. This planner will be referred to as the all-terrain model predictive controller (ATMPC) in this dissertation. The exact same simulation setup as in [77] is used (with Vortex Studio [41] as the simulator running on Ubuntu) but with different hardware. Because ATMPC is a local planner and does not contain a global planner, the environment is made narrower than what was used in the experiments with TPP so that any failure scenarios are more likely to result from the planner's vehicle handling capability than from the lack of a global planner. In these experiments, the vehicle was set to travel at a speed of 5 m/s and the planner was given a 0.2-meter resolution of the actual terrain. With a batch size of 10, each experiment trial took about 5-15 hours each.

The first experiment involved pitting BACRE with ROI against ATMPC. BACRE with ROI was not expected to be able to find any failure scenarios as it was hypothesized that it would get stuck in a local maxima like it did with the TPP vehicle. Remarkably, it managed to find a failure scenario in each trial. Fig. 3.20a shows a breakdown of the population tree levels and the evolution of the maximum regret attained by BACRE with ROI. Most trials only required around 450-700 iterations to find a failure scenario, with one requiring just under 1000. These failure scenarios consisted of 26-40 mutations applied from the root scenario. Three of the failure scenarios are shown in Fig. 3.21. Of the five trials, four failure scenarios were due to the vehicle getting stuck on steep terrain and one was due to rollover. Considering that BACRE with ROI did not use the risk of failure metrics, we did not expect it to create such rugged terrain or even find any failure scenarios. However, the magnitude of the vehicle's acceleration (middle row of Fig. 3.21) and the renderings (bottom row) shows that BACRE with ROI exceeded our expectations and was capable of generating rugged, adversarial terrains for ATMPC.

The next experiment involved pitting BACRE-3D against ATMPC. BACRE-3D performed as expected and was able to find a failure scenario in each trial. Fig. 3.20b shows a breakdown of the population tree levels and the evolution of the maximum regret attained by BACRE-3D. Like with BACRE with ROI, most trials only required around 450-700 iterations to find a failure scenario, with one requiring just over 1000 due to a population reset. Most of the failure scenarios consisted



(a) BACRE with ROI vs. ATMPC.



(b) BACRE-3D vs. ATMPC.

Figure 3.20: Summary plots from the experiments with ATMPC. **Left:** The evolution of the maximum regret attained vs. the number of search iterations. **Right:** A breakdown of how many scenarios were in each tree level in the latest population. **(b)** Data from previous populations are shown in dashed lines, but these are only shown in the left plot.

of 20-31 mutations, with one requiring 44 mutations (this was from trial 2 with the population reset). All five of the failure scenarios were due to rollover. Three of the five failure scenarios are summarized in Fig. 3.22. The terrains created by BACRE-3D were just as rugged as those created by BACRE with ROI. The inclusion of the risk of failure metrics clearly influenced BACRE-3D to create terrains that induced rollover, instead of terrain that had steep slopes. The maximum regret consequently increased more steadily due to these additional terms, compared to when these risk metrics were omitted (see Fig. 3.20).

In both sets of experiments, most of the failure scenarios resulted from the vehicle driving through rugged/intense regions when less rugged/intense regions exist. The reason for ATMPC's

