ABSTRACT

Title of dissertation: ADAPTIVE SAMPLING METHODS FOR TESTING AUTONOMOUS SYSTEMS

Galen Mullins, 2018

Dissertation directed by: Professor Satyandra K Gupta
Department of Mechanical Engineering

In this dissertation, I propose a software-in-the-loop testing architecture that uses adaptive sampling to generate test suites for autonomous systems based upon identifying transitions in high-level mission criteria. Simulation-based testing depends on the ability to intelligently create test-cases that reveal the greatest information about the performance of the system in the fewest number of runs. To this end, I focus on the discovery and analysis of performance boundaries. Locations in the testing space where a small change in the test configuration leads to large changes in the vehicle's behavior. These boundaries can be used to characterize the regions of stable performance and identify the critical factors that affect autonomous decision making software. By creating meta-models which predict the locations of these boundaries we can efficiently query the system and find informative test scenarios. These algorithms form the backbone of the Range Adversarial Planning Tool (RAPT): a software system used at naval testing facilities to identify the environmental triggers that will cause faults in the safety behavior of unmanned underwater vehicles (UUVs). This system was used to develop UUV field tests which

were validated on a hardware platform at the Keyport Naval Testing Facility. This process of developing test cases using simulations and preparing them for deployment in the field required new analytical tools. Tools which are capable of handling uncertainty in the vehicle's performance, and the ability to handle large datasets with high-dimensional outputs. This approach has applications across a wide range of domains, including the generation of self-righting plans for unmanned ground vehicles (UGVs) using topological transition graphs. In order to create these graphs, I had to develop a set of manifold sampling and clustering algorithms which could identify paths through stable regions of the configuration space. Finally, I introduce an imitation learning approach for generating surrogate models of the target system's control policy. These surrogate agents can be used in place of the true autonomy to enable faster than real-time simulations, accelerating the testing process. These novel tools for experimental design and behavioral modeling provide a new way of analyzing the performance of robotic and intelligent systems, and provide a designer with actionable feedback.

ADAPTIVE SAMPLING METHODS FOR TESTING AUTONOMOUS SYSTEMS

by

Galen Mullins

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Satyandra K. Gupta, Chair/Advisor
Professor Hugh Bruck
Assistant Professor Mark Fuge
Professor Linda Schmidt
Professor Ramani Duraiswami (Dean's representative)

Dedication

To my dear wife, Amy

Acknowledgments

My sincere thanks goes to my advisor, Professor Satyandra K. Gupta, for his unwavering support and guidance throughout this entire process. He has been an invaluable mentor, without whom I would have not been nearly as successful at translating my work into publishable results. He has also been patient throughout my many changes of thesis topics as I struggled to align my dissertation research with my full time job at JHU/APL.

I would like to thank my labmates at University of Maryland who were there for me throughout the process; Peter Svec, Krishnanand Kaipa, Eric Raboin, Brual Shah, Michael Kuhlman, and Josh Langsfeld. To my friends Gary Jackson and James Auwaerter for being excellent listeners and always agreeing to review my papers.

I would also like to express my gratitude to my committee members, Drs. Hugh Bruck, Linda Schmidt, Mark Fuge, and Ramani Duraiswami, for their feedback, support, and constructive criticism throughout this entire process. Your involvement has been invaluable.

Special appreciation goes to Chad Hawthorne and Jim Horris. Who brought me onto the RAPT project and gave me the freedom to pursue my research. Then going above and beyond by supporting my efforts to document and publish the results. Giving me a lead role on the RAPT program was a life-changing event and without it this dissertation would not exist.

Next, my thanks goes to the RAPT team; Melissa Huntley, Michael Biggins,

Johan Stewart, Jordan Appler, and Paul Stankiewicz. Together they created the simulation, autonomy, and computing infrastructure which forms the heart of the RAPT program. Thanks also goes to the RAPT sponsors at TRMC; Vernon Panei, Shannon Arnold, and Jonathan Elliot for supporting my research and encouraging the publication of my work. Thanks goes also to Nitin Sydney for asking the difficult questions throughout the many phases of RAPT.

Thanks to my colleges at the Johns Hopkins Applied Physics Laboratory. First the AEODRS team who supported the UGV study; Michael Zeher, Leif Powers, and Rachel Hegeman. To Guatam Vallabha, for creating a test environment in the MATLAB Robotics system toolbox. To Aurora Schmidt, for educating me on formal method approaches. To Bob Bamberger, for being an outstanding supervisor. To Austin Dress, for his advice on neural network design.

Finally, I could not have done this without my family. My amazing wife, Amy, who tirelessly supported me through a decade of graduate school. Your endless love and patience is what kept me going. To my brother and sister, who always had words of support for my work. To my parents, Mike and Janet, who instilled in me a lifelong love of learning and the values required to complete this degree. They set me on this path and I am proud to have finished it.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Motivation

Designing tests is an integral component of developing any software system. The requirements for autonomous systems are designed around meeting certain capabilities, such as autonomous manipulation, self-righting, or retro-traverse to a safe location [1]–[4]. Yet a majority of validation and verification research is focused on fault detection and robustness [5]–[9] rather than overall performance of the decision making software. Currently, the primary way of testing an autonomous system's ability to complete missions is to use simulations of realistic environments[10]–[12]. However, selecting test scenarios which reveal the full performance envelope of the system is still an open question.

Prior research into search-based testing for autonomous vehicles has focused on the development of stochastic optimization techniques to guide tests towards potential collisions. [13]–[18] All of these prior approaches assume that there exists some convex function which can be used to find scenarios where the vehicle transitions from success to failure. In this dissertation I instead assert that the performance of an autonomous vehicle across the testing space is inherently discontinuous. These discontinuities represent the performance boundaries of the system,

locations in the testing space where a small change to the test configuration causes a large change in the vehicle's actions. Techniques which address both the discontinuous and multi-modal nature of a vehicles performance have, to date, not been developed.

In order to address this challenge I propose a set of new algorithms for software-in-the-loop testing of autonomous systems which automatically explore the testing space for discontinuities in the systems performance. In this dissertation, I introduce adaptive sampling techniques which explore the test-design space, and discover the transitions between performance modes. By establishing where these transitions occur it becomes possible to create sets of focused tests which are used to identify the transition factors that cause changes in vehicle performance. Techniques for the automatic generation of test cases in these transition regions have a variety of applications across the following domains:

- **Requirements Design:**

  Developing requirements for autonomous systems is a recent challenge that impacts many government and commercial projects.[3], [4], [19], [20] Writing the physical requirements for a robotic system such as power, mobility, and reliability is a well understood process. However, developing requirements for the decision making components of an autonomous system remains an open question. Developing techniques which can validate the requirements for tasks such as self-righting will help inform the development of capability requirements for the autonomous systems.

- **Designing Field Tests:** There are few portions of any research and development program that are as expensive as designing and executing field tests. Both the time and expense involved means the number of tests that can be performed are extremely limited. The current process is to have subject matter experts hand design vignettes meant to test each behavior of the system individually. However, it is difficult to predict the emergent behaviors that occur when multiple competing objectives are active. In addition, any desire to challenge the system is offset by the desire to avoid costly failures. Thus the test designers need a thorough understanding of how they can expect the system to react in a variety of situations [21]. Finally, they need a method of quantifying the certainty of seeing a specific behavior from the vehicle for a given test scenario.

- **Debugging autonomous software systems:** Perhaps the most obvious application of adaptive test design debugging the system while the autonomous system it is still in the design stages. The advantage of the methods introduced in this dissertation are that they allow for the development of delta-test cases for changes in behavior whereas current techniques only focus on fault detection. These delta-tests show examples of both intended and unintended behavior that are invaluable for finding bugs and improving the software.

- **Hardware design:** Stochastic optimization as a method for generating mechanical designs has been gaining traction in recent years, and new methods for searching across multiple objectives can only push the boundary further.

It is possible to use the techniques introduced in this dissertation as another computational design method. For example; a method for computing the self-righting capabilities of a robot based upon its geometry can be used either by a human designer or as an objective function of evolutionary algorithms.

There are three major challenges that this work addresses that have not been addressed previously in this field.

- *High dimensional state spaces:* Generating meta-models for high dimensional functions is an open problem. Due to the computational expense involved with attempting to fit non-parametric models to high dimensional system few adaptive sampling methods have been applied to systems beyond 3 dimensions. Before adaptive sampling techniques can be applied to test scenario design, the algorithms must scale gracefully to an arbitrary number of input dimensions.

- *Nonlinear Discontinuous Black-box Systems:* The performance surfaces for autonomous systems are inherently non-linear and discontinuous. Meaning traditional design of experiments and global regression approaches cannot be applied. In addition, a blackbox approach means a user cannot make any *a priori* assumptions about the underlying function. This limits the amount of tuning which can perform to the hyper-parameters of the meta-models.

- *Objectiveless Optimization:* Search-based testing and surrogate optimization techniques require objective functions that can be complicated to design and are tailor made for a specific platform and mission. Creating a design process

that is adaptable to a variety of systems requires algorithms that can utilize only the pre-existing metrics logged from the simulations.

1.2 Goal and Scope

This dissertation presents novel algorithmic approaches for automatically generating test scenarios for black-box autonomous systems. The main research issues addressed are as follows:

i *Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles:* Designing field tests for autonomous unmanned undersea vehicles (UUVs) is a challenge that has not been addressed before in a formal manner. Tests in the past have been designed as stand-alone vignettes which test each behavior of the system independently. Traditionally, these are designed around the physical capabilities of the system. Rather than tests which explore the capabilities of the decision making software. This was the motivation behind the Range Adversarial Planning Tool (RAPT). RAPT is a software suite for simulation-based testing of autonomous vehicles that automatically generates sets of diverse and challenging test scenarios. My approach does not require custom objective functions, instead searching for discontinuities in the high-level mission criteria of the simulation. I utilize an adaptive sampling framework that uses a light-weight meta-model that balances exploration, generating diverse test cases , with achieving high resolution in the areas of interest. I then apply unsupervised clustering techniques to identify

distinct sets of cases that form the boundary sets that divide regions of stable performance. These are then returned to the user, along with estimates of the variable sensitivities, in order to inform the test-engineers so that they can design more appropriate tests.

ii *Design and Execution of Live Field Tests for Autonomous Vehicles:* Given a suite of relevant test scenarios generated via the RAPT framework, the next logical step is to take those tests and execute them on a hardware platform in the field. This is a challenge known as the reality gap; where machine learning results developed in a simulation environment are applied to the real world. Due to modeling error in the simulation environment and uncertainty during execution in the field it is not reasonable to expect the exact same vehicle performance in both environments. Therefore, it is necessary to adapt the techniques for selecting test cases to account for uncertainty. As well as develop new tools which isolate the behaviors we are attempting to test. Using distance from a performance boundary as a method for gaging the robustness of a given scenario is one way to gain this assurance. To this end, I analyzed the performance of the adaptive sampling algorithms on both a synthetic stochastic system and a non-deterministic UUV simulation. From this analysis I developed algorithms for identifying sub-clusters of specific behaviors from the hierarchical scoring structure output by the simulation. From these I was able to identify test cases which were both informative and reproducible in the field. These test cases were run using an OceanServer IVER

UUV at the Keyport Naval Test range under adverse conditions. Finally, I provide a post-test analysis of these results as a case study of the effectiveness of the RAPT test-generation framework.

iii *An Adaptive Sampling Approach for Autonomous Self-Righting Validation:* Robots operating in dynamic, unstructured environments run the risk of tipping over, thus becoming unable to move normally. To restore normal mobility, a computational framework was developed to generate self-righting plans for arbitrary robot geometries on planar surfaces of arbitrary slope. However, the previous instantiation of this framework required an exhaustive search of the configuration space to identify transitions between stable and unstable states. This restricted its ability to analyze systems with many degrees of freedom, since the number of queries required increases exponentially with dimension. In this dissertation, I propose using adaptive sampling to query preferentially along discontinuities, enabling the identification of higher quality transitions using fewer queries. In addition I extend the previously two-dimensional framework to support high-fidelity models of robots in three-dimensions. These updates required changing the way I represented the robot in configuration space (C-Space) and a new adaptive sampling algorithm which supported constrained sampling along high-dimensional manifolds. To support the generation of self-righting paths I also developed a set of graph-based clustering techniques for determining the connectivity between stable robot states. I then compare the resulting transition graphs against those previously val-

idated on 1, 2, and 3 degree of freedom robots. Finally, I demonstrate the techniques working on a high-fidelity model of a Naval bomb-defusal robot with a 5 degree of freedom manipulator. Which was too complex for the the previous grid-based sampling method to analyze.

iv *Deep Imitation Learning for surrogate meta-models of autonomous vehicles:* Developing surrogate agents which can approximate the behavior of a vehicle's control policy can allow for faster generation of results than using the full autonomy under test. This is of particular use when the full autonomy requires multiple linked machines to operate or is otherwise restricted to only real-time operation. Recent advances in neural networks have allowed for deep imitation learning to be successfully applied to both individuals and teams of agents. In this work I explore the application of deep imitation learning to replicating the control policies of autonomous vehicles using only externally observable features. I demonstrate how these surrogate models can then be applied to predict performance in unknown scenarios. This study shows that not only are the performance boundaries of the surrogate agent highly correlated with those of the actual autonomy but that the distance from the performance boundary is an excellent measure for determining the accuracy of the surrogate's behaviors.

Chapter 2: Literature Review

The validation and verification of autonomous systems has been an active area of research in the past few years. Particularly, the area of search-based testing, where optimization techniques are used to generate test-cases for autonomous systems. In addition, recent advances in meta-modeling techniques are increasing the ability of test engineers to model and predict the behaviors of complex systems.

This review is divided into four sections. The first addresses the different approaches that have been applied to software testing. From designing requirements and testing priorities, to classical design of experiments, and formal methods. The second addresses other works in the domain of test-case generation for autonomous vehicles. The third covers the topics of surrogate optimization, meta-model generation, and adaptive sampling. The fourth covers methods for machine learning and autonomous planning algorithms.

2.1 Test design

In this subsection I cover traditional software testing approaches. Including system and requirements design, design of experiments, formal methods, and search-based testing methods.

2.1.1 System Design Processes

Industry and government organizations have been working together over the past decade to develop extensive requirements and test procedures for autonomous vehicles[3], [4], [19]. This includes establishing standard coding practices, defining reference architectures for all of the electrical and computational components, and developing certification testing requirements. However these requirements are built around an assumption that a human operator will ultimately take control of the system. No performance and behavioral certification standards yet exist for autonomous systems. How to validate these systems remains an open question.

Multiple design processes have been proposed to address this challenge[1], [2], [22]–[26]. What all of these approaches have in common is the approach of identifying *capabilities* requirements. Which define certain tasks that the autonomous system must be able to complete in the designated environment. In addition, they all provide methods for prioritizing tests based on cost, stakeholder need, and system coverage.

The Multi-relationship evaluation design (MRED) [1], [2] is an exceptional example of a well defined process for relating technology components and capabilities, identified as techology test level (TTL), with the performance metrics and test resources. For example a TTL-metric pair might represent a single joint and its range of motion. It then combines the stakeholder preferences, technological maturity, and operator skill to score each TTL-metric pair. These scores determine the priority of the test.

Fuzzy-logic rule-sets are another popular method for developing and prioritizing tests for autonomous systems [22], [26]. Fuzzy logic is attractive in that it can create continuous functions from sets of logical rules without the user having to explicitly define the relationships between all the inputs. This allows the designer to address all of the desired inputs as pair-wise or lower order interactions. Which aligns with the ways user requirements are defined. One way to assemble these criteria are from the physical requirements and operator requirements [26]. Another is to rate the priority of each task and sub task of the mission in a hierarchical manner along with it's estimated costs and return on investment[22]. In both cases the test priority is based upon expected impact rather than expected performance.

What all of these formal design strategies have in common is that they provide methods for identifying the performance criteria of the system and estimate their priority to the end user. What they lack is a method for actually performing the validation. This is where test-case generation and software testing techniques come into play.

2.1.2 Design of Experiments

Design of Experiments (DOE) is a systematic method for determining the relationships between the inputs and outputs of a system using statistical analysis. It is primarily used to take complex systems with large numbers of inputs and devise ways to minimize the number of experiments required to either identify a specific property of the system or fit a regression model [27]–[30].There are three areas that

fall into the domain of DOE that are of particular interest to this dissertation; optimal design, sensitivity analysis, and combinatorial testing.

The goal of optimal design is to create a set of test cases which maximize the information gained from running the experiments [27], [31], [32]. It is assumed that the user has no a priori information about the underlying function. These are typically used for model-fitting purposes and are appropriate for systems with continuous input and output spaces.

Frequently, optimal design will feed into sensitivity analysis [33], [34] which identifies the factors which have the greatest impact on the output of the system. This is also known as analysis of variance (ANOVA) and is an approach I utilize as part of my mission design process and variable scaling methods in Chapter 3. The primary drawback of these techniques is that they only model the linear and second-order interaction effects of variables. As I shall discuss later in this dissertation the performance surfaces for autonomous systems are inherently non-linear and discontinuous with many local effects. This reduces the effectiveness of global sensitivity modeling techniques.

The most important tool that Design of Experiments provides with regards to software systems is combinatorial testing [6], [35]–[38]. Applied to systems that have discrete inputs, combinatorial testing is the process of reducing the entire testing space by testing non-interacting inputs simultaneously. These are referred to as *orthogonal arrays* and are utilized heavily as part of a standard software testing regime. Frequently the goal when designing a combinatorial test regime is to create a *covering array* which, as the name implies, covers all unique permutations that

satisfies the t-covering property. Which means that all permutations for all t-way interactions are represented.

2.1.3 Formal Methods

A great deal of recent research in the domain of software validation and verification has focused on formal methods [39], [40]. Formal methods are any process where a mathematical model describing the system's operation is generated and tested for exceptions which break the specifications. It is explicitly a white-box testing method that is popular due to the provable guarantees it can provide about the robustness and reliability of the system. Models that have been used in the past include finite state machines [41], [42] and process algebras [40]. They can be utilized to provide coverage guarantees [43], [44] of a testing suite. The drawback of these techniques is that the resulting model must fully describe the decision making process of the software and test engineers must have full access to the model. Given the increasing complexity of autonomous systems and the black box nature of proprietary software, these limitations prevent these methods from being applied to many systems.

Physics based verification is another application of Formal Methods which instead of modeling the systems software models the kinematics and constraints of the physical system. It has been successfully applied to provide provable guarantees of safety [45], [46] for both aircraft [45] collisions and to automatic brake systems in cars [46]. These techniques work by computing the inevitable collision states of the

vehicle and validating any proposed trajectory generated by the system to see if they violate these constraints. The challenge with these approaches is that any trajectory the vehicle performs must be solved analytically and the proven formulations cannot support kinematics described by transcendental functions.

While model-checking is primarily the domain of white-box testing it is possible to apply it to black-box systems using model learning techniques [47]. These methods continually generate candidate finite-state machines and apply hypothesis checking against the true system to update their model. If counter-examples are found they update their internal model until it is consistent. These generated models can then be checked against the formal specifications as executed during white-box testing.

2.1.4 Combinatorial Testing

There are many methods of performing tests against black-box software systems but it remains an NP-hard problem. Exhaustive testing for a complicated system with a large number of inputs is in-feasible due to the shear number of tests required. Approximate methods such as space-filling designs or pseudo-random techniques like Monte Carlo testing are often implemented as a quick approach to validating a system. A more principled approach is to utilize combinatorial testing [36][37], a method of reducing a large testing-space into a smaller set of orthogonal tests. Combinatorial tests are designed to detect faults and take advantage of the fact that 90% of faults are caused by single or two-factor interactions. With faults

becoming progressively less likely as the number of interactions increase [35].

The problem is that even eliminating higher order interactions the state-space is still too large to test all permutations. Thus, the focus becomes on not just generating combinatorial tests but trying to optimize the coverage of the selected tests [38].

2.1.5 Search-based testing methods

Search-based testing is a meta-heuristic test-case generation technique which utilizes a fitness function to rate candidate tests and uses global optimization techniques to generate queries that will find faults [38], [48]–[51]. It is applied when the search space is too large for exhaustive testing but combinatorial testing cannot be applied. Genetic algorithms are perhaps the most frequently applied approach [50], [51], followed by particle swarm optimization [38], and pattern search approaches [52]–[55]. Many of these approaches are used in a white-box testing fashion, and build their fitness function around whether the test input executes a specific branch or module of the code [51], [53]. Where as others take a blackbox approach and only measure the difference in outputs [52], [54]

Mutant-killing tests are one of the benchmark methods for evaluating these techniques [38], [50], [51], [55]. In this procedure the target program is altered semantically, for example logical operators will be reversed or erroneous arithmetic procedures will be added to alter the input and output parameters of a module. A set of inputs is considered to have "killed" one of these mutants if it results in a

different output than the original program. Many search-based testing methods use these metrics as the fitness function to drive the search. The goal is to generate suites of inputs that kill the most mutants in the fewest trials.

2.1.6 Test-case Diversity

In addition to finding maximally informative test cases it is also important to achieve some measure of situation coverage [25] . Which is to say we want to ensure we have representative cases from a wide variety of situations. This problem can be broken up into two steps. The first is search approaches which prioritize coverage as their test-case generation metric. Known as Coverage-Directed test Verification (CDV) [56], [57] these approaches attempt to apply formal methods to ensure the tests have fully explored the testing space. This is a combination of a simulation-based method with formal method definitions for checking properties and estimating coverage. It has been applied to robotic tasks such as robot to human object handover [58].

The second step is selecting a diverse set of test scenarios to be executed in the field. An ideal set of test scenarios will exercise all the autonomy behaviors (expected and unexpected) in the fewest number of trials. Creating the measure of diversity typically depends on a selection of distance metrics [59] for the input and output spaces. Techniques such as Adaptive Random Testing (ART) [60] attempt to order test cases in such a way that they will exhibit a diverse set of failures in a single sequence. Optimization techniques using genetic algorithms such as DIV-GA

[61] randomly generate and optimize groups of test cases with the goal of detecting the maximum number of faults for a given number of tests.

The majority of these diversity criteria require some way of discerning between different fault types. In this dissertation I utilize clustering algorithms for identifying different classes of performance. Applying clustering algorithms has been a common approach [62] [63], [64] for classifying subsets of scenarios based on their similarity. Detecting anomalous behavior by searching for successes in clusters composed primarily of failures has been used to correctly identify coincidental correctness during fault localization [65] .Several popular clustering methods, including K-Means, Mean Shift, and Agglomerative Hierarchical Clustering (AHC) have been applied and compared for categorizing web-application tests and Mean-Shift had a slight advantage [66] over the other approaches. These research efforts informed the design of the clustering methods discussed in Chapter 3.