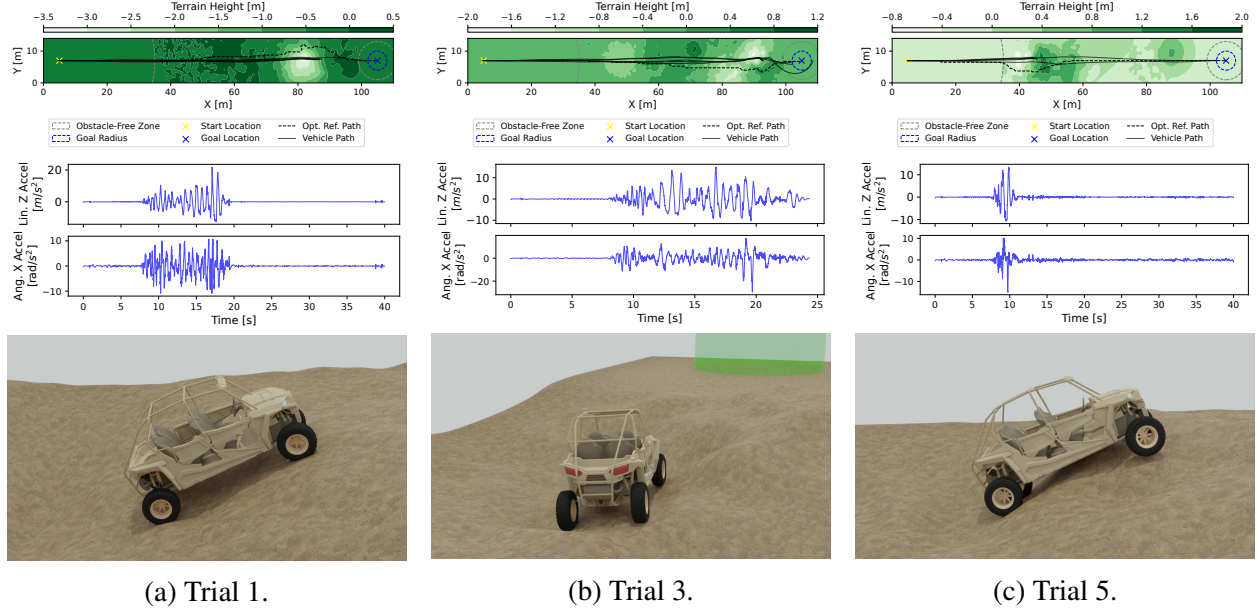


Figure 3.21: Shown are three failure scenarios from the BACRE with ROI vs. ATMPAC experiment. **Top:** A summary of the failure scenario showing the terrain height with the vehicle and reference trajectories overlaid. **Middle:** The linear acceleration about the vehicle’s z-axis and angular acceleration about the vehicle’s x-axis from one of the five vehicle runs. The magnitude shows how rugged the terrains are. **Bottom:** A rendering of the vehicle in the area that was most challenging for it to navigate. The vehicle is stuck on the slopes in the trial 1 and trial 3 renderings.

“aggressive” behavior is likely due to the lack of any incentive to take safer routes in its optimal control problem. Some of its failures are likely due to assuming the vehicle’s speed remains constant and it has enough momentum (even after slowing down following several impacts with the ground) and enough traction to perform various maneuvers. Moreover, the cases where the vehicle chose to drive along the steepest inclines up the hill (instead of shallower ones) could also be due to the lack of a global planner and/or too short of a prediction time horizon. While the A* reference planner is not perfect (as it does not perform local planning), it does avoid the steepest regions that the ATMPAC vehicle chose to drive through. As seen in the top rows of Fig. 3.21 and Fig. 3.22, the vehicle paths do not take the same route as the reference agent. A more dynamically feasible reference agent could of course be used (at the price of added runtime cost), but these experiments suggest that A* is sufficient.

Even though BACRE with ROI performed rather well against the ATMPAC vehicle, the im-

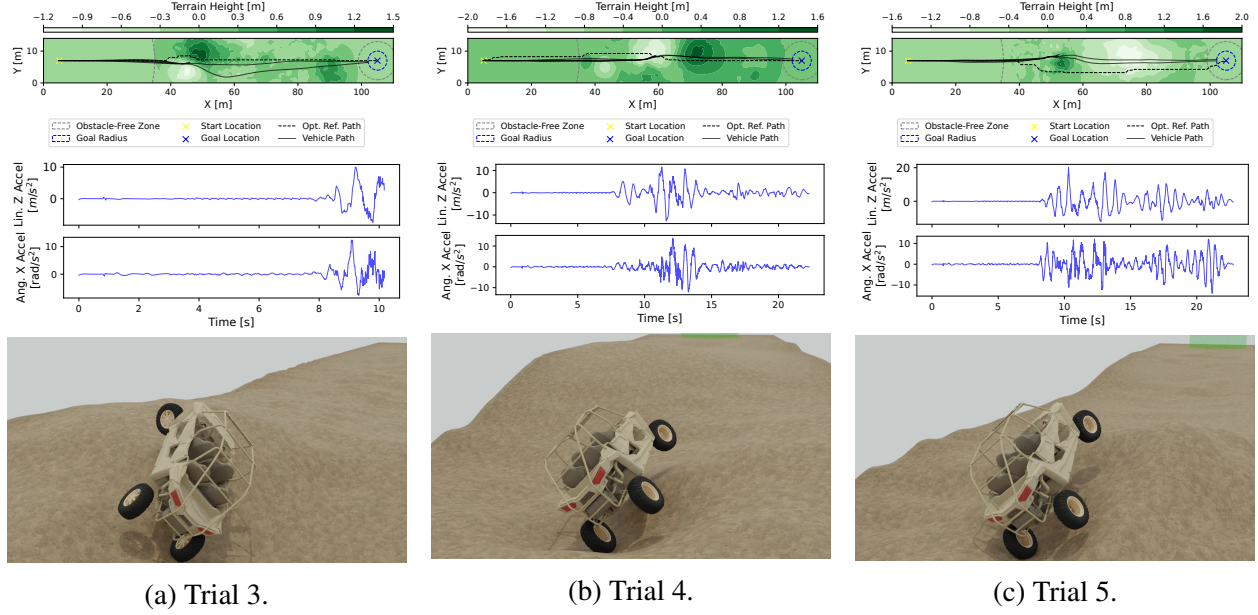


Figure 3.22: Shown are three failure scenarios from the BACRE-3D vs. ATMPC experiment. **Top:** A summary of the failure scenario showing the terrain height with the vehicle and reference trajectories overlaid. **Middle:** The linear acceleration about the vehicle’s z-axis and angular acceleration about the vehicle’s x-axis from one of the five vehicle runs. The magnitude shows how rugged the terrains are. **Bottom:** A rendering of the vehicle in the area that was most challenging for it to navigate. The vehicle is rolling over in all of the renderings shown.

provements to avoid local maxima are strongly recommended. The ability of BACRE to find failure scenarios without the risk of failure metrics for this test system is rather interesting. These performance differences of BACRE between the TPP and ATMPC vehicles suggest that some trial and error is needed to determine if the original regret metric (2.8) is sufficient or if it needs to be augmented with other terms, such as the ones proposed in Section 3.3.2. Overall, these experiments demonstrate that BACRE-3D is well-suited for identifying adversarial scenarios for sophisticated off-road AVs.

3.5 Conclusions

This chapter presented an algorithm called BACRE-3D, a novel extension of its predecessor BACRE, as an effective method for generating adversarial scenarios with 3D terrains for black-box off-road

autonomy systems. BACRE-3D was created by addressing two algorithmic and two implementation limitations of BACRE. A novel region of influence (ROI) mutation method was proposed for improving search efficiency, and a mechanism for adaptively improving the population’s diversity was proposed to keep BACRE from getting stuck in local maxima. To apply BACRE-3D in the off-road context, several elementary terrain sculpting tools were proposed for mutating a 3D terrain, and an augmented regret metric was proposed by incorporating various risk of failure metrics to enrich the regret signal.

Numerous experiments were conducted to demonstrate the improved performance of BACRE-3D over BACRE. The first set of experiments showed that BACRE with ROI was able to find failure scenarios much faster than the previous versions of BACRE from Section 2.4. The second set of experiments showed that BACRE-3D was able to find a flaw in Project Chrono, a popular open-source multi-physics simulation engine, as well as a state-of-the-art off-road local trajectory planner/controller. BACRE-3D has demonstrated that, without any knowledge of how the test systems work, it can find adversarial scenarios that highlight a performance limitation of the autonomy system, or of the simulator, using only a series of sequential elementary mutations. While BACRE is the first adversarial algorithm intended to target off-road autonomy systems, BACRE-3D has proven to truly be the first adversarial method for finding realistic and unstructured adversarial scenarios for off-road AVs.

There are numerous avenues for future work that could improve BACRE-3D’s performance and applicability to the off-road domain. The ROI mutation mechanism was shown to considerably improve BACRE’s search efficiency, however, one of its limitations is in the process of choosing where along the vehicle trajectory is the best place to center the search for applying a mutation. Strategizing where along the vehicle trajectory to focus on could further improve the ROI mutation mechanism. Since the BACRE-3D experiments only used autonomy systems with partial terrain knowledge, future studies should consider applying BACRE-3D to more “complete” autonomy systems, such as those with sensors and a perception system. It is unclear if future improvements would be necessary for BACRE-3D to be effective against those systems. Another area for future

work would be to extend BACRE-3D to the case of finding failure scenarios for *multiple* intelligent agents operating in off-road environments (e.g., multiple AVs or even a human-autonomy team). Though, significant improvements to the search efficiency may be necessary before BACRE-3D is worth testing with human subjects.