2.2 Testing autonomous systems

Simulation-based testing of autonomous systems has not been explored as extensively as the field of black-box software testing but has started to receive a great deal of attention in the past couple years [13], [14], [23], [52], [67]–[69]. In this paradigm the input to the system is a scenario configuration being executed by the autonomy, the challenge is discovering scenarios or sets of scenarios which fully exercise the autonomy's capabilities and reveal faults in the system.

The approach that is the most closely related to this dissertation is search-

based falsification. This method of test-case generation utilizes stochastic optimization to find cases which violate the systems specification [70]. These research efforts typically use robustness criteria to drive the search towards areas where the violations may occur. One area where this has been of particular interest is the study of aircraft collisions [13], [14], [45], [67], [71]. Validating that autonomous collision avoidance systems can successfully avoid encounters that will lead to inevitable collision states. While the foundational work has been provable safety through formal methods [45] a great deal of attention has also been spent on search-based test-case generation methods [13], [14], [67], [72]. These works parameterize encounters based on approach angles, velocities and time of arrival. They apply a fitness function that estimates the probability of collision from the closest point of approach, or use Hsu's method for determining collision probability [73]. Then by running simulations of the encounter against the collision avoidance software they optimize for cases where collisions occur.

In addition to scenario-generation efforts which parameterize the initial state of the simulation there are also tools for the testing of temporal properties of hybrid systems. These include S-TaLiRo [74], Breach [75], and RRT-REX [76]. The S-TaLiRo toolkit [74] allows users to define their requirements using temporal logical language. It then turns these requirements into minimal representations and uses a robustness degree computation to drive a stochastic optimization towards locations which violate the requirements. It parameterizes the scenario in terms of the input parameters to the controller and trajectory control points which are then optimized to find paths which violate the robustness criterion. RRT-REX is a framework

that leverages notions of coverage to optimize bug-finding performance. It uses an rapidly-expanding random tree algorithm to select states to test and each rooted path in the tree corresponds to a partial simulation (a simulation over an abbreviated time horizon).

The S-TaLiro toolbox has recently been utilized to generate test-scenarios for autonomous cars [17] where the primary scoring criteria was avoiding collisions in the event of lane-crossing behavior. This work describes how a parameterized scenario space can be used to generate test cases guided by stochastic optimization of a robustness metric based upon a continuous-time signal. This toolbox was used to test autonomous cars as part of Sim-TAVT [77], a recently published approach that utilizes a framework and objective similar to those in Chapter 3. This work uses relative collision speed as the robustness metric and use simulated annealing to drive the search towards cases where the system just barely fails. This is like the approach used in [13] for generating UAV collision cases and both works have direct relations to the approach described in this dissertation.

Another testing strategy is to search for challenging test cases rather than for falsifying test cases. The concept of a challenging test case, one where it is possible to succeed but will stress the decision making of the autonomy, has only recently begun to attract the attention of the research community [69]. It has been applied to testing obstacle avoidance and navigation of ground vehicles in a 3D environment [23] and is the basis for many of the aircraft collision generation frameworks [13], [14], [67]. In each of these cases the conclusion was that the more challenging the autonomy found the scenario the more useful the collected data was for evaluating

and improving the system.

Surrogate-based optimization has also been used to generate test scenarios. Radial basis functions have recently been applied to model distances between cars and drive a search for collision cases [16]. This study confirmed that surrogate models could help find solutions faster than Monte Carlo methods but have only addressed small numbers of runs and a small number of dimensions.

Where my work differs is my decision to eschew the objective functions used to guide these optimization-based approaches. Instead I make the assumption that there is no convex function which can be used to drive the optimization and instead search for discontinuities in the output space of the vehicles performance. The second difference is that previous falsification methods stop once they have found a single failure case and further simulations only refine on that result. Since I am interested in generating a diverse set of test cases my adaptive sampling methods continue to explore sparsely sampled regions as long as computational resources are available.

2.3 Optimization, Modeling, and Clustering

In this subsection I address the history of two techniques utilized in this dissertation. Adaptive sampling, which forms the basis of the algorithms used for generating test cases, and high-dimensional modeling.

2.3.1 Adaptive Sampling

In this dissertation I am interested in two closely related methods for querying black-box systems; surrogate optimization and adaptive sampling. Both of these technologies could be classified under the umbrella of active learning as they are methods for intelligently choosing queries in order to update an underlying model. They are utilized in the analysis of complex systems, where taking samples requires expensive simulation or physical experiments [78]. Where they differ slightly is in their goal. Surrogate optimization uses the underlying model to drive the search towards a global optima. Whereas adaptive sampling is generally applied to maximize the information gained from the entire queried data-set in order to create as accurate meta-model as possible.

Adaptive sampling for surrogate model generation has been shown to be competitive with optimized space-filling designs in the design of experiments (DOE) community for black box systems [79], [80]. The surrogate models are typically used to aid in the optimization of complex systems where taking samples requires expensive simulation or physical experiments [78]. This process has been applied to a variety of optimal design tasks such as manufacturing mold designs [81], [82], chemical processes [83], or optical trapping [84]. A variety of surrogate optimization methods such as Kriging (also known as Gaussian processes) [85]–[88] and neural networks [83] have been used successfully to generate sets of test cases. Kriging was used in [86] with an information metric based on finding areas of maximum variance in the test values. The technique of [86] is similar to the one I present in

Chapter 3. One of the leading methods for adaptive sampling is the LOLA-Voronoi method [79], [80] which uses local estimates of the gradient along with Voronoi decomposition to estimate the best regions for subsequent sampling. This study was shown to reduce the number of samples necessary to minimize cross-validation error compared to space-filling techniques.

The techniques behind adaptive sampling also can be used to make any sampling-based planning method more efficient . Trajectory generation tasks which require complex plans in real-time such as traversing congested harbor traffic [89], liquid pouring tasks [90], hexapod locomotion [91], or cleaning pliant surfaces [92], [93] all benefit for using adaptive sampling methods to reduce the total number of computations.

My work differs from these methods by choosing samples along the system performance boundary rather than choosing samples that minimize the global error in the surrogate model. This sampling strategy, querying regions where critical transitions occur, has been explored before in multiple domains and has been used to train Radial Basis Functions (RBF) [94], Kriging models [85], [94], and Support Vector Machines (SVM) [95]. In each of these, the metrics guiding the sample selection were based on the properties of the underlying meta-model and the selection of points tailored to improve its accuracy. In these works the surrogate model driving the search is the final product. Whereas in this dissertation the focus is on obtaining a diverse set of test-cases.

2.3.2 High-dimensional modeling

The problem facing many adaptive sampling approaches is their ability to scale to large numbers of dimensions. This leads to the more general problem of attempting to model high-dimensional functions or create classifiers for high-dimensional functions [96]. The general approach is to break the system into a linear combination of underlying functions of first-order through Nth order. Research has shown that there are typically few higher order interactions even in high-dimensional systems[96]. By modeling it as a series of lower interactions it simplifies the modeling problem.

The underlying function to represent each of the terms varies by application, from standard polynomials to Radial-Basis functions (RBF)[97] [98] to support vector machines (SVM)[99]. The challenge of utilizing HDMR methods is that they are often based around specific sampling schemes designed to maximize accuracy. For example random sampling (RS-HDMR)[100] assumes a uniform distribution across the entire state-space while Cut-HDMR [97] performs a pattern search along individual dimensions while holding all others fixed. While some adaptive methods have been developed [99] they don't deviate far from the other approaches. Unfortunately, none of these approaches can be directly applied to the problem of discovering the transition regions of a system.

Instead of attempting to perform a regression on the global performance surface another possible approach is to model the sub-surface that describes the performance boundaries. One approach that has promise is to utilize manifold learning

to create local-planar approximations [101] in order to encode a higher-dimensional manifold. This is also similar to geometry based ensemble techniques that have been applied in the past to describing the decision boundaries of a black-box classifier. [102] Both of these approaches could be potentially applied to modeling the performance boundaries of an autonomous system, which is discussed further in the Future Works section.

2.4 Planning and Control

Many of the techniques within this dissertation have applications beyond test design and like-wise many algorithms used for planning and control have direct relationships to this dissertation. In this subsection I will discuss a few research areas which have direct impact on the topics of this dissertation. The ability to transfer results from simulation to the real world has relevance to the work in Chapter 4. Sampling-based planning methods for manipulation tasks are directly related to the work Chapter 5. Finally I offer a brief background on the field of imitation learning which is utilized in Chapter 6.

2.4.1 Transfer Learning

Transfer learning is the process of taking a model which was trained in one domain and applying it to another similar domain which it has never seen before [103]. The sub-set of transfer learning that is the most relevant to this dissertation is the problem of transferring between a simulation and the real world. This problem is

known as the Reality-Gap [104], where the differences between the simulated world and the real world prevent solutions found via simulation to be directly applied to the hardware system. The reality-gap poses a major issue for the scenario generation software proposed in this dissertation. Even if the software is capable of finding salient test cases in simulation the ability to reproduce those same behaviors on the real platform is uncertain.

One approach for estimating the transferability of a result is to generate transferability metrics. Rather than focus on improving their simulation environment previous studies have searched for ways to estimate their confidence in the simulator for certain regions of the state space [104]. The approach of creating "transferability" metrics [105] by iteratively alternating between hardware and software runs is interesting but less useful if, as in our case, the test engineer only has one chance to run successful field tests.

Another method for overcoming the reality gap is by first identifying which solutions can be transfered between two simulators [106], [107]. The idea behind that approach is any controller which cannot handle transfer between simulation environments with differing fidelity will not be able to handle the transfer to reality. This could work for our simulation-based testing approach but assumes that multiple simulation environments are available, which would double the integration costs for any autonomy under development. One solution to the multiple simulation problem that has been recently developed has been creating neural network approximations called World Models [108]. While still a nascent effort these world models could automatically be generated to create lower-fidelity high-speed simulations. When

coupled with the imitation learning surrogate agents in Chapter 6 this could be a potential avenue for making faster-than-real time predictions of vehicle performance.

When transitioning from test design to test execution the ability to recreate the exact situations which occurred in simulation will be limited by physical and practical considerations. One solution to this is to use mixed reality frameworks, which combine live and virtual elements. These simultaneously allow the tester a wider range of possible scenario configurations while also providing a method of testing autonomous behaviors on hardware while limiting risk to humans or other vehicles.[109], [110]. One such framework that is currently being deployed at military test ranges was developed under the Safe Testing of Autonomy in Complex, Interactive Environment (TACE) program [110].Designed to test UAVs, it provides an infrastructure that allows the user to run field tests with live-virtual constructs (aka a mixed reality environment). This allows for dynamic additions of mission elements such as obstacles, other agents, and simulated weather conditions. In addition a watchdog autonomy monitors the activity of the UAV and will throw an alert and take over if it makes a mistake that would lead to unsafe behavior (e.g. collision).

2.4.2 Sampling-based Planning

The motion planning methodology introduced in Chapter 5 falls under the family of sampling based planning methods, which have proven in the past to be highly efficient at generating motion plans for high-degree of freedom systems [111].

The method for generating those random samples can vary, from attempting to sample more densely near obstacles [112] to using information based methods to identify where valid trajectories are likely to occur [113]. Our sampling methodology has the most in common with Probabilistic Road-Map (PRM) techniques which focus on obstacle regions or boundaries [112], [114], but differs from these in both minor and major ways. In a minor fashion, a different constraint drives our search, instability, and we want to collects samples from both sides of the boundary. More important to our work is that, unlike those techniques, we don't have a priori knowledge of where the boundaries of our system are. We must instead infer where these boundaries lie from querying our simulation.

Motion planning methods for manipulation share the closest resemblance to our self-righting problem as both are concerned with motions that result in contact with another object or surface. These motion planners require the application of constraint-based sampling methods e.g. AtlasRRT [115]or CBIRRT [116]. In these systems, there are lower-dimensional manifolds where the robot is in contact with the target object within the larger C-space of the system. One general method for solving complex sets of contact constraints is CBIRRT2 [117], which has a similar framework to our own as it constrains motions to specific manifolds and searches for "bridges" where it can pass between different manifolds. Unlike these motion planners, we are attempting to characterize the entire C-Space. In addition, their approaches have no way of preferentially guiding samples toward the bridges between manifolds. Our system also contains directional transitions that are not supported using a bi-directional sampling process.

Path planners capable of respecting stability constraints have been well studied in the past [118]. The primary way our approach differs from these is that we are seeking to exploit the transition from stable to unstable instead of avoiding it. In our framework, each node in our graph represents an entire region of connected C-Space where the robot is stable. These regions can easily be explored using PRM techniques but the edges describing how the robot can transition between them have not been addressed in prior literature. While direct planning techniques for these types of discontinuous contacts exist [119] they are currently to expensive to be used as a local planner for a PRM-based search. In fact for this research we eschew the use of a local planner entirely. Instead using clustering algorithms to generate our connectivity graph within and between stable regions.

The self-righting approach we take also bears resemblance to gravity powered reorientation of convex objects via underactuated manipulators [120][121] [122]. Indeed, the projection operation we use to ensure our samples fall on the stable manifolds of our system can be defined using the capture regions of polyhedra [120]. While analytical approaches directly define the tipping points for convex hulls, they unfortunately cannot be applied to our objective of finding the critical thresholds. This is because our approach depends on changing the shape of the convex hull, which occurs as the robot moves, rather than manipulating a static geometry. This coupling of actuation and geometry makes the application of analytical solutions difficult.

2.4.3 Imitation Learning

Imitation learning is a rapidly growing field of research [123], especially popular in the domain of robotic manipulators [124]. It provides an alternative to the traditional approach of designing motion planners, instead enabling humans to train robots without being familiar with the details of robot operation. Sometimes referred to as inverse reinforcement learning [125], in imitation learning a robot attempts to learn a policy that replicates that of its teacher. Rather than directly copying the initial instructs by rote the goal for this new policy is to allow the robot to adapt to variations in the environment while still performing the desired task.

There are multiple approaches that can be used to generate predictive models of an agent's behavior. Learning the values for a parameterized controller via observation of a human demonstrator is one such approach [126], [127]. The approach I take in this dissertation is to build a neural network which takes sensor data as input and control actions as output [123]. The drawback of this approach to imitation learning is that while methods which utilize pre-recorded data such as Generative Adversarial Imitation learning (GAIL) [128] provide compelling results they do not perfectly replicate the original agents actions. Attempts to create "ghost" agents has been fairly successful when applied to pre-recorded data of soccer games [129]. However, this was only successful as the ghost agents had not ability to control or affect the state of the game. When the agent is actually playing the game rather than only observing and predicting we instead require online methods.

Another area where creating exact models of other agents is of utmost impor-

tance is Opponent Modeling [130].The goal of opponent modeling is to build models of the opposing agent which can be used to create plans to defeat their particular strategy. It has typically been applied to tasks such as poker [131], real-time strategy games [132], and unmanned surface vehicle blocking [133]. This bears a close resemblance to the goal of creating tests. As in both cases it is critical that these opponent models effectively capture the failures and faults of the opponent. One approach of note is the Deep Reinforcement Opponent Network (DRON) [134] which provides a general method for generating these agents. However, we discovered that Q-learning methods were not precise enough for our purposes and instead turned to the related area of behavioral cloning methods.

Behavioral cloning using DAgger was introduced in [135] as a method of directly replicating a human operator's control actions. In this work they demonstrated that expert-level performance could be approached after only 12 annotated simulations. Since introduced, it has been successfully applied multiple times in simulation [136]–[138] and on hardware [139]. It has also been applied to learning from human experts [135], [139] and for encoding more computationally expensive algorithms such as Model Predictive Controllers [136]–[138]. However, in all of these cases the scenarios and behaviors were relatively simple, i.e. avoiding obstacles and tracking ideal trajectories. What has not been addressed before using DAgger is the effect of autonomous decision-making on a system with multiple behavioral modes.

2.5 Summary

Despite the wealth of research into the field of black-box software testing developing tests for autonomous vehicles is still an open problem [3], [4], [19], [21]. Search-based test approaches have been extensively explored as a method for generating test scenarios and test plans [7], [13], [14], [48]–[53], [55], [68], [71], [73], [85], [86], [140] however they fall primarily into two categories. The first are methods for optimizing priority and coverage [22], [43], [50], [51], [53], [55]. These attempt to characterize the entire test space but require known models of the system which means they can only be applied for white-box testing. The second are global optimization methods which utilize objective functions that describe the quality of the mission [13], [67], [69], [141]. These are more appropriate for characterizing black-box systems but have several drawbacks. The first is that these objective functions require extensive knowledge of the mission and the software to design. The second is that global optimization techniques will provide highly refined solutions in a single region but won't explore enough of the space effectively characterize the test system. My approach solves both these problems by utilizing adaptive sampling to preferentially explore the space. Instead of requiring an objective function it derives one from high-level mission performance metrics. In addition the approach in this dissertation differs from traditional adaptive sampling techniques by eschewing the standard strategy of sampling to maximize global accuracy but instead seek to maximize resolution in the regions of interest.

Chapter 3: Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles

The work in this chapter was published in the following venues,

G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 1443–1450

G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, J. D. Appler, M. H. Biggins, K. Chiou, M. A. Huntley, J. D. Stewart, and A. S. Watkins, "Delivering test and evaluation tools for autonomous unmanned vehicles to the fleet," *Johns Hopkins APL technical digest*, vol. 33, no. 4, pp. 279–288, 2017

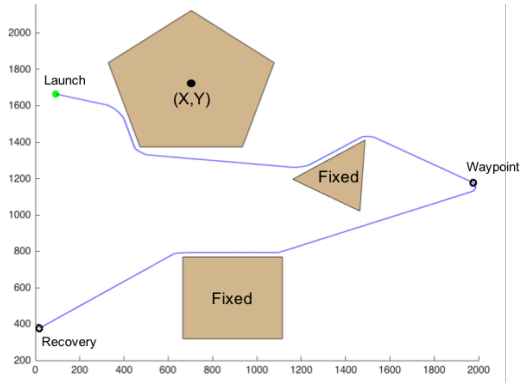
G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *Journal of Systems and Software*, vol. 137, pp. 197–215, 2018

+

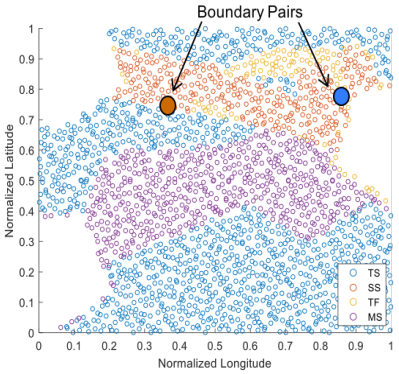
3.1 Introduction

As autonomous vehicles become more complex, understanding how they will behave in complicated and uncertain environments poses a greater challenge to both the engineers who the write the underlying code and the testers validating the system. The software controlling autonomous vehicles comprises many different integrated software modules. Designers of the system may have expertise in the individual modules that form the decision-making components, but the complex interplay that results in the final emergent behavior of the system cannot be easily characterized or predicted.

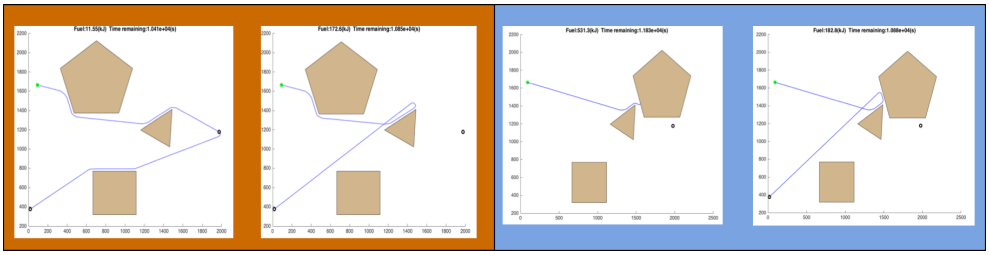
For example, consider an unmanned underwater vehicle (UUV) tasked with a covert survey mission. The multiple subsystems and behavioral modes of the UUV must work in concert in the presence of competing priorities, i.e. offsetting the risk of detection when surfacing with the need to localize via GPS. Competing priorities are of particular concern for long duration missions where the vehicle must transition among multiple mission objectives [21]. These systems can exhibit a variety of performance modes, which we define as discrete types of behaviors that can be derived from observable metrics of the mission. For example, colliding with an obstacle, returning home early, or completing the mission successfully are types of performance modes. It can be difficult to provide guarantees of the system's decision-making capabilities without discovering all of the possible performance modes. This requires both a simulation framework capable of exercising the autonomy realistically [145] and a suite of tests that provide coverage of the operating space [25].



(a) Navigation Scenario for Autonomous System



(b) Performance Modes



(c) TS/SS Boundary Pair

(d) TF/SS Boundary Pair

Figure 3.1: Example of a simple navigation scenario for an autonomous system (a). The mission is to travel from the launch point to the goal waypoint and then back to the recovery point. The testing space consists of the (X,Y) position of the pentagonal obstacle. The autonomy is scored based on whether it reaches the goal waypoint and recovery waypoint. This leads to 4 performance modes: total success (TS) for reaching both waypoints, mission success (MS) for only reaching the goal waypoint, safety success (SS) for only reaching the recovery point, and total failure (TF) for reaching no waypoints. The performance mode plot (b) shows the resulting performance mode for each (X,Y) position of the pentagonal obstacle. The highlighted dots on (b) illustrate two examples of performance boundary pairs (c)(d) that exist in the testing space.

Within this ideology, a test scenario can be viewed as a single sample of the entire testing space. One issue immediately encountered is that the number of parameters in the testing space quickly increases when attempting to simulate realistic missions. Moving and static obstacles, environmental factors, time constraints, and mission types are just a few of the different parameters an engineer may wish to vary as part of his testing regime. As missions and environments become more complicated, the number of parameters that constitute the testing space becomes too large to test the autonomy software under all permutations, resulting in the familiar curse of dimensionality. Simulated mission duration could be several hours long and if the autonomy under test cannot be run faster than real time, the number of samples will be severely restricted. Therefore, we must carefully select the scenarios that will be simulated with the goal of obtaining the maximum amount of information about the autonomy under test.