CHAPTER 4

Conclusions, Summary of Contributions, and Future Work

4.1 Conclusions

This dissertation discusses and addresses the important challenge of finding scenarios that highlight weaknesses and limitations in off-road autonomy systems. BACRE-3D was developed as an efficient and powerful tool for addressing this task. By taking a black-box approach, BACRE-3D can find limitations of a wide variety of off-road autonomy systems so long as the vehicle and autonomy system models are fixed. The primary requirements for using this tool are having a simulation platform with sufficient modeling capabilities (verifying this criteria is beyond the scope of this dissertation) and the ability to compute a reasonable reference trajectory to compare the test vehicle’s trajectory against. The one caveat to taking a black-box approach is that we cannot determine the underlying cause of the vehicle failure without extensive manual analysis. However, as a result of the evolutionary nature of the algorithm, BACRE-3D can provide the sequence of mutations that show the degradation of the vehicle’s performance. Extensive numerical experiments showed that BACRE-3D is poised to efficiently find adversarial scenarios that point out limitations in off-road autonomous vehicles as well as the simulation platform. These scenarios would otherwise be challenging to design considering the complex interactions within the system as a whole, and they would be very difficult for a random search method to generate without infinite compute resources. Even if experts had a deep understanding of the system’s failure mechanisms, it would be very time-consuming to manually design failure scenarios. Allowing an algorithm to expose the components that are easiest to break is one of the most effective testing tools that we have.

4.2 Summary of Contributions

This section summarizes the key contributions provided in this dissertation:

- On a general note, this dissertation shows that the problem of finding static, unstructured scenarios that are adversarial towards off-road autonomy systems (by highlighting their weaknesses and limitations) can be formulated as an evolutionary problem in which the mutation process consists of applying high-level perturbations to the scenario definition (e.g., rearranging obstacles and sculpting the terrain locally).
- BACRE was developed as an evolutionary algorithm for finding adversarial scenarios for off-road AVs. While BACRE was created as a stepping-stone for BACRE-3D, it does present a few novel contributions: (1) the inclusion of a reference agent to compare the test vehicle’s behavior to (this does not appear in the on-road literature to the best of the author’s knowledge), and (2) a novel black-box regret metric.
- BACRE-3D was developed as a novel extension of BACRE that is capable of generating 3-dimensional static environments that are adversarial towards off-road autonomy systems. The two algorithmic improvements to BACRE (the ROI mutation mechanism and the population reset mechanism) are novel in their own right and were designed specifically for this application. The regret ROI mutation mechanism is a novel improvement to BACRE’s search process that consists of: (1) learning an online distance from the vehicle trajectories where mutations have the most influence on the regret, and (2) only applying mutations to the environment within those regions (which drastically reduces the search space). The population reset mechanism is a novel improvement to BACRE’s overall optimization process to prevent the algorithm from remaining stuck in a local maxima. The mechanism works by adaptively creating a new generation with higher complexity than the previous one to meet the level of complexity needed to make the test system fail. An extensive set of numerical experiments was performed to validate the improved performance of BACRE-3D and they

also showed that BACRE-3D was able to identify limitations in a state-of-the-art simulator and an off-road trajectory planner.

- A custom, open-source simulation platform called AutomaticSceneGeneration was created based on Unreal Engine 4 and ROS2 to assist in the automated creation of diverse off-road scenarios and in the oversight of navigation tasks within those scenarios. While the current version of this platform only supports the creation of scenarios with obstacles on flat ground, the extensive automation features it contains serve as an example for what the off-road AV testing community deserves.

4.3 Future Research Directions

BACRE-3D is the first known adversarial method proposed for finding scenarios that highlight an off-road autonomy system's performance limitations. In lieu of the algorithm's simplicity and capability, it is expected that it will play an important role in the off-road AV testing community. However, the algorithm does suffer from several limitations, and there are other areas of interest for future work.

- While finding failure scenarios for a system is important, doing so efficiently is always a high priority whenever possible. The ROI mutation mechanism is a novel addition to BACRE and it was shown to considerably reduce the number of search iterations needed. An interesting research direction would be to explore extensions of this mechanism or investigate if other enhancements could be made to further improve upon BACRE-3D's efficiency.
- This dissertation only focused on creating *static* unstructured environments. BACRE-3D's capabilities would be strengthened if it were capable of controlling scenario attributes within *dynamic* environments, too.
- Characterizing failure scenarios probabilistically is yet another important avenue for future exploration. Understanding the probability that the vehicle could encounter various types of

failure scenarios is useful for quantifying how well the system performs overall and for prioritizing which failures to address.