To do this, we focus our attention on performance boundaries, defined as regions in the testing space where small changes in the scenario result in transitions between performance modes. The canonical example is how a small change to the position of an obstacle can cause the system to take a different path and fail to reach its goal as illustrated in Fig. 3.1. With regards to testing, scenarios that lie along such performance boundaries are high-value because they evoke behavior and decision changes made by the autonomy. Due to the black box nature of autonomous systems, understanding where these transitions occur is key to predicting the performance of the system and is useful for both design, i.e. fixing software bugs, and validation purposes, i.e. understanding the likelihood of triggering a certain

behaviors in different regions of the testing space. Furthermore, scenarios that lie along performance boundaries are also the most sensitive to changes in the system; thus, they are useful for determining the performance regression between software versions.

Given the goal of discovering performance boundaries, we can reduce the total number of runs required by tailoring our scenario generation techniques to preferentially sample in places where performance boundaries are predicted to occur. In this chapter, we introduce a novel adaptive search technique designed specifically to find performance boundaries, with a particular focus on the ability of the search technique to scale to a high number of samples and high number of dimensions. In addition, we provide a method for identifying performance boundaries in the resulting data sets through unsupervised clustering techniques.

The remainder of this chapter is organized as follows. In Section 3.2 we discuss our framework for software-in-the-loop testing. In Section 3.3 we present the problem formulation and an overview of our approach. In Section 3.4 we discuss an adaptive sampling approach for generating test scenarios and the objective function for sampling along the performance boundaries. In Section 3.5 we introduce a method for identifying performance modes and selecting test scenarios which lie upon the performance boundary. In Section 3.6 we look at the performance of other adaptive sampling techniques on synthetic test functions. In Section 3.7 we present the results of a case study where the testing methodology is applied to a realistic UUV mission. Finally, in Section 3.8 we summarize our findings and introduce outstanding issues for further research.

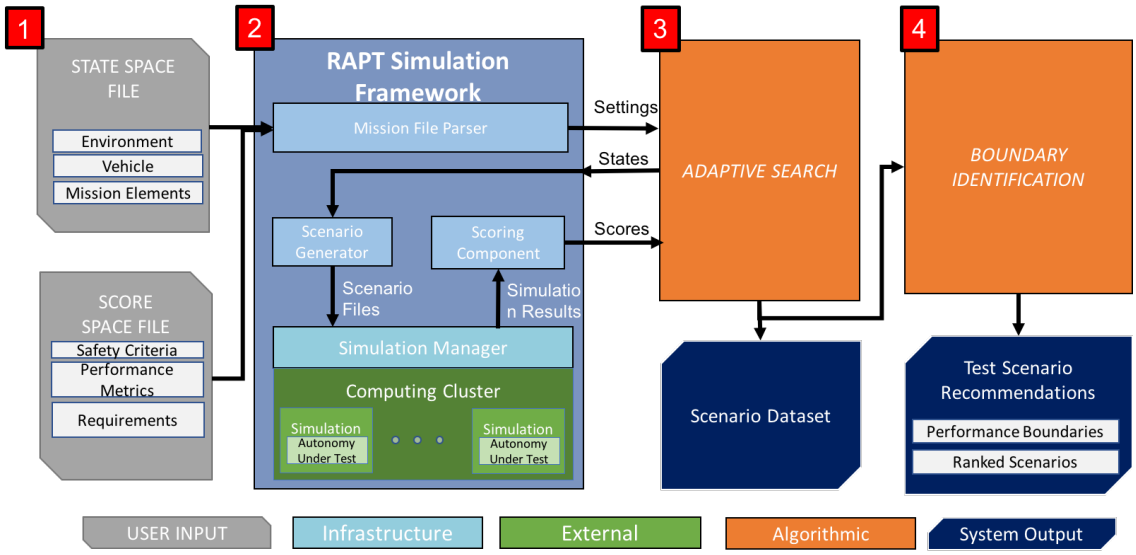


Figure 3.2: A flowchart of the Range Adversarial Planning Tool (RAPT) framework for generating test scenarios. 1) The user defines a mission and how vehicle performance should be scored. 2) The RAPT simulation framework manages the launching of runs and parsing of results. 3) Adaptive search algorithms iteratively generate new scenario states to be run in simulation based on previous results. 4) Boundary identification algorithms cluster the scenarios by performance type and rank them based on distance to performance boundaries.

3.2 Background

In this chapter, we introduce a new experimental design process for generating test scenarios for any autonomous system utilizing software-in-the-loop simulation and adaptive sampling, which we call the Range Adversarial Planning Tool (RAPT). The goal of RAPT is to assist test engineers in two regards: (1) To help them understand the decision-making process of the Autonomy Under Test (AUT) and (2) to aid in designing the final suite of tests for field testing. This is a simulation-based testing framework designed to be applicable to any autonomous system. In this section we give a brief overview of the infrastructure components illustrated in Figure 3.2. The algorithmic components are discussed in more detail in Sections 3.4 & 3.5.

3.2.1 State Space

The proposed testing process begins with a test engineer selecting which elements of the scenario will be varied during the test generation process. These elements create a parameterized scenario and their ranges constitute the testing state space, hereafter referred to simply as the state space. Examples of scenario elements used in past search-based generation techniques include obstacle configurations for ground vehicles [23], [69], sensor ranges in self-driving car applications [146], and ranges of relative bearings and distances for aircraft encounters [13], [14].

In our system, the state space is defined by a set of configuration files (known as state space files) that describe the environmental settings, mission elements, and

vehicle parameters. These settings include ranges for different simulation elements such as the time of day, the number and location of obstacles, different mission types, etc. The number of variable simulation elements in the state space constitutes its dimensionality. Individual scenarios which are passed to the simulator are created based on specific instantiations of each element within their respective state space ranges.

3.2.2 Score Space

The score space of the autonomy is defined in a similar fashion to the state space. Because the reasoning component of the AUT is treated as a black box, the test engineer must specify metrics on which the AUT is scored based on externally observable attributes. These could include binary metrics such as mission completion, discrete metrics such as the number of safety violations, or continuous metrics such as the amount of fuel consumed. While some scoring metrics may produce continuous values, these can be mapped to discrete behaviors or performance modes based on threshold values, e.g. waypoint success based upon a reaching specific distance from the waypoint.

Unlike search-based test generation techniques used in the past, this formulation does not require the user to define an objective function, which are often difficult to design and require careful tuning of the scoring parameters. The framework supports an arbitrary number of score metrics, however the larger the score space, the more performance modes that will be identified, potentially diluting a

search.

3.2.3 Simulation Framework

Our target system under test (SUT) is a simulation of the AUT performing the mission described in the state space files. It takes scenario states from the test-generation software as an input and converts those into scenario files that can be read by the simulator. An external job scheduler manages the transfer of scenario files, launching of simulations on a computing cluster, and retrieval of results from completed runs. These jobs are submitted in batches tailored to the size and speed of the cluster. After the simulation is complete the results are scored and returned to the test-generation software.

3.2.4 Recommended Test Suite

Once all submitted scenarios have been run, the performance modes are identified and the test scenarios are ranked based on their distance from the performance boundaries. In addition we return sets of boundary pairs which represent different types of performance transitions. For example, one set may contain examples of the AUT on the boundary between completing and failing its mission, while another boundary set may contain examples of the AUT on the boundary between successfully returning home and running out of battery. Each pair of scenarios across the boundary has minimal state differences in the scenario, thus providing clues as to the relevant features that instigated the transition in performance modes.

3.3 Problem Formulation

What differentiates our process from previous work is the concept of performance boundaries. As defined earlier, performance boundaries are regions of the testing space where the performance of the AUT is uncertain, i.e. small alterations to the scenario configuration can cause transitions in the AUT behaviors which result in large performance changes. This section first defines terms that are used throughout the remainder of the chapter and then provides an overview of the problem approach.

3.3.1 Definition of the SUT

(i.) The scenario configuration state space $\mathcal{X}^n = [\mathcal{X}_1, ..., \mathcal{X}_n]$ of n elements. Each element in the state space vector represents a variable in the environment, mission, or vehicle parameters with a range of possible values (obstacle positions, time windows, mission priorities, etc.). The state space in this context is synonymous with the testing space, i.e. the space of all possible tests that could be performed based on the parameters specified by the test engineer.

(ii.) A scenario input state is defined as the vector $X = [x_1, x_2, ..x_n]$ where $\forall i \in n : x_i \in \mathcal{X}_i$. The scenario is a specific instantiation of each parameter from their corresponding state space range. Thus, the state space consists of all the possible scenario configurations that could be tested. A sample set of N scenarios states is defined as $X^N = [X_1, .., X_N]$. The normalized state vector

where each $\bar{x}_i \in [0,1]$ is defined as $\bar{X}$.

(iii.) The performance score space $\mathcal{Y}^m$ of m parameters where each output score is defined as the vector $Y = [y_1, y_2, ..y_m]$. Each element in the score vector represents a performance metric by which the autonomy is evaluated, such as percentage of fuel consumed or number of waypoints reached. A sample set of N score vectors is defined as $Y^N = [Y_1, .., Y_N]$. The normalized score vector where each $\bar{y}_i \in [0,1]$ is defined as $\bar{Y}$.

(iv.) A black box system under test (SUT) function $\mathcal{F}(X^N) = Y^N$. It accepts a set of N input states $X^N = [X_1, ..., X_N]$ and returns sample set of N score vectors $Y^N = [Y_1, ..., Y_N]$. For our purposes this providing a scenario configuration as input, running the simulation until completion, and receiving the scoring metrics against the history of the simulation as output.

(v.) A performance mode is defined as $\mathcal{P} \subset Y^m$ where $\cup_i \mathcal{P}_i = Y^m$ and $\forall i \neq j, \mathcal{P}_i \cap \mathcal{P}_j = \emptyset$. In other words a performance mode is a category of scores which represent a distinct type of performance for the system under test.

(vi.) The boundary region $B_{a,b} \subset \mathcal{X}$ between performance modes $\mathcal{P}_a$ and $\mathcal{P}_b$ is defined as the region where $\forall X_{i,a} \in B_{a,b}$, $\exists X_{j,b} \in B_{a,b}$ s.t. $|X_{i,a} - X_{j,a}| < D_\epsilon$ and vice versa. Where the D_ϵ is the width of the boundary region and set of all boundaries that exist for the SUT in question is referred to as $\mathcal{B}$

(vii.) A boundary pair $b_{ij} \in B_{a,b}$ is a set of two samples each of which is the other's closest neighbor in a difference performance mode. It is defined as $b_{i,j} =$

$[X_i, X_j, Y_i, Y_j]$ where $|X_i - X_j| = D_{ij} < D_\epsilon$, $X_i, X_j \in X^N$, and $Y_i \in \mathcal{P}_a$,$Y_j \in \mathcal{P}_b | a \neq b$.

(viii.) The sampled boundary region is defined as $S_{a,b}(X^N, D_\epsilon) \subset B_{a,b}$ where $\forall X_i \in S_{a,b}(X^N, D_\epsilon)$, $\exists X_j \in X^N$ such that $|X_i - X_j| < D_\epsilon$ and $X_j \in B_{a,b}$.

3.3.2 Problem Statement

3.3.2.1 Search Problem

Given a SUT function along with the state space and score space which define its inputs the search function is defined as follows:

$$\Gamma(\mathcal{F}, \mathcal{X}^n, \mathcal{Y}^m, N) = \mathcal{L}^N. \tag{3.1}$$

Where N is the number of samples allocated to the search. The output, $\mathcal{L}^N$, is a set of labeled samples $\mathcal{L}^N = [X^N, Y^N]$ consisting of the queried states X^N and their respective scores Y^N.

Our objective is to generate the set of samples X^N which maximizes the volume of the sampled boundary regions $S_{a,b}(X^N, D_\epsilon)$ for all boundaries in $\mathcal{B}$ for the smallest possible value of D_ϵ. This region is illustrated in Figure 3.3.

The number of performance modes of the SUT and the mapping from score to performance mode are not known a priori.

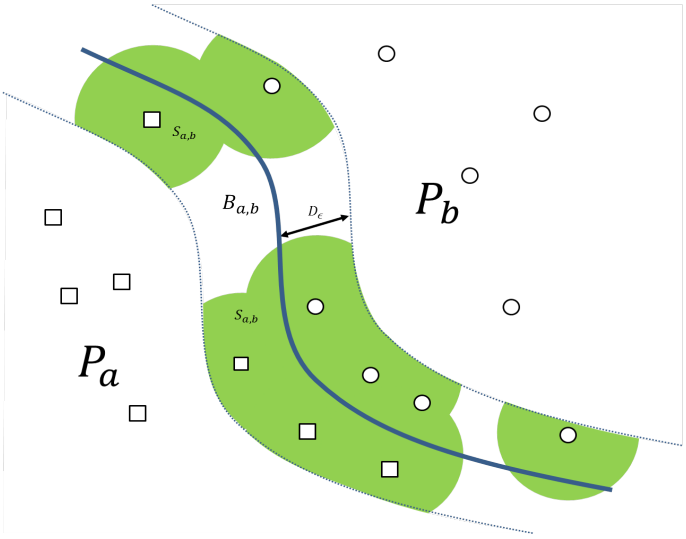


Figure 3.3: A performance boundary between modes $\mathcal{P}_a$ and $\mathcal{P}_b$ is shown in the solid line with the boundary region with width D_ϵ shown in the dashed lines. Samples from the set X^N are illustrated with either a square or circle depending on their performance mode. The sampled region $S(X^N, D_\epsilon)$ is shaded green.

3.3.2.2 Boundary Identification Problem

We formally define the boundary identification algorithm as a function

$$\mathcal{C}(\mathcal{L}) = \mathcal{B} \tag{3.2}$$

which accepts a set of labeled samples, $\mathcal{L}^N$, and returns the set of identified performance boundaries:

$$\mathcal{B} = [B_{1,2}, B_{1,3}, ..., B_{L-2,L}, B_{L-1,L}] \tag{3.3}$$

where L is the number of identified performance modes and N is the number of samples in $\mathcal{L}^N$. Each boundary $B_{a,b}$ is the set of samples that borders the performance modes a and b.

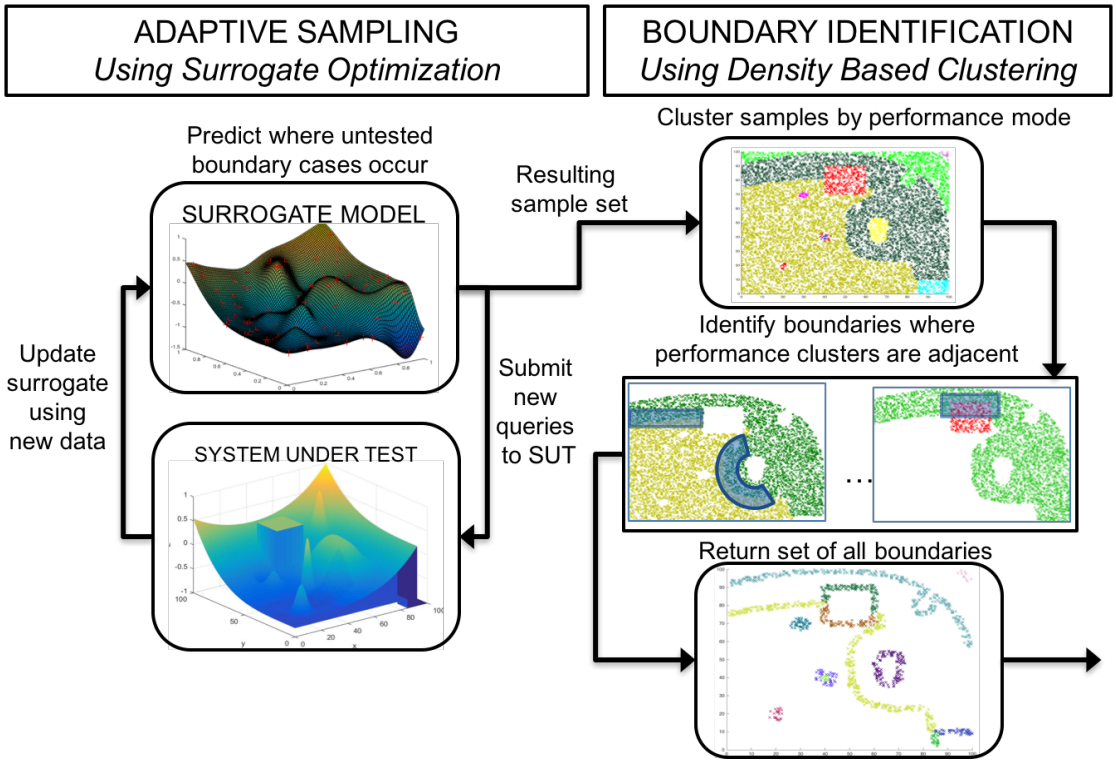


Figure 3.4: An overview of the adaptive sampling and boundary identification process.

Our objective is to successfully identify all samples in $\mathcal{L}^N$ which exist on the boundaries between performance modes and provide an estimate of their distance from the boundary.

3.3.3 Overview of the Algorithmic Approach

The scenario configurations we consider to be the most informative are those which occur in the transition regions between performance modes, previously referred to as the performance boundaries. The reasoning behind this claim is that it is ineffective to test the system in regions of the state space where performance is constant and known, i.e. scenarios where the system will either almost surely succeed or almost surely fail. Much more information about the system is gained

by testing in regions where critical decisions must be made by the autonomy that result in variable performance. Additionally, the traditional strategy of testing under worst-case conditions does not fully characterize the performance envelope of the system; there may be failure modes or performance boundaries that occur in regions other than worst-case conditions that are not immediately apparent. Given a user-defined state space, $\mathcal{X}^n$, and a limited number of queries, N, to the autonomy simulation, our objective is to find the performance boundaries of the system. As performance boundaries are where small changes in the state cause a large change in the score this can also be conceptualized as large gradients or discontinuities in autonomy performance.

To achieve this goal, the approach presented in this chapter is broken into two primary phases: search and identification. This cycle for test generation is illustrated in figure 3.4. During the search phase we utilize an adaptive sampling or active learning approach to select new test scenarios that are run by the autonomy simulation. In the nature of adaptive sampling, these new test scenarios are selected based on the performance score of the autonomy from previous simulations. We use a new modular adaptive sampling strategy to model the autonomy performance and preferentially select regions that might indicate performance boundaries. The high dimensionality of any realistic state space for an autonomy under test makes it intractable to simply perform an exhaustive spread of simulations. Thus, we have focused our problem of searching the state space primarily on adequate coverage of the boundary regions while minimizing the number of simulations.

In the identification phase, the samples generated during the search phase are

used to identify the performance modes in the resulting data using unsupervised clustering algorithms. Once test cases have been classified by their performance mode, the boundaries between performance modes are identified and the tested scenarios adjacent to boundaries can be used to aid in live test design.

3.4 Search Strategy

3.4.1 Search Problem

As discussed previously, the goal of the search algorithm is to create the highest quality set of test scenarios given the allocated number of simulations. This involves creating both an informative and diverse set; the search algorithm must choose samples in areas that indicate the presence of a performance boundary while also preventing oversampling by continuing to explore the state space with samples in untested regions.

Unlike prior works on scenario generation [14], [23], [68] which utilize global optimization techniques such as genetic algorithms, we take an adaptive sampling approach. We do so for two important reasons. The first is that the objective of this dissertation is to fully discover and characterize all possible performance boundaries, not just the most extreme ones that many multimodal optimization techniques would produce. The second is that optimization objective functions are notoriously difficult to design and are typically system dependent. Thus, we focus on exploiting underlying features of the performance surface to discover the regions of interest, allowing for a more general approach that does not require domain-specific

knowledge.

3.4.2 Adaptive Sampling

Adaptive sampling is an iterative process consisting of submitting queries to the SUT, using the returned scores to generate a meta-model, and then applying an information metric to the meta-model to generate a new set of queries. This is an alternative to space-filling designs, such as Latin-Hypercube or Sobol sequences, which attempt to optimize uniform coverage and density and are precomputed based upon the size of the state space and the available number of queries. In this chapter we utilize a generalized framework for adaptive sampling which allows for changing the underlying meta-models and information metric. This is more formally defined in Algorithm 1. The adaptive sampling algorithm uses the normalized unit states $\bar{X}$ and scores $\bar{Y}$ for the information metrics.

There are multiple query strategies that can be used for adaptive sampling including entropy, model improvement, uncertainty, and density. However, all these strategies were developed for the purposes of maximizing the accuracy of the underlying meta-model, whereas our objective is to generate samples that exist near performance boundaries. Thus, we have designed our metrics to look for areas with high gradients that have not yet been sampled. This is similar to the exploration-exploitation approach of the LOLA-Voronoi algorithm [80] and as such we have included it as one of our baseline comparisons. The Voronoi tesselation present in LOLA-Voronoi, however, scales poorly with both the number of samples and input

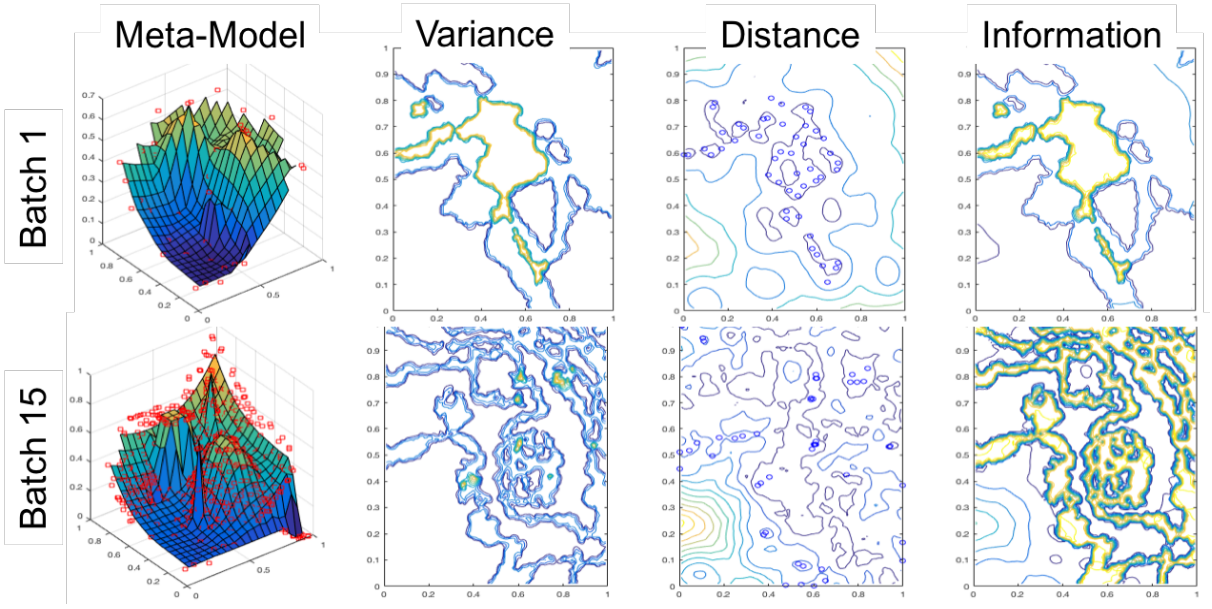


Figure 3.5: Plots showing the evolution of the information metrics as samples are collected for the Custom2D synthetic function. From left to right are the Gaussian Process meta-model, contour plots show the NNDE and NNVE values σ_K and d_K, and a contour plot of the information M_{NNDV}. The model in Batch 1 is only trained on 50 samples, leading large areas of similar information. The model after Batch 15 has collected 550 samples, leading to more sharply defined predictions of the boundary regions.

dimensionality. For n points in $\mathbb{R}^d$, it takes $O(nlogn + n^{[d/2]})$ computational time, making it infeasible for higher dimensional problems.

We introduce two new meta-model metrics for the purpose of discovering performance boundaries: one which uses a Gaussian Process Regression (GPR) meta-model and one which uses a k-nearest neighbor technique for density and variance estimation. As the Gaussian process scales with $O(n^3)$ and the k-nearest neighbors algorithm which scales with $O(knlogn)$, we believe these can offer better scaling as the number of dimensions and the required number of samples increases. These meta-model evaluators are defined as $\mathcal{M}(X)$ - they take existing samples as inputs and return the expected information gain of a proposed query as an output.

The GPR meta-model uses a zero mean function and a Matern-covariance function [88] with $nu = d/2$ and isotropic distance measure. Given a proposed query it returns the mean value μ, the first-order gradient of the mean $\bigtriangledown\mu$, and the variance of the query σ. The Matern-covariance is proportional to the distance to the nearest sample; thus, variance in this case makes it an appropriate reflection of how far away the query is from one of the training samples. The GPR meta-model evaluator uses the magnitude of the gradient and uncertainty as follows:

$$\mathcal{M}_{GPR}(\bar{X}) = (|\bigtriangledown \mu(\bar{X})|)^g \cdot (\sigma(\bar{X}))^v \tag{3.4}$$

where g and v are tuning parameters to balance exploration of high uncertainty regions with high gradient regions.

The Nearest Neighbor Density and Variance (NNDV) evaluator estimates the local properties of a query using its nearest neighbors. We utilize a k-nearest neighbors density estimate [147] and variance estimate [148] to obtain the predicted variance σ_K of the sample and its mean distance d_K to its neighbors. The information is then computed as follows:

$$\mathcal{M}_{NNDV}(\bar{X}) = (\sigma_K(\bar{X}))^g \cdot (d_K(\bar{X}))^v \tag{3.5}$$

where g and v are the same tuning parameters used in the GPR meta-model evaluator. The evolution of this information metric is illustrated in Figure 3.5.

When dealing with systems which have categorical scores, we need to choose

a different variance measure for our information function. For these systems we use the "unalikeability" measure u described in [149]. This metric is 1 when every element of the set is from a different category and 0 when all elements of a set are members of the same category. It is defined as follows,

$$u = \frac{\Sigma_{i \neq j} c(x_i, x_j)}{n^2 - n} \tag{3.6}$$

where

$$c(x_i, x_j) = \{ \begin{matrix} 1, x_i \neq x_j \\ \\ 0, x_i = x_j \end{matrix} \tag{3.7}$$

In typical adaptive sampling fashion, the meta-model evaluators are used to select the subsequent batch of samples based on the set of queries with the highest expected information gain, as outlined in Algorithm 1. Currently our methods re-train the meta-model evaluator at every iteration. This brings the computational complexity for the entire search process is $O(\frac{n^4}{L})$ for the GPR meta-evaluator search and $O(k\frac{n^2}{L}logn)$ for the NNDV meta-evaluator search where L is the number of samples in each batch. This could be improved in future implementations by utilizing meta-models which can be iteratively updated with new data instead of being trained from scratch after each iteration.

3.4.3 Sensitivity Scaling

When applying the search techniques to a realistic SUT with a user-defined state space, there may be a high number of input states and output scores. Addi-

Algorithm 1 AdaptiveSearch($SUT, \mathcal{X}^n, \mathcal{M}, N$)

Input: A function representing the system under test $\mathcal{F}$, a scenario state space $\mathcal{X}^n$, a meta-model evaluator $\mathcal{M}$, and a desired number of samples N
Output: A set of labeled samples $\mathcal{L}$
Select a query batch size of L and an initial batch of randomly selected query states X_0^L. In addition, choose a number of proposed queries, p, to perform per iteration.
for all $i \in [0, N/L]$ **do**
$\mathcal{F}(X_i^L) = Y_i^L$
concatenate($\mathcal{L}, [X_i^L, Y_i^L]$)
Train $\mathcal{M}$ on labeled sample set $\mathcal{L}$
Randomly select a new set of proposed queries $X^p : p > L$
$X_{i+1}^L = argmax_{X^L \subset X^p} \mathcal{M}(\bar{X}^L)$
end for
return $\mathcal{L}$

tionally, the state variables which actually contribute to the output of the system may be not known *a priori*. Reducing the range and dimensionality of the state space can result in a much more efficient and effective search. We can achieve this by applying sensitivity analysis techniques that search only over the state variables which contribute to the system output and treat the remainder as independent noise. This is done by scaling the range of each state based on its input sensitivity. Thus, states with little importance appear to be identical with regard to the distance metric while the ranges of highly influential states are magnified to provide a more focused search.

In this work, state sensitivities are determined by fitting a classification tree to the data and computing variable importance $VI(x)$ as described in [150]. The computational complexity of training a classification tree is $O(mnlogn)$ where m is the number of input features. Thus, while it is a non-trivial calculation, it does not change the overall complexity of the search algorithm.

The sensitivity-scaling information metric is given by

$$\mathcal{M}_{scaled}(\bar{X}) = \frac{1}{M}\sum_{i=0}^{M}\mathcal{M}_{NNDV}(\bar{X} \circ VI_i(\bar{X})) \tag{3.8}$$

where $VI_i(\hat{X})$ is the vector of variable importance measures for the classification tree trained on output $\bar{Y_i}$, and M is the number of score outputs. Moving forward we refer the scaled states as $\hat{X} = \bar{X} \circ VI_i(\bar{X})$ The effects of applying sensitivity scaling are discussed in 3.6.3. We refer to this variant as the Scaled Neighborhood Density and Variance (S-NDV) evaluator.

3.5 Boundary Identification

3.5.1 Scenario selection Problem

The data set of simulation results generated during the search phase can easily approach hundreds of thousands to millions of runs. Analyzing this data is not a trivial task - there may be thousands of examples of autonomy behavior that need to be diagnosed. By clustering scenarios with similar behaviors and identifying the boundary sets between these clusters, we provide a means for a test engineer to methodically evaluate the trending behaviors of the system.

The selection of the algorithms used in boundary identification was driven by two needs. The first is the lack of *a priori* knowledge of the number of performance modes. The second is that there are no guarantees about the shape of the performance mode clusters. These two facts preclude methods such as k-means clustering

and Gaussian mixture models from being applied. In addition, preference was given to techniques that required minimal hyperparameter tuning if given normalized data sets with similar numbers of samples.

3.5.2 Identifying Performance Modes

The nature of black box testing dictates that we cannot look inside the AUT decision engine to determine which behavior is executing. Instead, we must use externally observable metrics and infer changes in behavior from changes in the performance of the system. Our current approach is to apply unsupervised clustering techniques to identify the performance modes of the system.

In cases where the autonomy is scored using discrete values, e.g. binary criteria for mission success and safety success, it is trivial to identify distinct performance modes from the resulting scores. In these instances, the performance mode is simply the combination of all the discrete score labels. In order to apply our techniques to systems which provide continuous outputs, we utilize mean shift clustering [151] on the score space to identify the performance modes and classify the samples. Once the samples have been classified with respect to their performance mode, they are then subjected to DBSCAN clustering [152], a density-based clustering technique which groups contiguous sets of samples together. These algorithms were selected because they do not require *a priori* knowledge of the number of possible classifications or the landscape of the score space. If the hyperparameters are scaled appropriately according to the state space and score space, they provide an efficient means of

identifying performance modes from continuous outputs.

Once samples have been classified by performance mode, the boundaries are composed by performing a pair-wise comparison between every performance mode with a differing performance mode. We utilize a k-nearest neighbor detection algorithm to determine the closest neighbor in a differing performance mode for each sample. Any samples that are within D_ϵ distance of their nearest neighbor in the differing performance mode are added to the final boundary set, i.e. $D_{ij} < D_\epsilon$. The final boundary set is then constructed from boundary pairs defined as

$$B_{a,b} = [b_{(a,b),1}, ..., b_{(a,b),k}] \tag{3.9}$$

where a and b signify performance modes $\mathcal{P}_a$ and $\mathcal{P}_b$, respectively. The boundary pairs $b_{(a,b),i}$ are composed of points in the sampled set $\mathcal{L}^N$ and satisfy:

$$b_{(a,b),i} : Y_{i1} \in P_a, Y_{i2} \in P_b, |X_{i1} - X_{i2}| \leq D_\epsilon \tag{3.10}$$

This approach is defined further in Algorithm 2.

3.5.3 Boundary Scaling

Similar to Section 3.4.3, high-dimensional system dictate the use of variable importance scaling during the k-nearest neighbors search and DBSCAN steps of the boundary identification process. This involves computing $\hat{X}^N$ as described in Section 3.4.3, utilizing $\hat{X}_Y$ in place of $\bar{X}_Y$ during the DBSCAN clustering, and finally

Algorithm 2 BOUNDARYIDENTIFICATION($\mathcal{L}$)

Input: A set N of labeled samples $\mathcal{L}$ containing the input states X^N and output scores Y^N
Output: A set of identified performance modes, a collection of boundaries $\mathcal{B}$, and distance estimate vector D

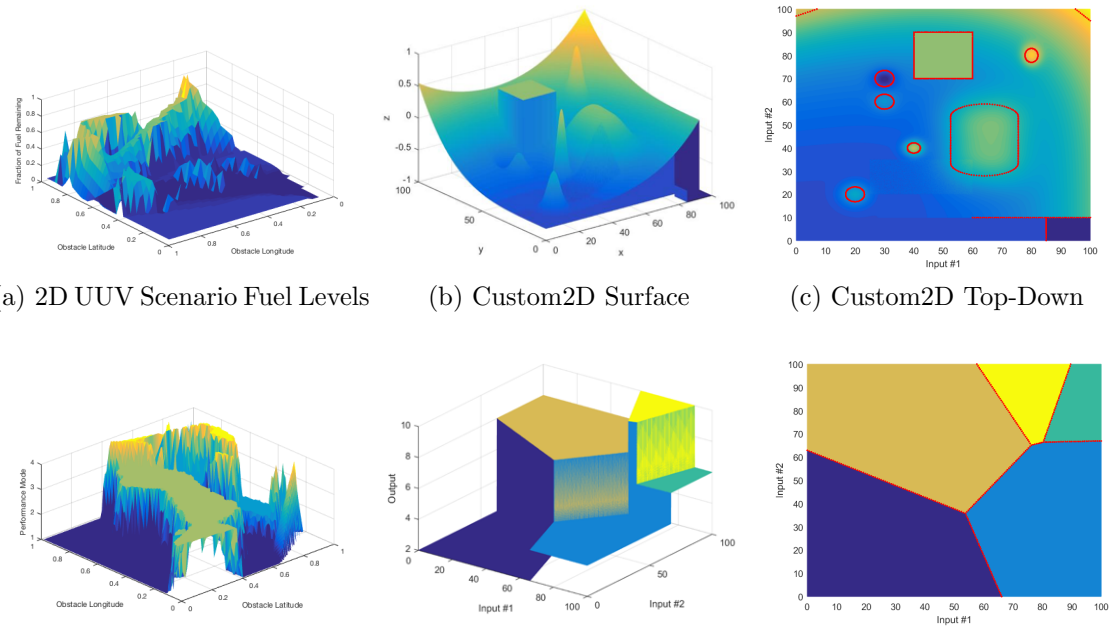
Let λ_P be the threshold distance for the flat kernel mean shift function, ϵ_C and n_{min} be the radius and minimum member parameters for the DBSCAN function. Let D_ϵ be the maximum distance between two samples to be considered part of a boundary.
$\mathcal{P} = MeanShift(Y^N, \lambda_P)$, identify the performance modes
for all $P_l \in \mathcal{P}$ **do**
  Create the set of all states belonging to that performance mode
  $X_{P_L} = X_i | Y_i \in P_l$
  Append the new cluster of states $C_Y = [X_{P_L}, Y]$ to the list of existing clusters
  $C \frown [C_Y]$
end for
for all $C_Y \in C$ **do**
  Create a set of subclusters for the regions of interest using the DBSCAN algorithm
  $\hat{C_Y} = DBSCAN(\bar{X_{P_L}}, \epsilon_C, n_{min})$
  Append the subclusters to the complete set of clusters
  $\hat{C} \frown [\hat{C_Y}]$
end for
for all $\hat{C_Y}_i$ and $\hat{C_Y}_j \in \hat{C} | Y_i \neq Y_j$ **do**
  $D_{ij} = knnsearch(\bar{X_{P_i}}, \bar{X_{P_j}})$
  $\mathcal{B}_{ij} = [X_{P_i}, X_{P_j}, Y_i, Y_j] \forall X_{P_i}, X_{P_j} | D_{ij} < D_\epsilon$
end for
return $\mathcal{B}$

using $\hat{X}_{Yi}$ and $\hat{X}_{Yj}$ during the k-nearest neighbors search. The effects of applying this scaling on the final boundary pairs are explored further in the case study of Section 3.7.

3.5.4 Boundary Threshold Criteria

A reasonably complex scenario could contain several dozen input parameters. This means we will likely only be achieving sparse coverage of the state space even after applying the adaptive search approach. Special consideration of the distance threshold, D_ϵ, must then be given to account for changes in number of dimensions, number of samples, and the number of expected performance modes.

As such, we have added an option to allow for a scaled threshold criteria based upon the distribution of estimated boundary distances for the entire data set. Therefore, for systems where the true boundary is unknown, we replace the metric



(a) 2D UUV Scenario Fuel Levels (b) Custom2D Surface (c) Custom2D Top-Down

(d) 2D UUV Scenario Performance Modes (e) Plates2D Surface (f) Plates2D Top-Down

Figure 3.6: Comparison of the synthetic test functions (b)(e) with results from the UUV scenario (a)(d) presented in Figure 1. The red lines in the top-down views (c)(f) represent where the true boundary locations are for the synthetic functions.

$D_{ij} < D_\epsilon$ with a quantile threshold, where the boundary pairs $b_{(a,b),i}$ are composed of points in the sampled set $\mathcal{L}^N$ and satisfy:

$$b_{(a,b),i} : Y_{i1} \in P_a, Y_{i2} \in P_b, Pr[D^K < D_i] \leq q_B \tag{3.11}$$

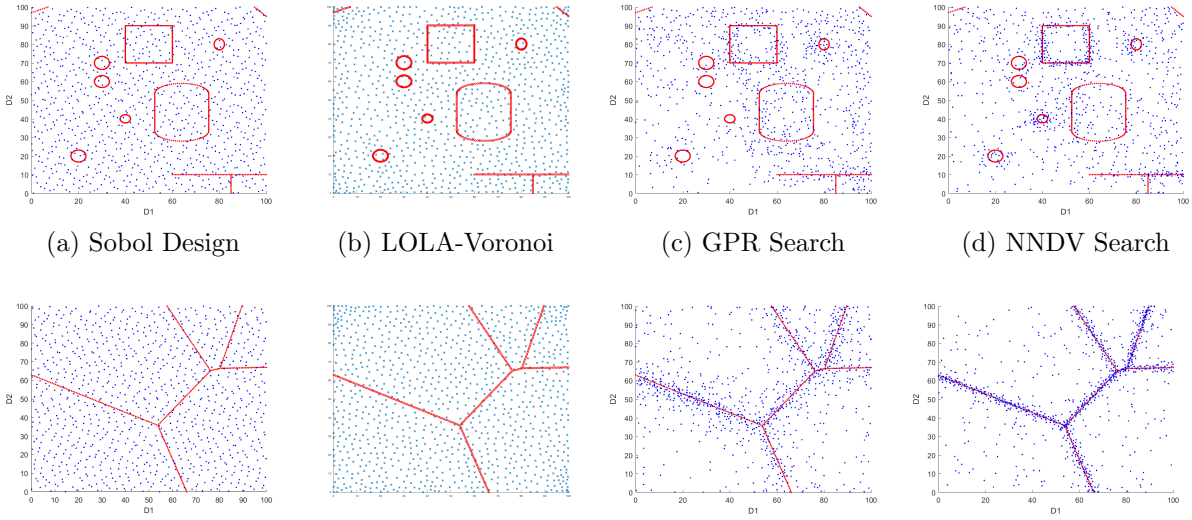
Where q_B is the quantile threshold. D^K is the estimated boundary distance for every boundary pair in $B_{a,b}$, and $D_i = |X_{i1} - X_{i2}|$ is the distance between the states of pair $b_{(a,b),i}$. For the results of this chapter, we utilize the 20th percentile as our quantile threshold.

3.6 Results

3.6.1 Test Systems

Several candidate systems were developed to evaluate the adaptive search and boundary identification algorithms. The first category of candidate systems was comprised of mathematical test functions with performance boundaries that were known *a priori*. The second category consisted of a simple unmanned undersea vehicle (UUV) scenario, presented in Section 3.7.

Three synthetic test functions were developed in order to evaluate the algorithms against a known mathematical surface. The intention in designing custom test functions was to mimic the wide variety of features and boundaries that may be present in an autonomous system's performance landscape. The three functions are as follows:



(a) Sobol Design (b) LOLA-Voronoi (c) GPR Search (d) NNDV Search

(e) Sobol Design (f) LOLA-Voronoi (g) GPR Search (h) NNDV Search

Figure 3.7: Scatter plots of the different sampling techniques on the Custom2D (top row) and Plates2D (bottom row) test functions. The 1000 samples taken are in blue and the true locations of the boundaries are in red.

- Custom 2D - Two input dimensions with one continuous unlabeled output. It contains peaks, valleys, plateaus and cliffs as features of interest (Figure 3.6(b)(c)).

- Plates 2D - Two input dimensions with one discrete output. There are 5 score categories, i.e. representative performance modes (Figure 3.6(e)(f)).

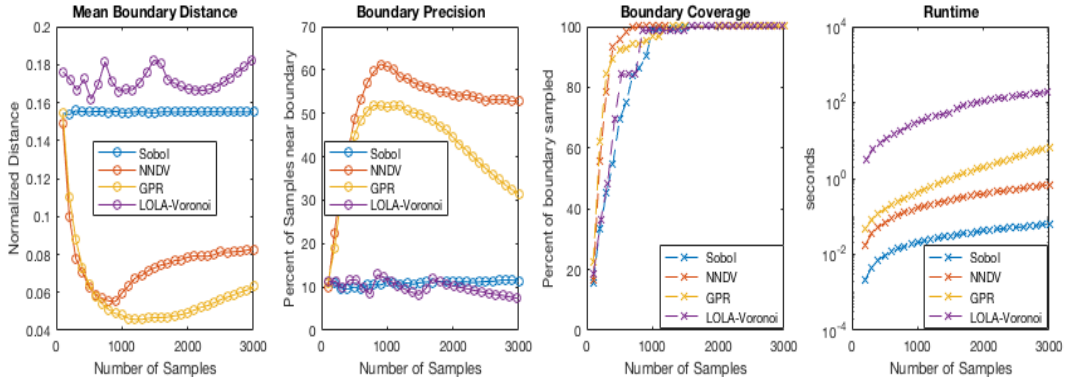
- Plates 3D - Three input dimensions with one discrete output. There are 5 score categories, i.e. representative performance modes.

These low-dimensional test functions have the advantage that they are easy to visualize and have performance boundaries that were known *a priori*. The performance boundaries were defined as the local maxima of the first derivative of each respective test function.

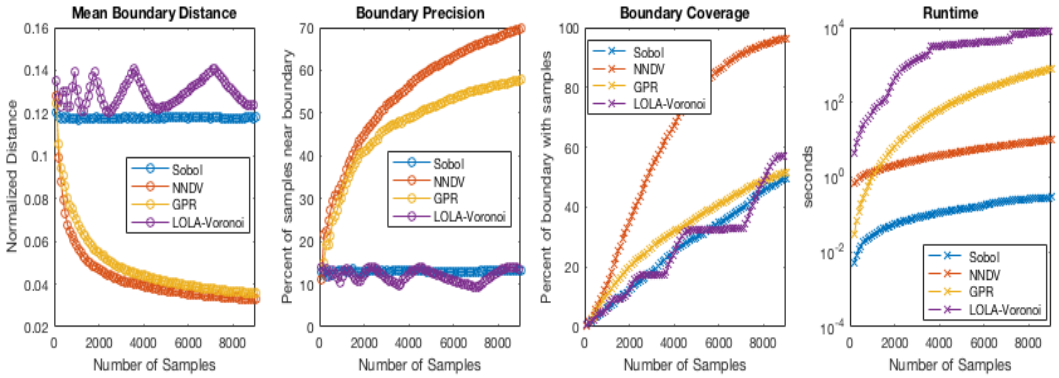
3.6.2 Search Performance on Synthetic Functions

We evaluated the performance of the search algorithms presented in Section 3.4 according to the metrics of Section 5.3.2, i.e. based on their ability to identify features in the test functions and sample near the performance boundaries. For comparison, we chose a Sobol sequence design as the baseline space-filling sampling approach. To compare against the current state-of-the-art in adaptive sampling, the LOLA-Voronoi sequential design method with a Blind Kriging model was also included for comparison. The LOLA-Voronoi code is accessible using the SUMO software toolbox [153] in MATLAB. We compared these methods against our adaptive search algorithms using both the GPR-based and NNDV information functions. We use the following metrics for each of the mathematical test functions: precision, coverage, convergence, and runtime. The results of these tests are shown in Figure 3.8 and summarized in Table 3.1.