- BACRE-3D considers the system under test as a black-box to broaden the range of systems it can work with. While the original intention of BACRE was only to find failure scenarios, exploring how to expand BACRE-3D's abilities to either identify, or assist in the identification of, the underlying cause(s) of the vehicle failures would be of significant benefit to the community. The only assistance that BACRE currently offers on this matter is to provide the sequence of mutations that resulted in the failure scenario. While this is valuable information, a significant amount of manual effort is required to individually inspect these scenarios.
- BACRE-3D requires that the test system is fixed. This precludes the ability to test systems with active learning components that can learn from their mistakes (which is undoubtedly an important ability that intelligent robotic systems should have). Exploring how to modify BACRE-3D's framework to properly test these systems is another important direction for future work.

BIBLIOGRAPHY

- [1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.
- [2] Luca Bascetta, Davide Antonio Cucci, Gianantonio Magnani, Matteo Matteucci, Dinko Osmankovic, and Adnan Tahirovic. Towards the implementation of a mpc-based planner on an autonomous all-terrain vehicle. In *Workshop on Robot Motion Planning: Online, Reactive, and in Real-time*, pages 1–7. International Conference on Intelligent Robotics and Systems, 2012.
- [3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [4] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.
- [5] Sorin Mihai Grigorescu, Bogdan Trasnea, Tiberiu T. Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37:362 – 386, 2019.
- [6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.
- [7] Weiwen Deng, Yong H. Lee, and Annie Zhao. Hardware-in-the-loop simulation for autonomous driving. In *2008 34th Annual Conference of IEEE Industrial Electronics*, pages 1742–1747, 2008.
- [8] Tamás Tettamanti, Mátyás Szalai, Sándor Vass, and Viktor Tihanyi. Vehicle-in-the-loop test environment for autonomous driving with microscopic traffic simulation. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2018.
- [9] Taeyoung Oh, Yunchul Ha, Dayeon Yoo, and Jinwoo Yoo. Ad-vils: Implementation and reliability validation of vehicle-in-the-loop simulation platform for evaluating autonomous driving systems. *IEEE Access*, 12:164190–164209, 2024.
- [10] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

- [11] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2633–2642, 2020.
- [12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [13] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.
- [14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [15] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.
- [16] F. Hendriks, M. Tideman, R. Pelders, R. Bours, and X. Liu. Development tools for active safety systems: Prescan and vehil. In *Proceedings of 2010 IEEE International Conference on Vehicular Electronics and Safety*, pages 54–58, 2010.
- [17] Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.
- [18] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. *2022 International Conference on Robotics and Automation (ICRA)*, pages 2419–2426, 2021.
- [19] Wenhao Ding, Chejian Xu, Mansur Arief, Hao ming Lin, Bo Li, and Ding Zhao. A survey on safety-critical driving scenario generation—a methodological perspective. *IEEE Transactions on Intelligent Transportation Systems*, 24:6971–6988, 2022.
- [20] Moritz Klischat and Matthias Althoff. Generating critical test scenarios for automated vehicles with evolutionary algorithms. *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2352–2358, 2019.
- [21] Xiaokun Zheng, Huawei Liang, Biao Yu, Bichun Li, Shaobo Wang, and Zhilei Chen. Rapid generation of challenging simulation scenarios for autonomous vehicles based on adversarial test. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1166–1172, 2020.

- [22] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277, 2019.
- [23] Akifumi Wachi. Failure-scenario maker for rule-based agent using multi-agent adversarial reinforcement learning and its application to autonomous driving. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6006–6012. International Joint Conferences on Artificial Intelligence Organization, 7 2019.
- [24] Wenhao Ding, Minjun Xu, and Ding Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.
- [25] Wenhao Ding, Baiming Chen, Bo Li, Kim Ji Eun, and Ding Zhao. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. *IEEE Robotics and Automation Letters*, 6:1551–1558, 2020.
- [26] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J. Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7, 2018.
- [27] Mark Koren and Mykel J. Kochenderfer. Efficient autonomy validation in simulation with adaptive stress testing. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4178–4183, 2019.
- [28] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1555–1562, 2018.
- [29] Cumhur Erkan Tuncali, Georgios Fainekos, Danil Prokhorov, Hisahiro Ito, and James Kapinski. Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles*, 5(2):265–280, 2020.
- [30] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9904–9913, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society.
- [31] Justin Norden, Matthew O’Kelly, and Aman Sinha. Efficient black-box assessment of autonomous vehicle safety. *CoRR*, abs/1912.03618, 2019.
- [32] Matthew O’Kelly, Aman Sinha, Hongseok Namkoong, John Duchi, and Russ Tedrake. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18*, page 9849–9860, 2018.
- [33] Reuven Y. Rubinstein and Dirk P. Kroese. The cross-entropy method: A unified approach to combinatorial optimization, monte-carlo simulation and machine learning. 2004.

- [34] Søren Asmussen and Peter W. Glynn. Stochastic simulation - algorithms and analysis. In *Stochastic modeling and applied probability*, 2007.
- [35] Zdravko I. Botev and Dirk P. Kroese. An efficient algorithm for rare-event probability estimation, combinatorial optimization, and counting. *Methodology and Computing in Applied Probability*, 10:471–505, 2008.
- [36] Mansur Arief, Zhiyuan Huang, Guru Koushik Senthil Kumar, Yuanlu Bai, Shengyi He, Wenhao Ding, Henry Lam, and Ding Zhao. Deep probabilistic accelerated evaluation: A robust certifiable rare-event simulation methodology for black-box safety-critical systems. In *International Conference on Artificial Intelligence and Statistics*, 2021.
- [37] Mansur Arief, Zhepeng Cen, Zhenyuan Liu, Zhiyuan Huang, Bo Li, Henry Lam, and Ding Zhao. Certifiable evaluation for autonomous vehicle perception systems using deep importance sampling (deep is). In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1736–1742, 2022.
- [38] Maggie Wigness, Sungmin Eum, John G Rogers, David Han, and Heesung Kwon. A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [39] Peng Jiang, Philip Osteen, Maggie Wigness, and Srikanth Saripalli. Rellis-3d dataset: Data, benchmarks and analysis, 2020.
- [40] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. pages 19–49. Springer, 2016.
- [41] CM Labs. Vortex studio, 2016. <https://www.cm-labs.com/en/vortex-studio/>.
- [42] Isaac Han, Dong-Hyeok Park, and Kyung-Joong Kim. A new open-source off-road environment for benchmark generalization of autonomous driving. *IEEE Access*, 9:136071–136082, 2021.
- [43] Samuel Misko, Arnold Free, Shiva Sivashankar, Torsten Kluge, Vladimir Vantsevich, and et al. Real-time, closed-loop and physics-based modeling and simulation system for unmanned ground vehicles in unstructured terrain environments,. *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, pages 1–19, 2022.
- [44] Tong Xu, Chenhui Pan, Madhan B. Rao, Aniket Datar, Anuj Pokhrel, Yuanjie Lu, and Xuesu Xiao. Verti-bench: A general and scalable off-road mobility benchmark for vertically challenging terrain. *ArXiv*, abs/2502.11426, 2025.
- [45] Adnan Tahirovic and Gianantonio Magnani. Passivity-based model predictive control for mobile robot navigation planning in rough terrains. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 307–312, 2010.

- [46] James Dallas, Michael P. Cole, Paramsothy Jayakumar, and Tulga Ersal. Terrain adaptive trajectory planning and tracking on deformable terrains. *IEEE Transactions on Vehicular Technology*, 70(11):11255–11268, 2021.
- [47] Siyuan Yu, Congkai Shen, and Tulga Ersal. Nonlinear model predictive planning and control for high-speed autonomous vehicles on 3d terrains. *IFAC-PapersOnLine*, 54(20):412–417, 2021. Modeling, Estimation and Control Conference MECC 2021.
- [48] Congkai Shen, Siyuan Yu, and Tulga Ersal. A three-phase framework for global path planning for nonholonomic autonomous vehicles on 3d terrains. *IFAC-PapersOnLine*, 54(20):160–165, 2021. Modeling, Estimation and Control Conference MECC 2021.
- [49] Shingo Shimoda, Yoji Kuroda, and Karl Iagnemma. High-speed navigation of unmanned ground vehicles on uneven terrain using potential fields. *Robotica*, 25(4):409–424, 2007.
- [50] Karl Iagnemma, Shingo Shimoda, and Zvi Shiller. Near-optimal navigation of high speed mobile robots on uneven terrain. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4098–4103, 2008.
- [51] Andrew Eick and David Bevly. Nonlinear model predictive controller for an unmanned ground vehicle on variable terrain. In *ASME 2015 Dynamic Systems and Control Conference*, volume 3, page V003T44A004, 2015.
- [52] Congkai Shen, Siyuan Yu, Bogdan I. Epureanu, and Tulga Ersal. An efficient global trajectory planner for highly dynamical nonholonomic autonomous vehicles on 3-d terrains. *IEEE Transactions on Robotics*, 40:1309–1326, 2024.
- [53] Ziye Zhao, Haiou Liu, Huiyan Chen, Jiaming Hu, and Hongming Guo. Kinematics-aware model predictive control for autonomous high-speed tracked vehicles under the off-road conditions. *Mechanical Systems and Signal Processing*, 123:333–350, 2019.
- [54] Daniel W. Carruth, Clayton T. Walden, Christopher Goodin, and Sara C. Fuller. Challenges in low infrastructure and off-road automated driving. In *2022 Fifth International Conference on Connected and Autonomous Driving (MetroCAD)*, pages 13–20, 2022.
- [55] Graham Loomes and Robert Sugden. Regret theory: An alternative theory of rational choice under uncertainty review of economic studies. 1982.
- [56] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deep-road: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*, page 132–142, New York, NY, USA, 2018. Association for Computing Machinery.
- [57] Jiefei Wei and Qinggang Meng. Adversarialstyle: Gan based style guided verification framework for deep learning systems. In *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 641–648, 2020.