The search methods introduced in this chapter outperformed both the space-filling approaches as well as the popular LOLA-Voronoi adaptive search in all of the evaluation metrics with the exception of runtime. This is particularly true in cases where the boundaries are sharply defined, as in the Plates2D test function. As shown in Figure 3.7, the GPR-based search concentrated nearly all of its samples in the regions near the boundaries with minimal cases selected in the uninteresting regions of small gradient. More importantly, it also managed to obtain near full coverage of the boundaries in under half the samples of the Sobol space-filling method. The results are even more pronounced for the NNDV search algorithm, with the added



(a) Plates 2D



(b) Plates 3D

Figure 3.8: Convergence plots of mean distance, precision, and coverage for the Plates2D (above) and Plates3D (below) functions

benefit of shorter runtime as well.

An interesting result of the two-dimensional system is that precision begins to worsen after the coverage reaches 100%, indicating that the search techniques saturate the boundary regions and begin exploring the rest of the space. In Figure 3.8, it can be seen that the GPR obtains samples closer to the boundary but sacrifices precision and coverage. When comparing the 2D test functions with the Plates3D, it becomes apparent that the added dimension greatly increases the number of cases necessary to obtain coverage of the boundaries. In two dimensions, the NNDV search converges in under 1000 samples. However, in three dimensions, more than 10,000 samples are required to reach 90% coverage. The other search methods, meanwhile, only achieve half of this coverage.

For the given number of samples, the LOLA-Voronoi search did not distinguish itself significantly from a space-filling design. One interesting feature of this method is the periodic effect of increasing and decreasing precision apparent in the convergence plots. This indicates that the LOLA-Voronoi algorithm has distinct phases of exploiting the existing model and searching in areas of high gradient vs. exploration where it tries to spread out its samples as much as possible. This is likely due to the fact that it was originally designed to minimize global model-fitting error. The techniques proposed in this chapter have the different objective of searching for performance boundary regions, resulting in sample sets that concentrate on high-gradient regions. Despite the superficial similarities in approach, the problem of identifying boundary regions in an unknown landscape is something that traditional adaptive sampling techniques are not tailored to achieve.

3.6.3 Effects of Sensitivity Scaling

A simple comparison of the search performance between the 2D and 3D test functions illustrates the challenge posed by higher dimensional landscapes. As introduced in Section 3.4.3, this necessitates the need to scale the state space based on the sensitivity of each state input. Scaling the input states based on sensitivity reduces the effects that non-contributing variables (i.e. variables that act as noise) have on the search process. To evaluate whether sensitivity scaling had the desired effect, two types of approaches were tested: input screening for removing non-contributing variables and variable separation for dealing with multiple outputs with disjoint inputs.

3.6.3.1 Variable Screening

We evaluated the NNDV and S-NDV search strategies against a variant of the Plates3D synthetic function where additional non-contributing inputs were artificially added. For example, if four non-contributing variables were added, the resulting function would be a 7D system, where three of the inputs contributed to the function output and four of the inputs acted as noise. The GPR-based search technique was not included in this analysis because it becomes computationally infeasible in high dimensions and large datasets. Each search was run for 10,000 samples and evaluated for precision and coverage. The results of applying the search technique on systems with varying numbers of non-contributing inputs (between 0 to 10) can be seen in Figure 3.9. As more non-contributing input states are added, the

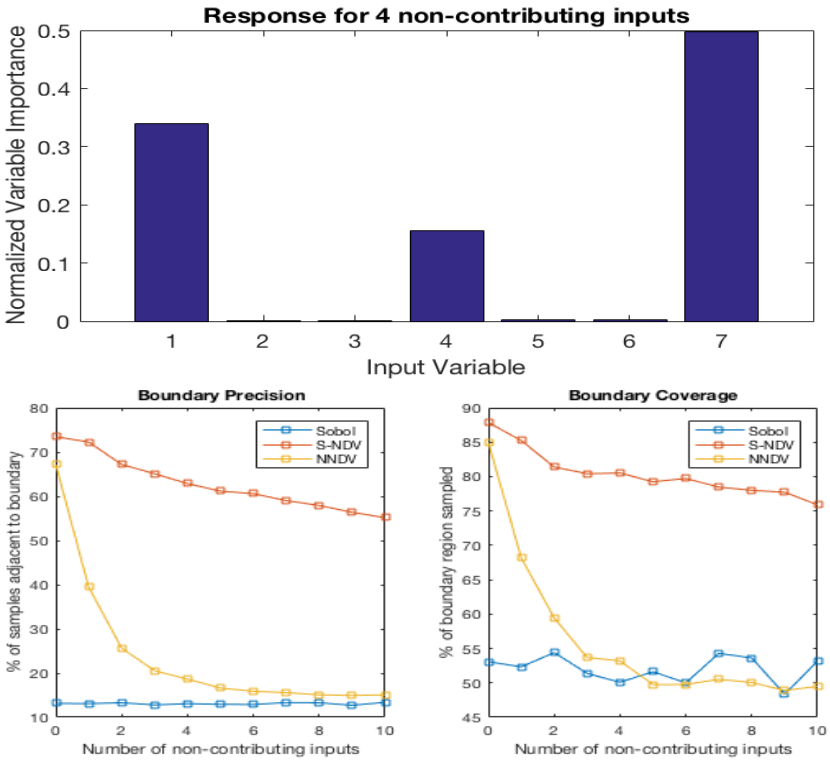


Figure 3.9: Variable importance plot (top) showing the variable senstivity for each input of the 7D input system. Convergence plots (bottom) showing the performance of each search method as the number of non-contributing variables increases.

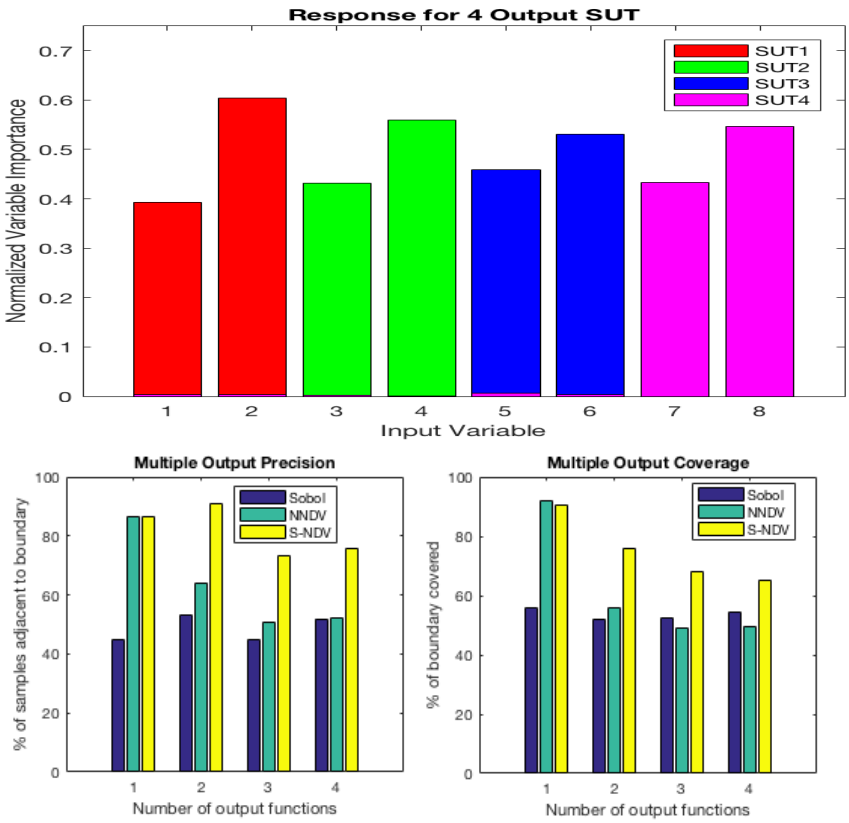


Figure 3.10: The variable importance for each of the 4 outputs of the 8 input system correctly reveals the relationship between the input and output variables (above). The effect on precision and coverage for each of the techniques as number of subfunctions searched simultaneously increases (below).

NNDV search quickly converges to having similar performance as the space-filling Sobol design. The S-NDV search suffers a slight decrease in precision (approx. 15%), however, this is minor when compared to the decrease for the NNDV search (approx 90%). This clearly demonstrates the benefits of using variable screening for sensitivity-based scaling of the state space. It is possible that performance could be further improved by completely eliminating non-contributing variables from the search rather than simply scaling their input range. This extension is being considered for future iterations of the algorithm.

3.6.3.2 Variable Separation

In addition to variable screening, the S-NDV algorithm was tested on systems where all input variables were utilized, however, each input dimension may have contributed to different output variables. This was done by creating a composite multiple-input, multiple-output function of several 2D synthetic test functions. The final system was defined as

$$f_{composite}(x) = [f_1(x_1, x_2), f_2(x_3, x_4), f_3(x_5, x_6), f_4(x_7, x_8)] \tag{3.12}$$

which results in an 8D input, 4D output function. The search process on the composite function was executed in four different ways: (1) with each subfunction searched independently (i.e. 1 output per search), (2) with pairs of subfunctions searched simultaneously (i.e. 2 outputs per search), (3) with triplets of subfunctions searched simultaneously, and (4) with all four subfunctions searched simultaneously. The coverage and precision was computed for each subfunction and the mean for all subfunctions is shown in the charts in Figure 3.10.

The results are similar to those seen in the previous section. The number of variables searched simultaneously has no effect on the Sobol design but severely impacts the NNDV search. While some degradation is seen in the performance of the S-NDV it still retains improved performance over a space-filling design.

3.7 UUV Case Study

3.7.1 UUV Mission

As stated previously, the methods of this chapter can be applied to any black box SUT. In order to demonstrate the ability of our system to discover performance boundaries in a real system, a case study was performed where the process was applied to an autonomous UUV software executing a complex mission in a medium fidelity simulation environment. The state space was designed with input from test engineers at the Naval Undersea Warfare Center Division Keyport to represent a realistic testing scenario for a medium-sized UUV.

The UUV was required to complete multiple mission objectives, as well as comply with all safety criteria. The mission objectives were to follow a set of predetermined waypoints, perform station-keeping in a set of three prioritized mission areas for a given amount of time, and reach a transmission area within a specified time window. A mission time criteria was also included such that the UUV must complete all of the mission objectives within an overall mission time window. The safety objectives were to avoid all obstacles, remain inside of the operational area, remain outside of a no-go region, and to return home to a recovery point. Additionally, tidal factors were considered by including the mission start time as a variable parameter. All elements of the mission were known *a priori* by the autonomy except for the positions of the obstacles. The mission elements are defined in more detail in Table 3.2 and illustrated in Figure 3.11.

The autonomy performance of a scenario was scored based upon criteria that results in one of four classifications, i.e. performance modes. The first category of autonomy performance is a total failure (TF) - this is where the UUV fails any of the safety objectives as well as any of the mission objectives. The second category is safety success (SS), where the UUV fails any of the mission criteria but passes all of the safety criteria. Mission success (MS) is where the UUV completes all of the mission criteria but fails any of the safety criteria. Finally, total success (TS) is where the UUV completes all of the mission criteria and all of the safety criteria.

The simulation environment and autonomy were developed in-house using the Johns Hopkins APL Autonomy Toolkit (ATK) [110], [154]. In order to test the ability of the test generation process to detect performance boundaries, the autonomy was specifically designed to have suboptimal decision-making. In this way, the autonomy would produce a more uniform distribution of the different performance modes. The simulated system had a maximum speed of 3 m/s and used a 6 degree-of-freedom transit model for underwater vehicles model derived from [155]. It possessed a sonar with a 100-meter range and 120-degree field of view. The environment possessed a tidal current that varied based on the time of day between 3 m/s due northeast to 3 m/s due southwest with peaks occurring at 2am and 12pm respectively. This complex mission required multiple behavioral subcomponents. ATK computes the current priorities for each behavior at every time step and selects which ones will be executed. When multiple behaviors execute simultaneously the final desired speed and heading is a weighted sum of all the desired vectors. The behaviors are as follows.

- Obstacle Avoidance and Waypoint Navigation - A potential field control law [154] which always executes with the highest priority.
- No-Go Area Avoidance - Causes all no-go areas to be treated obstacles unless the vehicle is in emergency return mode.
- Crabbing Navigation - Adjusts the vehicle heading to cancel out the effect of the current.
- Survey - A behavior that sets the current waypoint inside the next unexplored mission area and loiters for a predetermined time once it arrives.
- Transmission - A behavior that sets the current waypoint at the surface inside the transmission area and loiters there until all data is transmitted. This takes priority over the survey behavior.
- Return behavior - Computes the amount of fuel required to reach the recovery point and returns home once the mission is complete or it determines it does not have enough fuel to complete the mission. This overrides the survey and transmission behaviors. If battery levels are critical, it will also override no-go area behaviors.

3.7.2 Experimental Setup

For our experiment, the state space consisted of 18 variable parameters: the start time (1), the transmission window start time and duration (2,3), the latitude and longitude of the transmission area center (4,5), the latitude and longitude of the

no-go region center (6,7), the latitude and longitude of the 3 minor obstacles (8-13), the latitude and longitude of the 2 barrier obstacles (14-17), and the priority order for the mission areas (18). These state space parameters gave us representative cases of input states that have a strong effect on the autonomy, such as the tidal force imposed by the start time, and input states that have a weak effect, such as the position of a minor obstacle. The start time was varied in a 12-hour period between midnight and noon, the transmission window was set to open between 30 minutes after mission start time to an hour after mission start time with a duration between 30 minutes to an hour. Additionally, the transmission area was set to only vary in position on the western half of the operational area. The no-go region was set to only vary in position in the southeastern quadrant of the operational area. The minor obstacles could be varied in position anywhere in the operational area. The barriers were constrained to vary in position only within 400 meters of the operational area vertical centerline. All permutations were included for priority order of the mission areas.

Using this state space, one million runs were submitted to a computing cluster in batches of ten thousand using both a Sobol space-filling design and the S-NDV approach. Of the submitted simulations, we were able to collect 850 thousand valid runs for the Sobol dataset and 883 thousand valid runs for the S-NDV approach. The remainder were pruned from the dataset due to parameter settings that created invalid scenarios for the autonomy (e.g. placing an obstacle over the recovery point). The samples generated from each of these searches were then passed to the boundary identification algorithm where the distance threshold, D_ϵ, was set at the

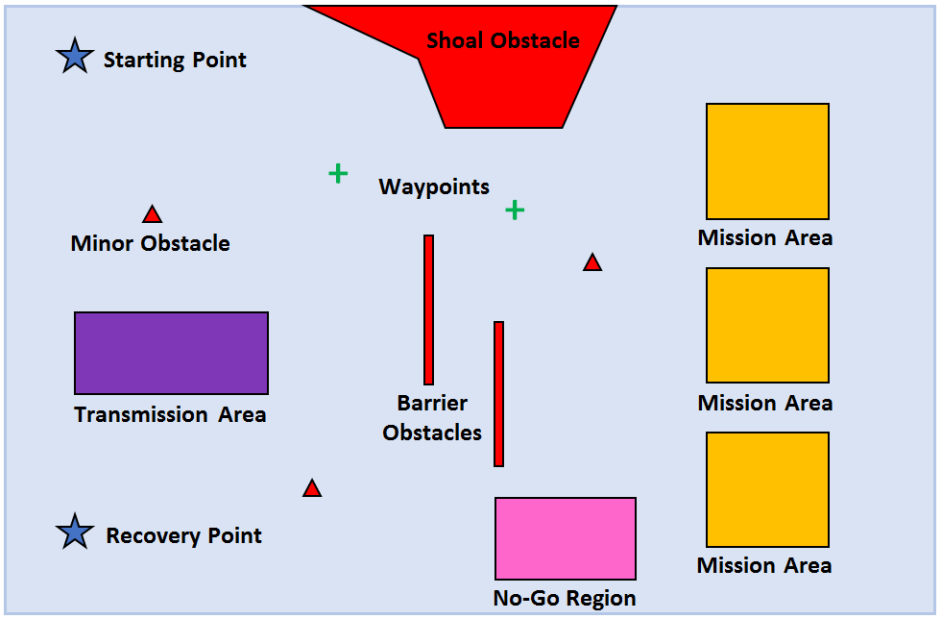


Figure 3.11: Depiction of the UUV mission.

80th percentile of all sample pairs for each boundary. Additionally, the DBSCAN settings ϵ_C and n_{min} were set at 0.2 and 20, respectively.

The number of possible performance boundaries equals the number of possible combinations of the performance modes. In this case study, the four possible performance modes (TF, SS, MS, TS) produce six unique performance boundaries. Since we do not have ground truth for the locations of the performance boundaries, we cannot apply precision, convergence, and coverage metrics to the system. Instead we look at the metrics of distance from the boundary, distribution of performance modes, and distribution of the boundary types.

3.7.3 Experimental Results

The results from each of the searches continued the trends seen on the synthetic test functions. After 850 thousand samples, the S-NDV approach had a more even

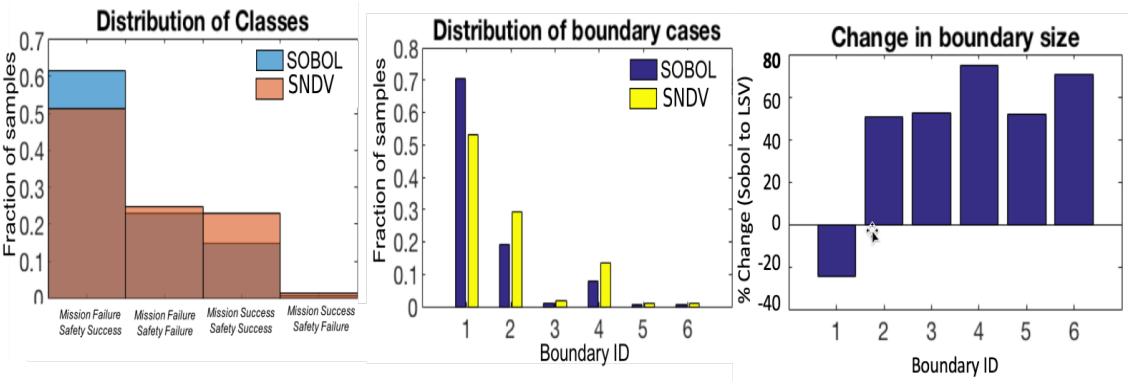


Figure 3.12: Results of the performance mode and boundary distributions for the Sobol and S-NDV search approaches executed on the UUV mission.

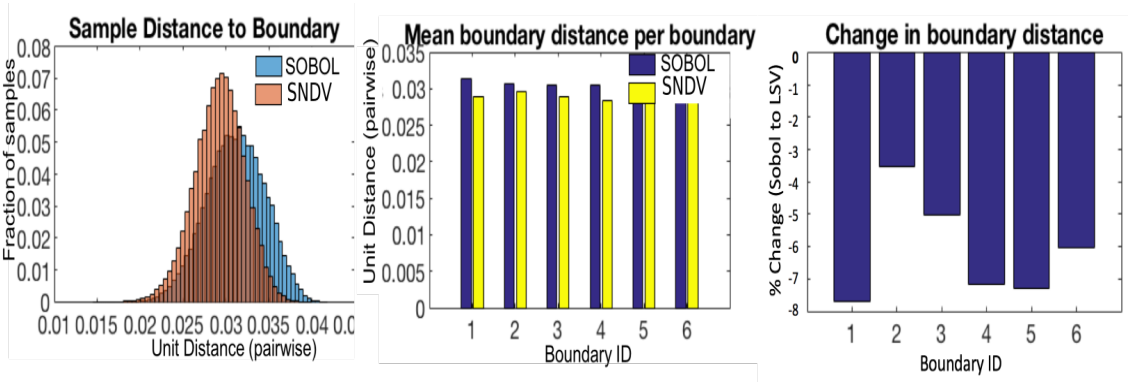


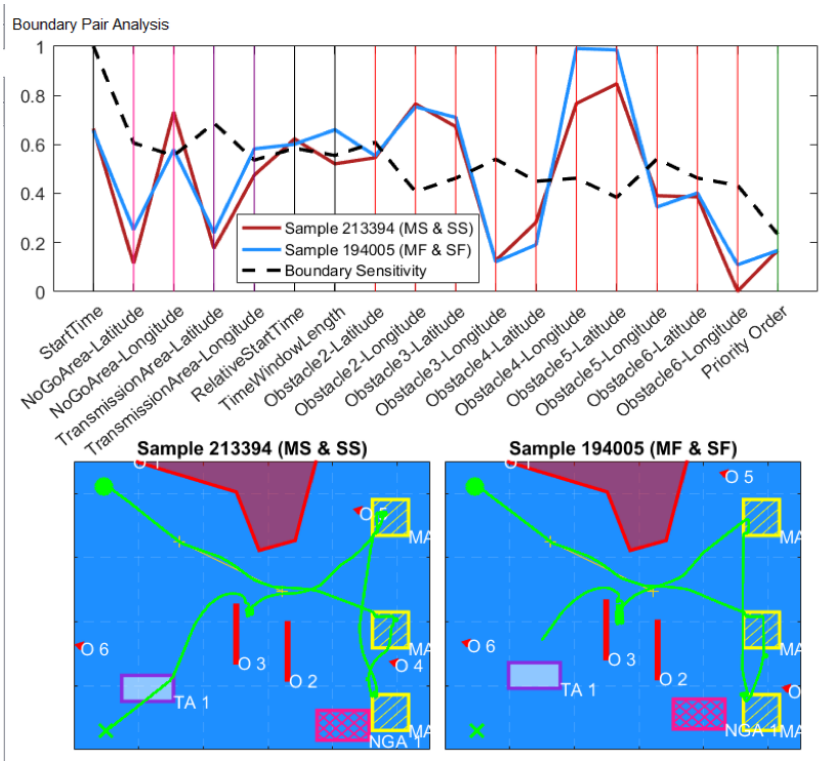
Figure 3.13: Results of the boundary pair distances for the Sobol and S-NDV search approaches executed on the UUV mission.

distribution of samples from each of the performance modes and boundaries. As seen in Figure 3.12(a), the most common performance mode (SS) was sampled less while the rare performance modes (MS and TS) were sampled at a higher rate. This led to a significant change in the resulting performance boundaries (Figure 3.12(b)), where the distribution of samples on the most common boundary (SS / TF) decreased by 20% while the distribution of samples on the remaining boundaries grew by at least 50% (Figure 3.12(c)). This trend indicates that the S-NDV algorithm was significantly better than a space-filling design at sampling the rare performance modes where it detected underrepresented boundary regions.