- [58] Jonathan C. Brant and Kenneth O. Stanley. Minimal criterion coevolution: a new approach to open-ended search. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017.
- [59] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *ArXiv*, abs/1901.01753, 2019.
- [60] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *ICML*, 2020.
- [61] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4940–4950. PMLR, 18–24 Jul 2021.
- [62] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Nicolaus Foerster, Edward Grefenstette, and Tim Rocktaschel. Replay-guided adversarial environment design. In *Neural Information Processing Systems*, 2021.
- [63] Michael Dennis, Natasha Jaques, Eugene Vinitzky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS’20, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [64] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob N. Foerster, Edward Grefenstette, and Tim Rocktaschel. Evolving curricula with regret-based environment design. In *ICML*, 2022.
- [65] Nils Wilde, Dana Kulić, and Stephen L. Smith. Active preference learning using maximum regret. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10952–10959, 2020.
- [66] Jianing Zhao, Keyi Zhu, Shaoyuan Li, and Xiang Yin. To explore or not to explore: Regret-based ltl planning in partially-known environments. *IFAC-PapersOnLine*, 56(2):11337–11343, 2023. 22nd IFAC World Congress.
- [67] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, 2015.
- [68] German Federal Ministry for Economic Affairs and Energy (BMWi). Pegasus research project. <https://www.pegasusprojekt.de/en/home>.
- [69] Hardi Hungar. Test specifications for highly automated driving functions: Highway pilot. Retrieved from PEGASUS Project <https://www.pegasusprojekt.de/en/lectures-publications>.

- [70] Eric Thorn, Shawn Kimmel, and Michelle Chaka. A framework for automated driving system testable cases and scenarios. Technical Report DOT HS 812 623, National Highway Traffic Safety Administration, September 2018.
- [71] P. Koopman and Frank Fratrick. How many operational design domains, objects, and events? In *SafeAI@AAAI*, 2019.
- [72] Epic Games. Unreal engine, 2014. <https://www.unrealengine.com/en-US/>.
- [73] Ted Sender, Mark Brudnak, Reid Steiger, Ram Vasudevan, and Bogdan Epureanu. Adversarial scene generation for virtual validation of off-road autonomous vehicles. *Proceedings of the Interservice Industry, Training, Simulation, and Education Conference (IITSEC)*, pages 1–12, 2022.
- [74] Patrick Mania and Michael Beetz. A framework for self-training perceptual agents in simulated photorealistic environments. In *International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 2019.
- [75] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.
- [76] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.
- [77] James R. Baxter, Bogdan I. Epureanu, Paramsothy Jayakumar, and Tulga Ersal. High-speed, all-terrain autonomy: Ensuring safety at the limits of mobility. *IEEE Transactions on Robotics*, pages 1–19, In submission.
- [78] Morgan Quigley. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation*, 2009.
- [79] John Y. H. Wong. *Terramechanics and Off-Road Vehicle Engineering: Terrain Behaviour, Off-Road Vehicle Performance and Design*. Butterworth-Heinemann, 2009.