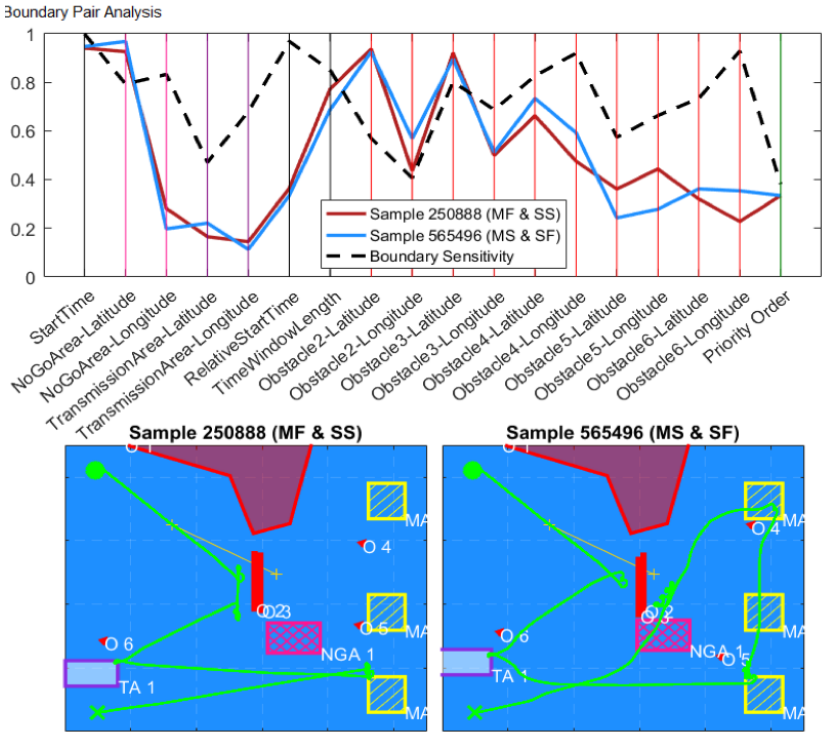
The S-NDV search was also more capable at sampling closer to the boundary. A histogram of boundary distances for all samples is shown in Figure 3.13(a). This plot shows the sample pair distances in the unit space, i.e. the distance after each input state has been normalized according to its state space sampling distribution. The mean distance from the boundary has clearly shifted in favor of the S-NDV search, showing a decrease of around 10%. Although a 10% decrease does not seem particularly significant, when operating in an 18D space, this distance can be put in context by considering how this translates to the actual scenario parameters. Table 3.3 illustrates the equivalent of a 0.01 normalized unit distance for several of the input parameters. It can be seen that this accommodates incredibly large shifts in both time and obstacle position. Thus, in order for performance boundary scenarios to be useful to a test engineer, they must be as close together as possible. The overall lower distance of the S-NDV data set is reflected in the generated boundaries where each boundary is between 4% to 8% closer than those derived from the Sobol data set (Figure 3.13(c)) .

3.7.4 Example Boundaries

In this section, we examine a few examples of the boundaries found by our system in more detail. As part of this research effort, we developed a graphical user interface for exploring the performance boundaries. This tool allows a user to search scenarios by their distance to particular boundaries. In addition, it presents visualizations of the scenarios on the boundary pair and a parallel coordinate plot that



(a) Total Success (Scenario 213394) / Total Failure (Scenario 194005)



(b) Safety Success (Scenario 250888) / Mission Success (Scenario 565496)

Figure 3.14: Example performance boundaries for the UUV mission. The top of each subplot displays a parallel coordinate plot of the normalized input states and the relative sensitivity each parameter. The bottom of each subplot provides visualizations of the scenarios that form the boundary pair in the context of the UUV simulation.

shows both the normalized input states of the scenarios and the relative sensitivity of each input parameter.

In Figure 3.14 we have examples of two boundaries displayed on this GUI: one Figure 3.14(a) is a transition between a total success and total failure while Figure 3.14(b) is a transition between mission success and safety success. Each of these cases demonstrate the major problem with potential field navigation, i.e. getting caught in local minima. While this deficiency is well-known without the use of performance boundaries, the interesting portion of this analysis lies in determining whether the boundary is a useful tool for understanding hidden aspects of the autonomous system's decision-making process.

In both boundaries, visual inspection shows only minor variation between the two cases. The parallel coordinate plots show the change within each state input more clearly. Inputs which were less sensitive, such as the positions of the minor obstacles (O4, O5, and O6), are allowed to vary more widely while inputs that are highly sensitive, such as start time and mission area priority, see much smaller variations.

In Figure 3.14(a), the UUV paths are identical until the vehicle reaches obstacle 3 (O3). At this point, it navigates successfully in Scenario 213394 while in Scenario 194005 it fails to get around the barrier. For a test designer, this may indicate a situation they will want to avoid so that the test article is not lost in the water. For an autonomy developer, it can provide insight about which approach angles and obstacle configurations cause problems for their autonomy that they could fix in later versions.

In Figure 3.14(b) the difference in the scenarios is less obvious via visual inspection. Again, we can refer to the parallel coordinate plot to analyze and diagnose the performance. In Scenario 250888, the autonomy does not interact with many of the elements that see high variations, such as the minor obstacle positions and no-go region. Additionally, elements which are likely to have an impact on the performance, such as start time and mission area priority, do not change. By eliminating these factors from the diagnosis, the engineer is left with the transmission area start time and window length, elements which differ slightly between the two scenarios and are computed to be highly sensitive. In both scenarios, the autonomy clearly heads to the transmission region after failing to find a path around the barriers. However, the shorter time window in Scenario 565496 means that the autonomy makes a decision to head to the transmission area earlier. Ultimately, this means less fuel is spent trying to pass the barrier and it is left with enough fuel to complete the mission. Further analysis of the safety performance in Scenario 565496 shows that while the autonomy is able to complete all of the mission areas, it is unable to navigate out of the local minimum made by the no-go region and the barriers. As time and fuel deplete, the safety protocols of the autonomy force it to travel through the no-go region in order to make it to the recovery point. This boundary provides valuable information to the test engineer about how the transmission window affects many aspects of the autonomous system's decision-making process.

3.8 Summary

In this chapter, we introduced a novel methodology for generating challenging, diverse test cases for an autonomous vehicle based upon discovery and identification of performance boundaries. The two primary intellectual contributions were a set of new objective-less adaptive sampling algorithms designed to find performance boundaries. The first was the NNDV algorithm, which was shown to outperform both uniform random testing strategies and state-of-the art adaptive sampling algorithms at generating high-resolution samples along performance boundaries. This adaptive sampling approach does not require the creation of system specific objective functions. It also specifically designed to handle systems with discontinuous and non-convex response surfaces. The second contribution was S-NDV Algorithm was shown to retain performance as the number of dimensions increase. Outperforming the NNDV algorithm on functions with non-contributing inputs. This allows us to scale our search to 18 input dimensions where previous methods had only addressed as many as 6.

The technique of searching for performance boundaries has applications to any autonomous system and mission. It can easily be adapted to any ground, air, sea-surface, or space platform and any state space that can be parameterized. In the next chapter we will discuss the process of applying this test-generation software to create actual field tests and compare the results on hardware to those predicted by the simulation.

ACKNOWLEDGEMENT

This research was supported by the Department of Defense Test Resource Management Center under work contract W900KK-14-C-0004.

Test System	**Sobol Design**	**LOLA Voronoi**	**GPR Search**	**NNDV Search**
Custom2D	Based on 1000 Samples			
Precision	6.4%	9.53%	11.6%	19.2%
Coverage	31.76%	49.0%	48.43%	59.2 %
Runtime(sec)	0.791	27.2	2.96	0.645
Convergence	800	800	700	700
Plates2D	Based on 1000 Samples			
Precision	6.4%	6.58%	11.6%	19.2%
Coverage	31.7%	39.4%	48.4%	59.2 %
Runtime(sec)	0.791	31.9	2.96	0.64
Convergence	1100	960	600	500
Plates3D	Based on 3000 Samples			
Precision	3.46%	4.22%	7.43%	12.17%
Coverage	1.31%	1.526%	2.64%	4.65 %
Runtime(sec)	0.233	246.0	32.7	2.12
Convergence	26200	N/A	21300	12100

Table 3.1: Comparison of Search Methods

Element	Description
Waypoint	A recommended target to pass through. It is not required for mission success.
Mission Area	A 500x500 meter region that the UUV must enter and remain inside of for a predetermined amount of time. It must complete each mission area in the correct priority order. Completing all mission areas is required for mission success
Transmission Area	A 700x750 meter region that the vehicle must enter and surface for during the open transmission time window. Completing this objective is required for mission success. Surfacing outside of this time window or region is a safety failure.
No-Go Area	A 400x500 meter region that the vehicle cannot enter. If the vehicle enters this region it will receive a safety failure but the simulation continues.
Obstacle	If the vehicle collides with this, it will receive a safety failure and the simulation ends. The barrier obstacles are 700x40 meters while the minor obstacles are triangular and 50 meters to a side.
Operational Area	A 3x3 kilometer region that the vehicle cannot leave. If the vehicle leaves this region it will receive a safety failure and the simulation ends.
Recovery Point	A target circle with a radius of 15 meters. The simulation ends when the vehicle reaches this point. If the vehicle does not reach this point it receives a safety failure.

Table 3.2: Mission Elements

Start Time	Barrier XY	Transmission Time	Obstacle XY
1636.8 s	152.5 m	783 s	457.12 m

Table 3.3: Example of how a 0.01 unit distance translates to parameters of the state space

Chapter 4: Development and Execution of Real World Field Tests

4.1 Introduction

In the previous chapter, we discussed the underlying approach for generating informative test scenarios where the autonomy software was running in simulation. The ultimate goal for the RAPT program was to generate scenarios which could be executed on hardware at a testing range. In this chapter we will cover the improvements and studies that were necessary to take the simulation-based results from our RAPT software and turn them into field tests. This includes studying the robustness of our algorithms to noise and increasing the number of outputs we consider when finding performance boundaries.

The effective transfer of simulation generated results to the real-world is one of the biggest hurdles facing the machine learning community. There are many ways in which errors can be introduced which cause the simulated system to diverge from the real-world. These include effects such as modeling errors, unaccounted sources of uncertainty, simulated versus real sensing data, or differences in the way the hardware platform communicates with autonomy software versus the simulator.

As part of the 2017 development effort we took several steps to account for the accuracy of our simulation. We performed extensive hydrodynamic testing of

(a) (b)

Figure 4.1: IVER UUV used for the on-water tests of the autonomous vehicle software. These photographs were taken during the hydrodynamic tests conducted in the Chesapeake Bay.

the vehicle in the Chesapeake bay to ensure our simulation models of the platform were accurate. This data included dynamic parameters such as buoyancy, drag, and surge speeds as well as mechanical parameters such as current draw and battery drain for different motor loads. We created new tidal current files based on data from oceanographic stations in the proximity of the Keyport test range. Using averaged data from the months of September and November, when the field tests were scheduled to occur. Finally, we updated the GPS/INS models to incorporate realistic drift in the state-estimate of the UUV.

It is our assertion that distance from a performance boundary can be used as a measure of certainty in vehicle performance. To effectively test a performance boundary we want to carefully select pairs of tests which straddle the boundary without being too close together. Therefore, we seek to establish our test generation software's ability to accurately identify the performance boundaries in the presence

of uncertainty. We were primarily concerned with two issues. The first is the ability of our search and boundary identification algorithms to handle a stochastic system under test. The second is how do we select tests that will be robust to transfer effects and how do we anticipate shifts in the systems behavior before we run the tests on the water.

In this chapter, we will discuss our approach for addressing the problem of generating scenarios from a non-deterministic simulation and the results of running tests designed by RAPT in the field. In section 2, we will discuss the results of applying our search and boundary identification algorithms to probabilistic systems. In section 3, we introduce sub-clustering algorithms which allow us to quickly retrieve and analyze information about any scoring metric of our system. In section 4, we will discuss the test-design process that was utilized to build the test-plan for the 2017 demonstration. In section 5, we will discuss the results of the 2017 demonstration. Finally, in section 6, we present our conclusions for the final demonstrated performance of the RAPT software.

4.2 Stochastic Systems

There is an inherent level of uncertainty when executing a scenario in the real world. That uncertainty could be the result of the error in the sensor inputs, stochasticity in the vehicle dynamics, or random processes inside the autonomy software. To accurately reflect reality a simulation environment will need to possess some level of uncertainty as well. Which means in turn that our test generation

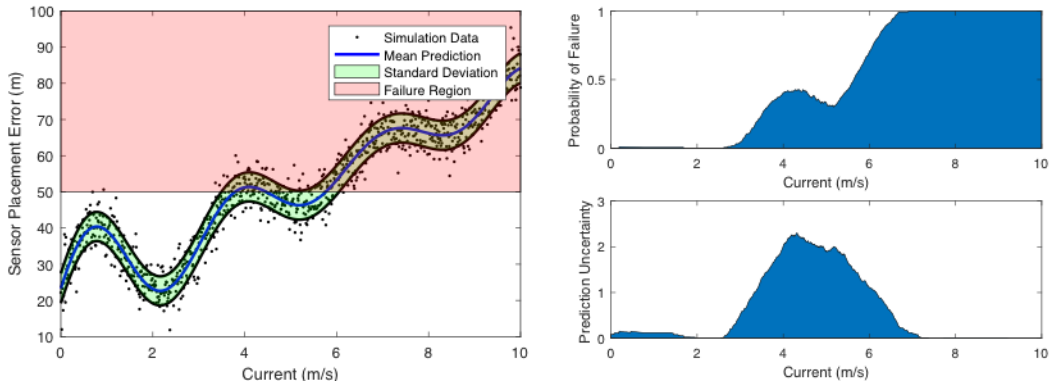


Figure 4.2: (Left) A plot of sensor placement results versus ocean current, individual points vary with gaussian noise around a mean function, the red region indicates the placement error has exceeded mission parameters. (right top) The probability of the mission failing as current increases, (right bottom) the performance boundary between success and failure is indicated by a region of high uncertainty.

algorithms need to be robust against probabilistic effects. In this section we will discuss the results of applying our test-generation algorithms to an SUT with noisy output.

4.2.1 Gaussian Noise on Continuous Outputs

First let us consider systems which have continuous outputs that are converted into a binary score. For example consider a UUV mission to place a sensor at a specific location where it fails if it places the sensor further than 50 meters away from its designated target. Let's assume that the relationship between sensor placement error and ocean current which can be modeled as a non-linear mean function with Gaussian noise. An example of this type of system is illustrated in 4.2.

For this sensor placement example we are left with a region where the probability of failure is increasing from 0 to 1 as the current increases from 3m/s to 7

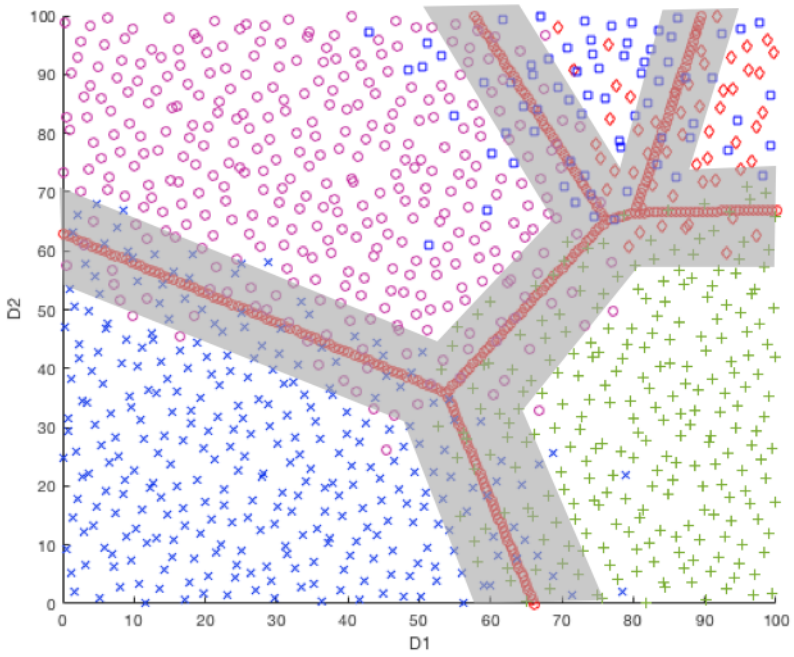


Figure 4.3: A scatterplot of the noisy Plates 2D function with boundary width $\sigma = 0.1$. The colored markers indicate the different classes, while the red lines indicate the true performance boundaries. The grey shaded region indicates the area within 1 standard deviation of the boundary.

m/s. Unlike the boundaries of the previous chapter in this case there is no sharp division between classes, instead the boundary is a region where either class could occur with some probability. We refer to this as a "fuzzy" boundary region. This is the easiest form of noise for our system to manage as the boundary region remains a region with high variance in the binary score and samples adjacent to samples from another class will only occur in the boundary region. In our sensor placement example this is the region of high variance which occurs between 3-7 m/s.

To determine the effect of continuous noise on the adaptive search process we created a version of our custom Plates2D function where instead of a sharp boundary between categorical outputs the boundary was a region where two Gaussian

distributions overlapped. The probability of obtaining a sample of an incorrect performance mode is given by the equation $P(\ C\ x) = 0.5 * e^{(} - (d(x)^2)/(2 * \sigma))$ where $d(x)$ is the distance of the sample x from the nearest boundary. An illustration of this function is shown in Figure 4.3 with the true boundaries shown as red lines and the standard deviation from the boundary shown as a shaded gray region. The width of this region was described via the standard deviation σ of these distributions and was varied between 0 and 0.4. We continue to use the precision and coverage metrics of the previous chapter. Where a sample is considered to be in the boundary region if it is within a distance of 0.1 of the boundary's center. The results of this experiment are show in 4.4

While the resulting boundary regions are wider than in the original deterministic function the NNDV search successfully samples the correct regions and returns tighter boundaries than the Sobol set. As boundary width increases the NNDV search begins to degrade in performance. Once the standard deviation of the boundary region reaches 0.35 the adaptive search begins sampling in the same space-filling manner as the Sobol design. At this point the entire search space becomes probabilistic as all of the performance modes overlap, causing the entire system to consist primarily of noise. As such, there is insufficient information for the adaptive search to exploit and defaulting to a global search approach is appropriate.

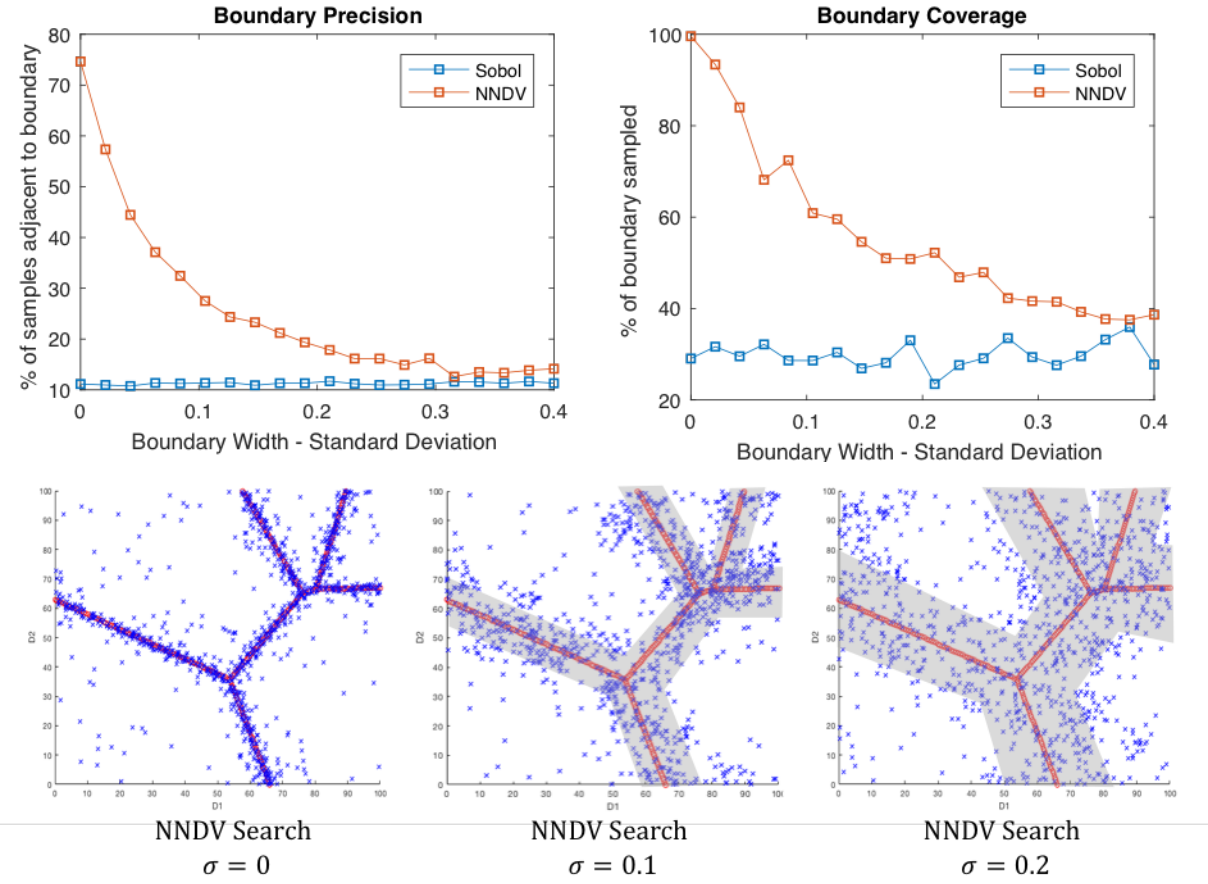


Figure 4.4: (Above) Plots comparing the performance of the Sobol and NNDV search approaches as the width of the performance boundary increases. (Below). Scatterplots showing the effect of increasing boundary width on our adaptive search.

4.2.2 Global Error

The more difficult case to consider is when there is a global probability of failure across the entire testing space. Rather than uncertainty being localized to the region around a performance boundary there will be random occurrences of unexpected behaviors throughout the testing space. The challenge for our system is differentiating between these outliers and actual performance boundaries. To demonstrate the effects of global distributions on our algorithms we took the same Plates2D Custom test function and applied a global error rate to the final categorical score. For this system the probability of receiving a sample with an incorrect performance mode is $P(\ C\ x) = \gamma$. An illustration of this function can be seen in Figure 4.5

This is the hardest form of noise for our system, as any outlier sample can create a high variance region which will cause the search to erroneously search that area. The current iteration of our algorithms will treat an area with high variance as a boundary even if that variance is the result of probabilistic effects rather than a change in the boundary. The results of applying increasing levels of global noise to our system are shown in Figure 4.6.

To demonstrate the effects of global distributions on our algorithms we took the same Plates2D Custom test function and applied a global error to the final categorical score. The results of a system with 95% probability of getting a dominant performance mode and 5% chance of receiving a different mode is shown in Figure 4.6

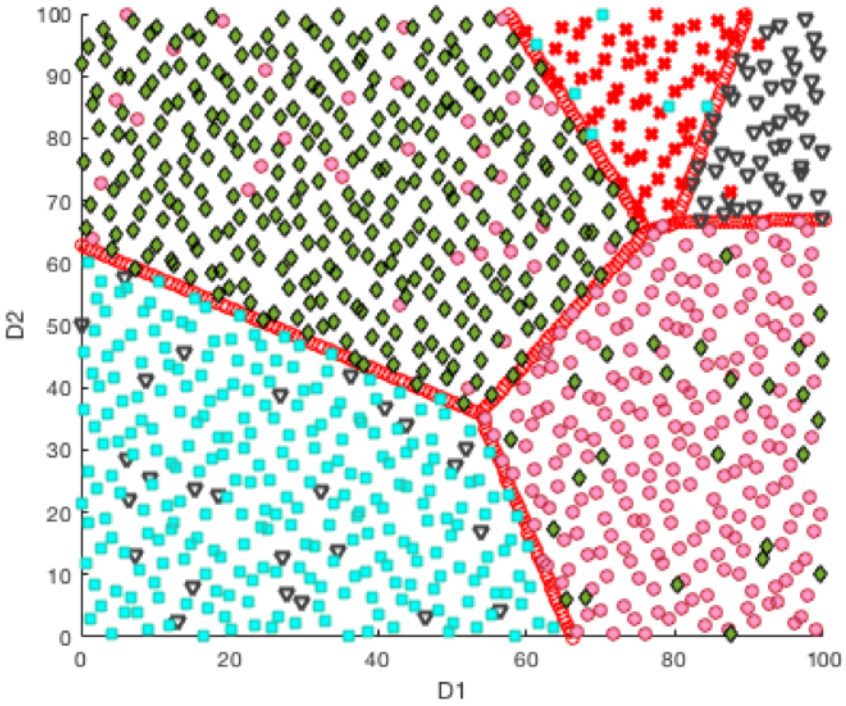


Figure 4.5: Scatter plot of our noisy Plates 2D function with a global noise rate γ=0.1. Each different colored marker type indicates a different performance mode and the red lines indicate the true boundaries of the system. Samples from incorrect performance types can be found uniformly distributed throughout the space.

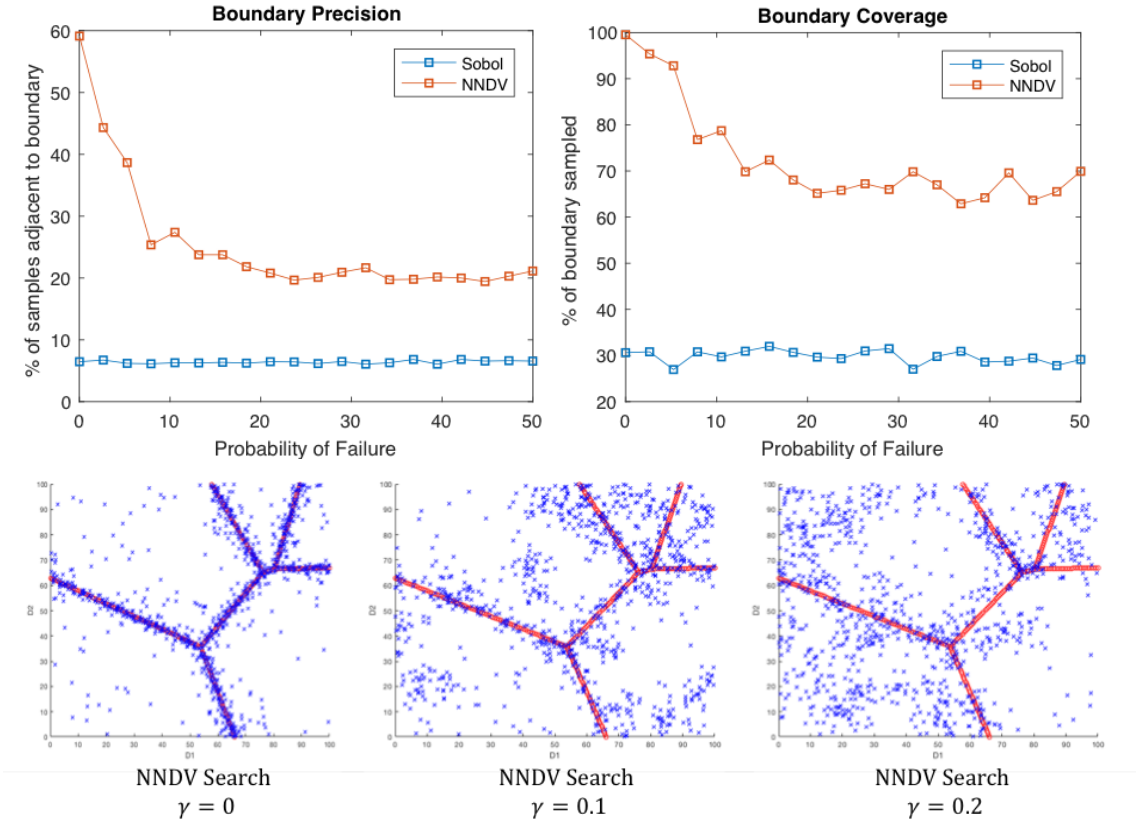


Figure 4.6: (Above) Plots comparing the performance of the Sobol and NNDV search approaches as the global failure rate increases. (Below). Scatterplots showing the effects of increasing global noise on our adaptive search.

The performance of the adaptive search drops steeply as the failure rate increases from 0 to 10 percent. However despite a decline in precision and coverage the adaptive search still outperforms the Sobol design for systems with large error rates. This gives us confidence that our adaptive search techniques are an improvement over space-filling techniques for finding the true boundary regions, even in the presence of global error.

4.2.3 Uncertainty in the UUV Simulation

To add realistic uncertainty to our UUV simulation we added stochasticity to the vehicle dynamics, random perturbations of the ocean current, and drift in the inertial navigation system (INS). To understand how these changes to the simulation will affect the results produced by our algorithms we must first characterize the effect this type of noise has on the output of the system. As well as how these non-deterministic effects impact the behavior of the autonomy.

For our noise characterization tests we generated 40,000 scenarios in a 5-dimensional state-space where we varied start-time, obstacle position, and no-go area position. For this study each scenario was run 10 times. The mission objective was to avoid obstacles and no-go areas and explore two survey areas before returning to the recovery point. The vehicle had the ability to detect obstacles using its sonar sensors but had to rely on its state estimate to avoid no-go areas and reach the survey areas. The survey areas were deliberately made small for this test to increase the overall difficulty. An illustration of a representative scenario run three

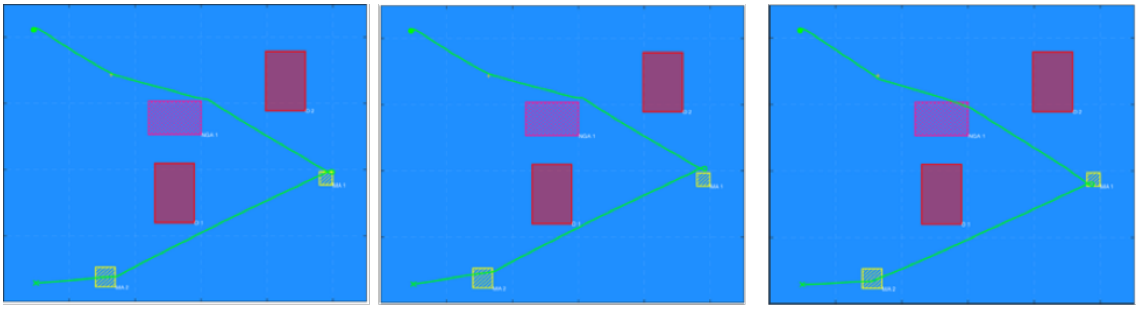


Figure 4.7: Examples of three different runs of the same scenario in the presence of estimation error and random currents.

different times is shown in 4.7

4.2.3.1 Continuous Noise on Simulation Outputs

It is clear from these examples that the amount of error in the vehicle's state estimate is directly responsible for deviations in the trajectory that cause it to either travel through the shaded no-go area or miss the survey area. The vehicle's autonomy software fails to account for this error when performing its path planning and thus fails under low levels of estimation inaccuracy. The effect of the error is most strongly in evidence if we plot the relationship between the latitude of the no-go area with the distance of the closest point of approach, see Figure 4.8. This relationship has the same properties as our sensor placement example from Figure 4.2. Where there is some underlying function with Gaussian noise. Resulting in the same type of "fuzzy" boundaries where there is uncertain vehicle performance in a wide region of the state-space. As we established earlier, these are the types of boundaries which our system can handle gracefully.

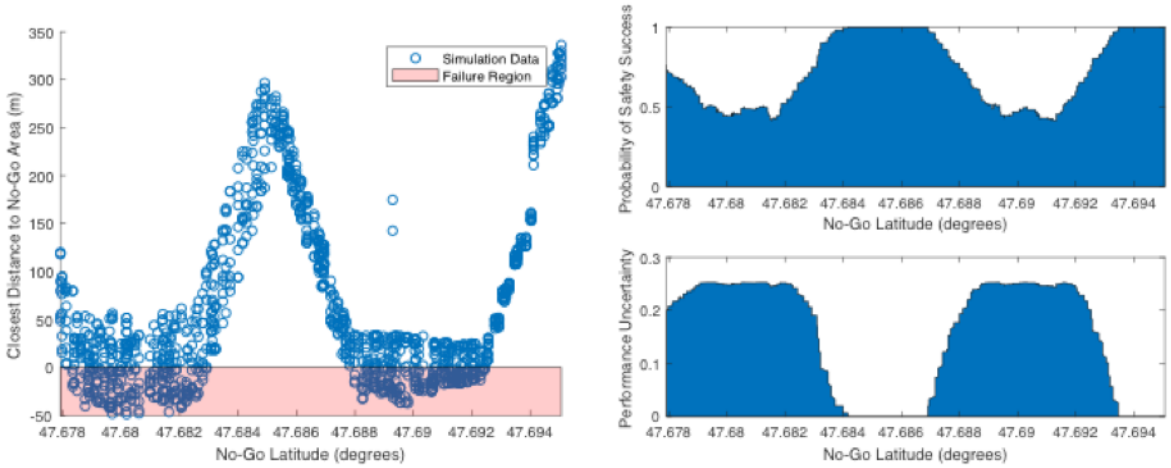


Figure 4.8: (Left) A plot of closest distance to No-Go Area 1 versus the latitude of No-GO area 1, individual points vary with Gaussian noise around some mean function, the red region indicates the vehicle has entered the no-go area. (right top) The probability of the safety success as the No-Go areas latitude moves north, (right bottom) the variance of the safety safety success as the No-Go areas latitude moves north.

4.2.3.2 Inferring Boundary Locations from Variance

Given these results it would be reasonable for us to define the boundary region as the region where there is high variance in the success and failure scores. However, despite the scenarios in these regions having high variance in the binary scores they have relatively constant performance otherwise. All of runs for a single scenario follow the same general trajectory with small deviations due to estimation error. Our goal of finding scenarios where the decision-making process of the autonomy changes sharply is not fully satisfied. The types of scenarios that we are more interested in discovering are illustrated by another scenario from our data-set, Scenario 35775, which is depicted in Figure 4.9

In this scenario the vehicle had a 50% chance of either returning home safely

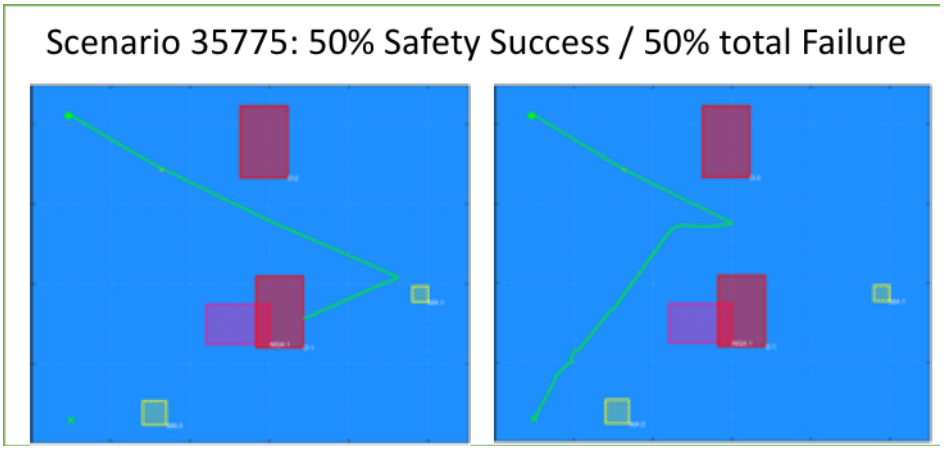


Figure 4.9: Example of multiple runs for a probabilistic scenario. Small inaccuracies in the position estimate can cause large changes in when it decides to return home.

or colliding with an obstacle on the return path. This is a critical location in the state-space that illustrates the effect of state estimation error on the decision process of the autonomous vehicle. In one case it believes it has enough fuel to continue the mission and in the other it believes it lacks the fuel necessary to complete the mission. As there are no obstacles or no-go areas on the ingress path to the first mission area we can infer that the decision must only be influenced by the ocean current. As such we investigated the relationship between start time, which determines the tidal current magnitude, and the closest point of approach to the survey area. The plot of this relationship is shown in 4.10 with Scenario 35775 highlighted in Red.

The tidal current in our simulation peaks between the hours of 2am and 3 am and this is reflected in the data. During these hours there is a sudden jump in the distance between the survey and the closest point of approach. This is due to the vehicle determining it lacks the fuel to overcome the current and must return early. For all other times the vehicle is almost guaranteed to reach the survey area and its

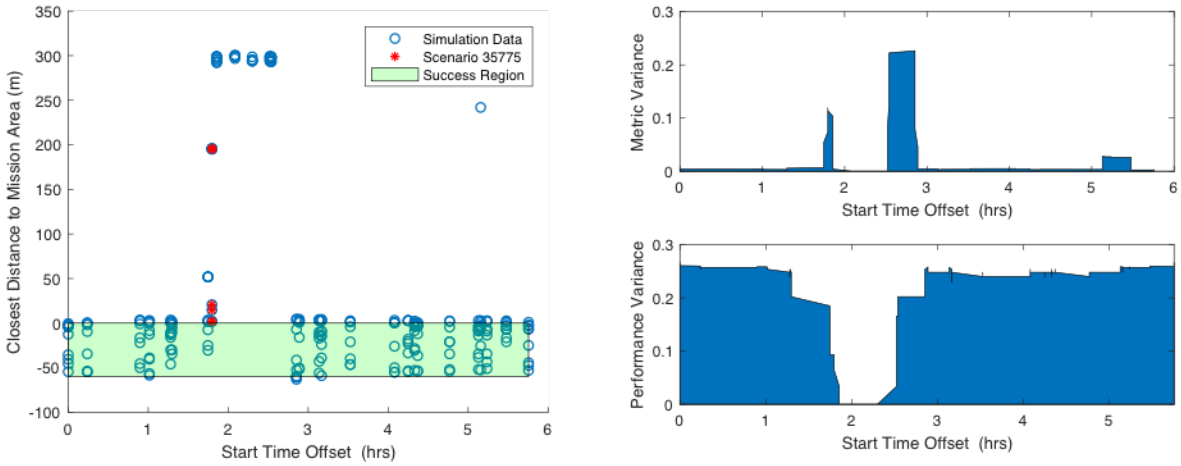


Figure 4.10: (Left) A plot of closest distance to the mission area versus the start time offset from midnight, individual points vary with Gaussian noise around some mean function, the green region indicates the vehicle has successfully entered the mission area. (right top) The probability of mission success as start time offset changes, (right bottom) the variance of mission success as the start time changes.

success or failure depends entirely on the magnitude of the state estimation error.

One issue this scenario illustrates is the problem with using variance as an indicator of a performance boundary. In Figure 4.10 we compare the variance in the continuous distance metric values to the variance in the binary mission success score. Even though the behavior of Scenario 35775 is representative of the type of behavioral transition we want our system to find, its variance for the binary mission success metric is actually lower than other regions of the state-space. However, a spike in the variance of the continuous closest distance metric occurs at Scenario 35775. Aligning perfectly with our desired performance boundaries.

4.2.3.3 Conclusions on UUV Simulation Uncertainty

Incorporating uncertainty into our simulation in the form of current perturbations and state estimation error has the effect of applying Gaussian noise to the continuous outputs of our simulation. Since we use binary success criterion to drive our adaptive search this is expressed as "fuzzy" boundary regions between success and failure. Given our results for the synthetic systems we can have some confidence about the ability of our algorithms to find the performance boundaries for the UUV simulation

The larger issue is whether searching for the performance boundaries of the binary scores yields the most informative scenarios. As indicated by our results for Scenario 35775, the performance boundary caused by turning back early could not be identified by looking at the binary success score. Rather it could only be discovered if we look at the continuous distance metrics. Therefore, if we were to instead use the continuous metrics to drive our adaptive search we could potentially find transitions in the decision making process of the autonomous vehicle.

4.3 Sub-clustering

As discussed in the previous section, binary success criteria are not always the best indicators of when a change in the vehicle's behavior occurs. Sometimes it is better to use continuous metrics such as closest distance to a waypoint or fuel consumed as they are more heavily affected by changes in the vehicle's trajectory. What we want is to obtain the boundary information for any score element in the score

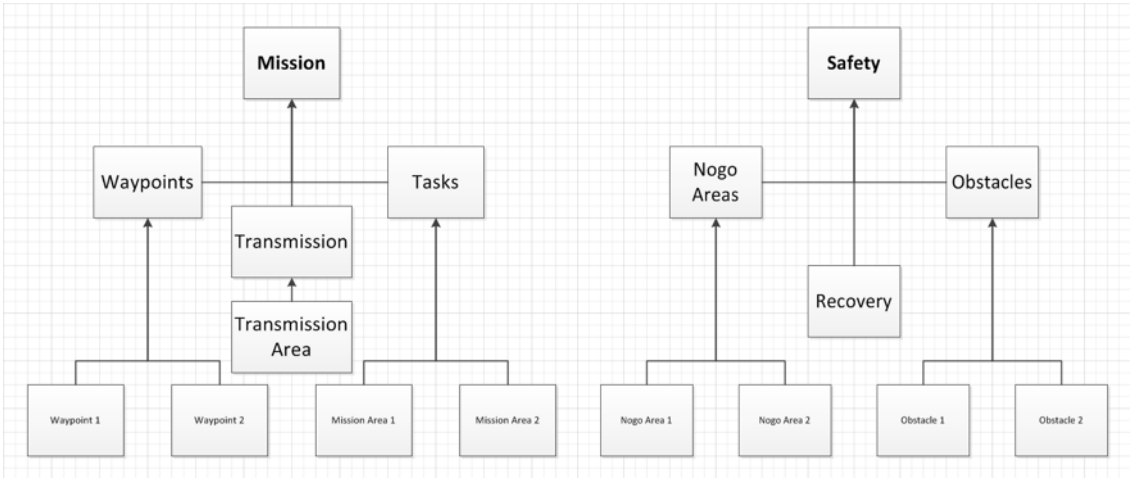
tree without having a priori knowledge of which ones will be important. However, it is too expensive to compute all of them simultaneously using our previous clustering technique. Therefore, what we require is a set of analytical tools which allow us to explore the boundaries for any of our possible scoring metrics. In this section we will discuss how we developed a hierarchical boundary identification method which will allow us to quickly compute and retrieve boundary information for any scoring metric in the simulation.

4.3.1 Definition of Sub-clusters

The key to our new approach is the sub-score tree, a structure which defines the relationships between binary success criterion such as Mission Success to sub-scores such as Waypoint Success or Transmission Success. These sub-scores are in turn computed using continuous data such as the closest distance to a waypoint or time spent conducting the survey. This hierarchical relationship between sub-scores and binary criterion is a fundamental part of the sub-clustering process. A diagram of the sub-score tree for the UUV mission is depicted in Figure 4.11

Each element of the sub-score tree H has two properties, a set of indices $H.K$ and a set of child nodes H.children. The indices $H.K$ indicate the position of that score element in the score vector Y.

Using the sub-score tree, it is possible to develop both sub-clusters and sub-boundaries for our system. We define a cluster $V^C \subset V$ as the subset of samples in V which all have the same class $c \in C$. Each of these clusters are composed of a



(a) Mission Success Tree (b) Safety Success Tree

Figure 4.11: Diagram of the hierarchical score trees for our UUV simulation illustrating the first 3 levels of the score tree for both mission success and safety success.

number of disjoint sub-clusters such that $V^C = V_1^C, V_2^C, \ldots V_n^C$. The cluster V^C is created by applying our algorithms to the score elements at the ith layer of H and its sub-clusters are created by applying our algorithms to the $(i+1)$ layer of H. For example, if we cluster based upon the Mission and Safety scores to create V^C, then we create its sub-clusters $V_1^C, V_2^C, \ldots V_n^C$ by applying the clustering algorithm to the next level of scores; Waypoint, Transmission, etc. Finally, we define a sub-boundary a set of paired samples which lie between two sub-clusters, $B^{(C_1,C_2)} = \{[v_1, v_2], \ldots, [v_(n_1), v_(n_2)]\}$ where all members of the boundary are members of V^C and each pair is made up of a member of V_1^C and a member of V_2^C.

The second category of clusters we introduce is the sub-score cluster V^k which is a set created by applying our clustering technique to the kth element of the score vector. These in turn have a sub-score boundary which we designate as B^k. These sub-score clusters are our true objective and later in this section we will discuss how

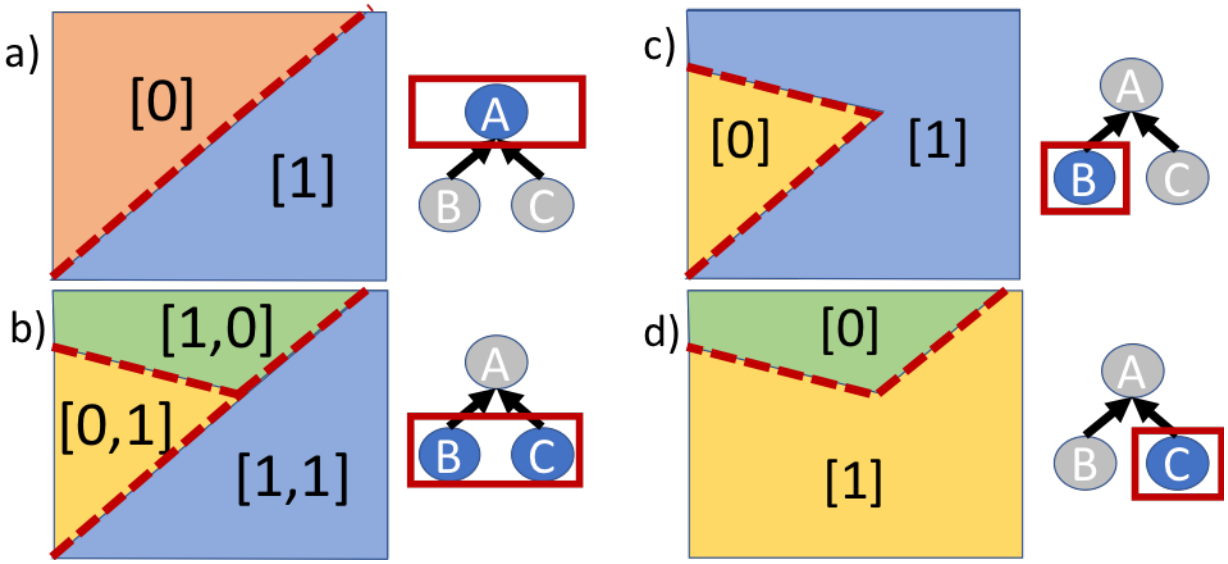


Figure 4.12: Illustration of the various sub-clusters and sub-boundaries for a simple three element score tree. (a) The clusters and boundaries for the root element A, (b) sub-clusters formed by applying our clustering algorithm to both B&C simultaneously, (c) the sub-score clusters for element B, (D) the sub-score clusters for element C.

these two types of clusters are related and how each of these are utilized for our analysis of the UUV simulation.

A diagram showing the different sub-clusters and sub-score clusters for a simple 3 element tree is illustrated in Figure 4.12. In this system the primary clusters are defined by the root element A, which is computed via an AND operator on the leaf elements B & C. The sub-clusters provide different ways of sub-dividing the system using the values of leaf elements. If both B& C are used, Figure 4.12b, the original parent cluster [A=0] is split into two clusters; [B=0,C=1] and [B=1,C=0]. If we were to isolate either score element B or C we could create sub-score clusters based upon their values alone, seen in Figure 4.12c,d.

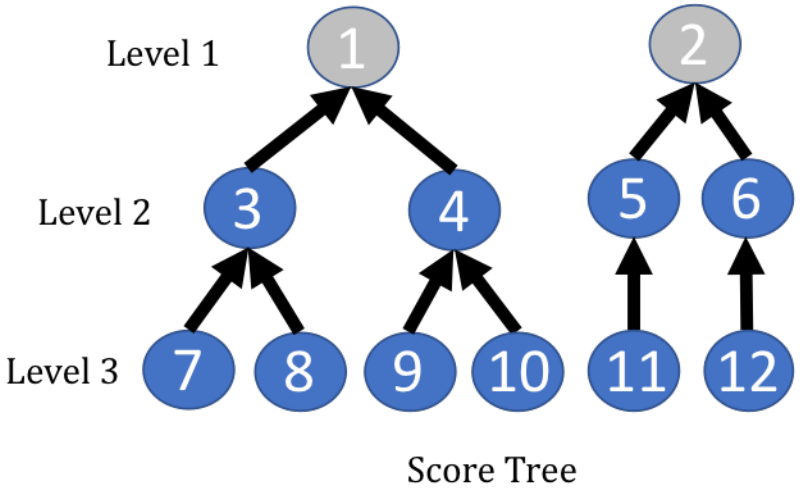


Figure 4.13: The 12 element score tree structure of our synthetic system.

4.3.2 Synthetic Sub-score Function

Before we could develop our sub-clustering techniques we required a synthetic test system which replicated the structure of our target SUT while providing ground truth of the sub-boundary locations. This would allow us to determine the true boundaries of the system and allow us to apply our coverage and convergence metrics to any search against the system. To do this we created a 2 dimensional system with a score-tree consisting of 12 score elements and three layers. The tree structure for this system is shown in Figure 4.13. Each cluster in this system could be broken down into progressively more sub-clusters through a partitioning process. Where each cluster is broken down based upon the clusters which occur at the next level of the score tree. From 4 types of clusters based upon the level 1 score elements all the way to 27 clusters based upon the level 3 elements. A diagram of these clusters is shown in Figure 4.14.

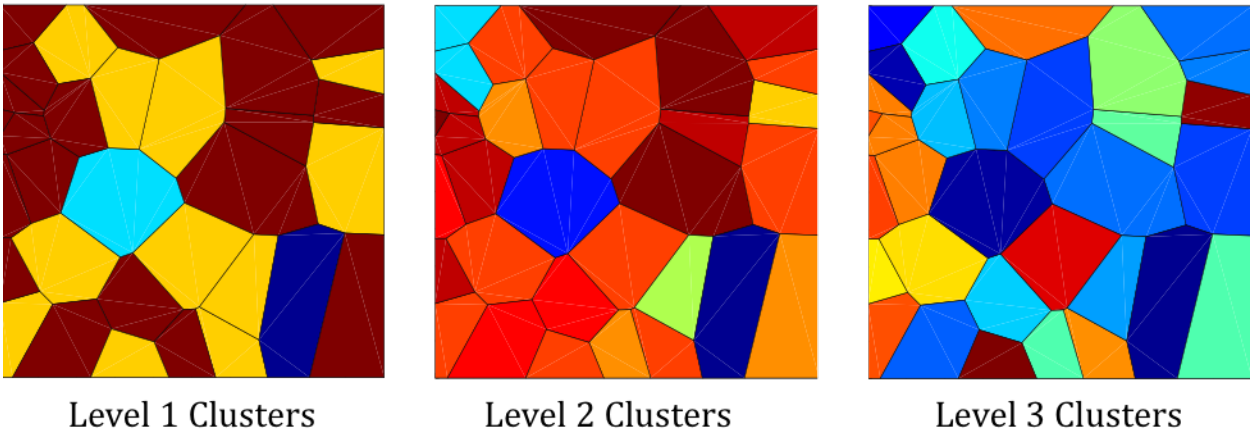


Figure 4.14: Example of how our 2D Hierarchical system breaks up into successively smaller and smaller clusters as we apply our clustering algorithms to progressively lower levels of the score tree.

As illustrated in Figure 4.14, the more sub-score elements used for clustering the more sub-clusters are created. Our synthetic system has 4 classes which define our primary clusters, each of these can be broken into smaller sub-clusters by applying clustering algorithms to second level of the tree. These sub-cluster can then in turn be broken down even further by applying a clustering algorithm to the third level of the tree . An illustration of this process can be shown in Figure 4.15.

Going hand in hand with the concept of sub-clusters is the concept of sub-boundaries. Sub-boundaries are the divisions between sub-clusters inside of a larger cluster. In our synthetic system there are 6 primary boundary types based on the transitions between different classes. Using the sub-clusters of the intermediate scores we can discover sub-boundaries which exist between the primary boundaries. Dividing each cluster and sub-cluster into smaller and smaller sets. An example of this process can be shown in Figure 4.16.

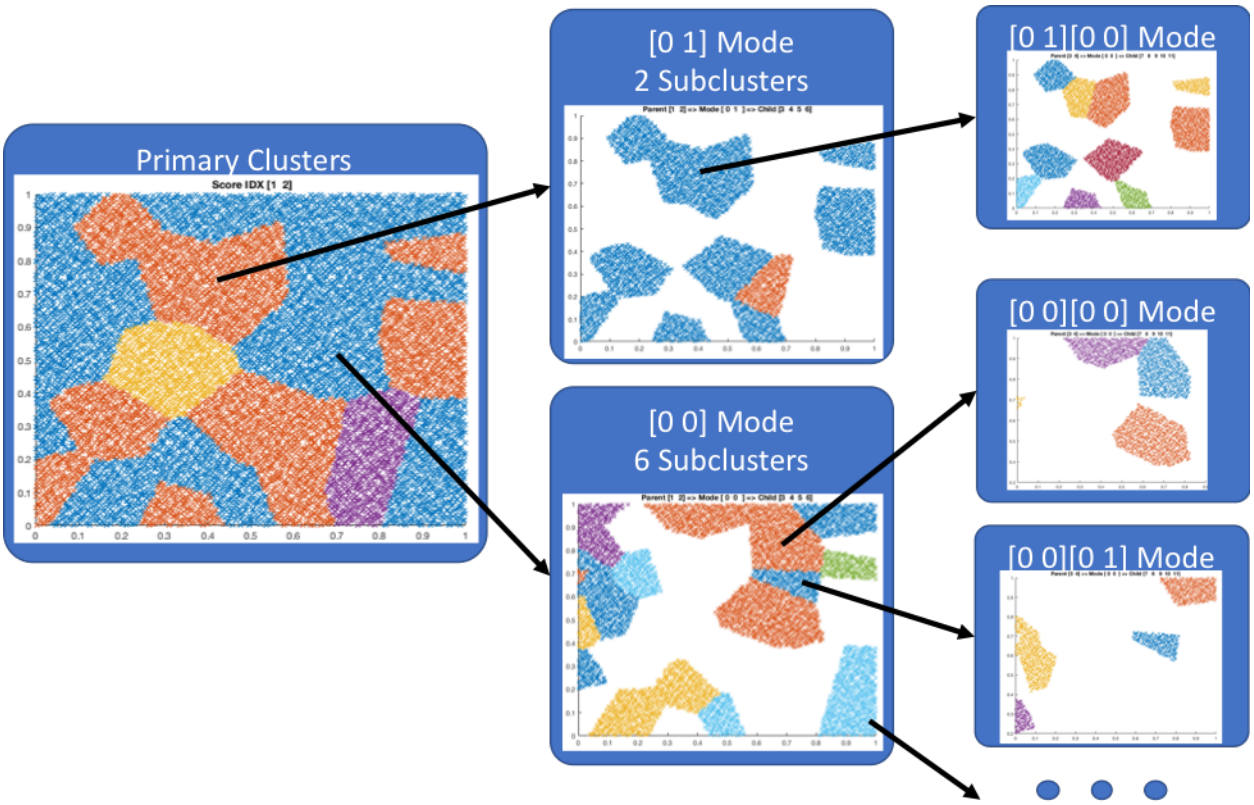


Figure 4.15: Example illustrating how each cluster of our system is split into smaller sub-clusters. The [0 1] cluster can be broken up into 2 sub-clusters while the [0 0] cluster can be broken up to 6 sub-clusters using the Level 2 score elements. These can be broken up further using the Level 3 score elements.

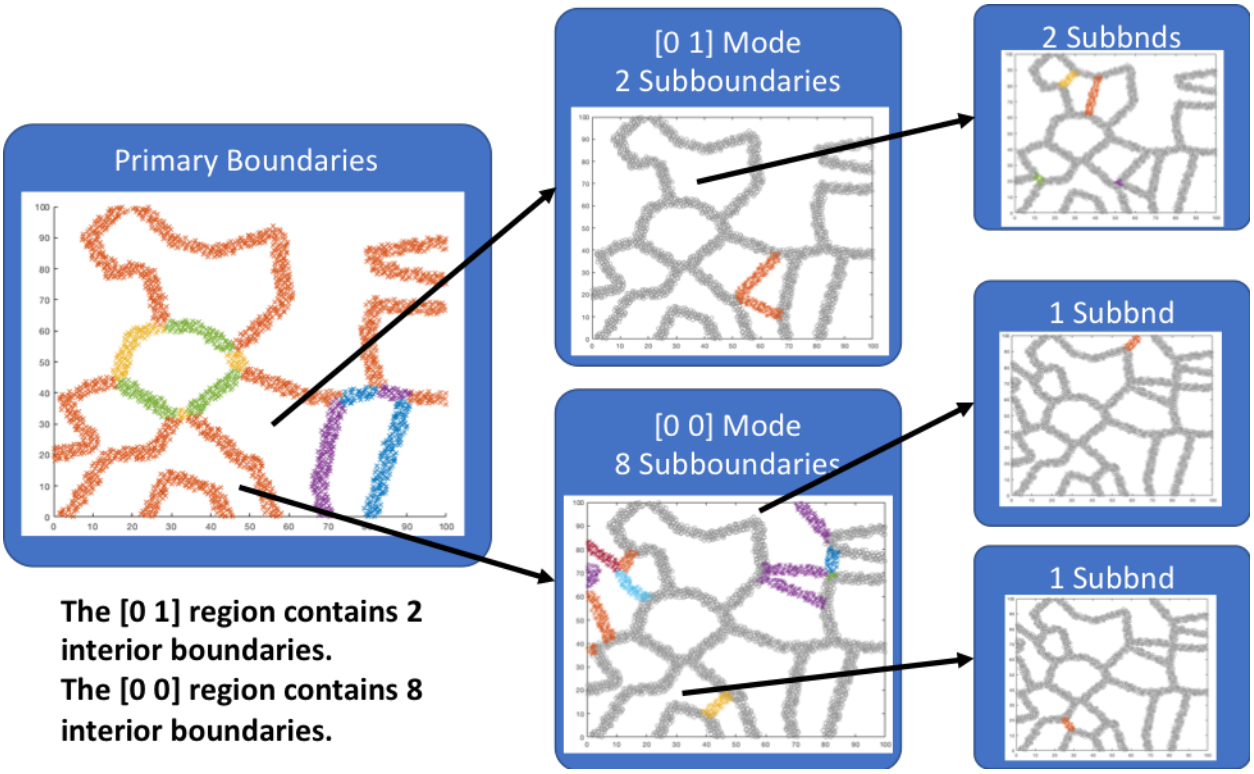


Figure 4.16: Example illustrating the sub-boundary process. At each level of the system the sub-boundaries (color lines) represent the separation between the sub-clusters at that level and therefore occur in between the boundaries of their parent levels (gray lines).

4.3.3 Hierarchical Sub-Clustering

What we want is to obtain the boundary information for any score element in the score tree without having a priori knowledge of which ones will be important. However, it is too expensive to compute all of them simultaneously using our previous clustering technique. In this section we introduce our approach for generating a sub-boundary structure which can be created more quickly than our prior method and allows for efficient retrieval of any score boundary. Our new sub-boundary algorithm utilizes the hierarchical nature of the sub-score tree to break the problem into smaller pieces. Allowing us to create clusters and their boundaries for every metric in our score tree in a reasonable amount of time. This process is illustrated in Figure 4.16 where each cluster is broken up into smaller and smaller sub-clusters by iteratively applying our clustering algorithms.

The algorithm works as follows, the primary clusters of our system are identified by applying Mean-Shift clustering to the root elements of the score tree . Each of these clusters is then subjected to clustering using the metrics at the next level of the score tree. These new clusters are added as children to the parent cluster. This process is applied recursively until it has reached the bottom of the score tree or a cluster cannot be subdivided. At each step of the sub-clustering process we use pair-wise comparison between members of different clusters to identify the boundary pairs. Since the sizes of these clusters become progressively smaller as the process continues each child in the sub-boundary tree takes less time to compute than its parent. This process is described more formally in Algorithm 3.

Algorithm 3 SUBBOUNDARIES(X, Y, H)

Input: A set of sampled states X with scores Y and a score tree H
Output: A sub-boundary tree T

Set the sub-score indices $K = H.K$
$Clusterusingtheselectedsub - scores[L, C] = MeanShift(Y(K))$, where L is the label vector and C are the classes.
Create labeled sample set $V = [X, Y, L]$
Set the sub-clusters for V as $V^(C_1), .., V^(C_n) \forall C_i \in C$ where $\forall l_i \in V_i^C = C_i$
for all $C_i \in C$ **do**
  for all $C_k \in C, k > i$ **do**
    $I_i = knnsearch(X^{C_i}, X^{C_k}, 1)$
    $I_k = knnsearch(X^{C_k}, X^{C_i}, 1)$
    Create boundary pairs $b_k = [v_j, v_l] \in B^{(C_i, C_k)}$ if $I_(ik)(x_j) = x_k$ and $I_{ki}(x_j) = x_l$
    $B.append(B^{(C_i, C_k)})$
  end for
  $T^{C_i} = Subboundaries(X^{C_i}, Y^{C_i}, H.children)$
  $S.append(T^{C_i})$
end for
return $T = [V, B, C, S, K]$

The output of this algorithm is a boundary tree structure. Each level in the boundary tree relates directly to a level in the score tree. Each node in the boundary tree contains information about the clusters at that level, the boundaries for those clusters, the score indices used for clustering, and its child sub-boundary nodes.

4.3.4 Score specific Sub-boundaries

Our objective is to identify the boundaries for every score element in the tree. However, it is too expensive to apply our original boundary identification algorithm to every score element simultaneously. Instead, we can use the sub-boundary tree T, which we computed using our previous algorithm, to reconstruct the boundaries associated with any scoring element. This means we can retrieve clusters and boundaries for any given score element with minimal additional computation. This process involves searching the sub-boundary tree structure for all sub-clusters which have the same value for the specified score element. Then merging all of the identified sub-clusters and sub-boundaries into a single set. Our method for doing so is given

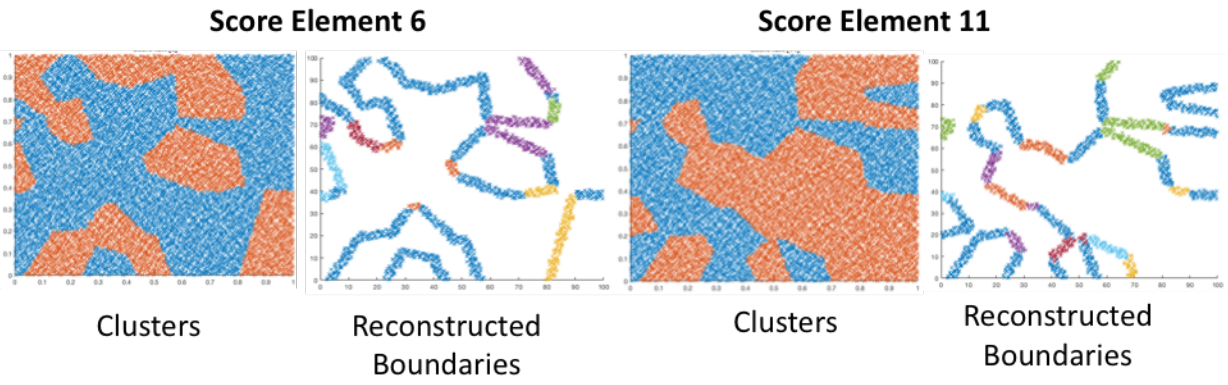


Figure 4.17: Examples of reconstructed sub-score clusters and boundaries for the synthetic system.

in more detail in Algorithm 4

Algorithm 4 RECONSTRUCT(T, y_s, K)

Input: A sub-boundary tree T, a selected score value y_s, and score indices K_s.
Output: A sub-score cluster V_s and sub-score boundaries B_s

```
[V, B, C, S, K] = T
if K = K_s then
    Find y_k = nearestNeightbor(y_s, Y)
    Set V_s = V^k where y_k ∈ V^k
    Set V_s = B s.t. ∀b_i = [v_j, v_l] ∈ B_s , either v_j ∈ V_s or v_l ∈ V_s
else
    for all T_i ∈ S do
        [V_i, B_i] = Reconstruct(T_i, y_s, K_s)
        V_s.append(V_i)
        B_s.append(B_i)
    end for
end if
return [C, B]
```

An example of reconstructed sub-score boundaries is shown in Figure 4.17. In this example we use score element 6 and score element 11. The clusters are shown in red and blue for when the score is 0 or 1 respectively. The different colors in the reconstructed boundary indicate the different sub-boundaries that were merged in the reconstruction process.

The sub-boundary technique has been successfully applied to the output of the UUV simulation. The sub-score tree for the UUV simulation consists of 4 levels

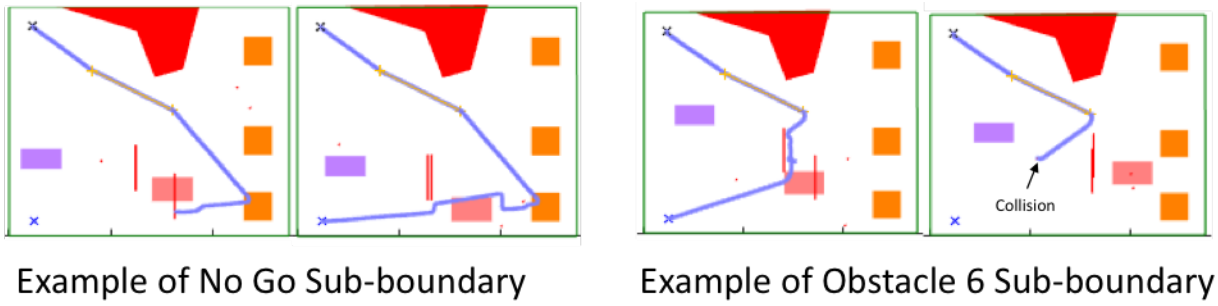


Figure 4.18: Example of two sub-score boundaries of the UUV simulation.

with 66 leaf metrics. Using a data-set containing 853K runs generated using NNDV sampling we were able to identify 153 distinct sub-cluster types. We can then reconstruct boundaries for any specific leaf metric such as a boundary in No Go violations or collisions with a specific obstacle as shown in Figure 4.18

4.3.5 Sub-clustering Performance

The key feature of our sub-boundary identification process is that it lets us discover all the sub-boundaries of the system without significantly increasing the computational time. To evaluate our new sub-boundary algorithm we took 863K runs generated by our UUV simulation and ran both our new and old algorithms against progressively deeper levels of the score tree. The results of this comparison are shown in Figure 4.19. Our new algorithm takes approximately 20 seconds to process a sub-score tree with 4 levels and 66 leaf metrics, which is comparable to applying our original boundary identification algorithm to 9 metrics simultaneously. Attempting to use our original boundary identification technique to all 66-leaf metrics simultaneously takes approximately 2 hours. Therefore, analyzing all of the score elements simultaneously using our new approach represents a 200-fold decrease

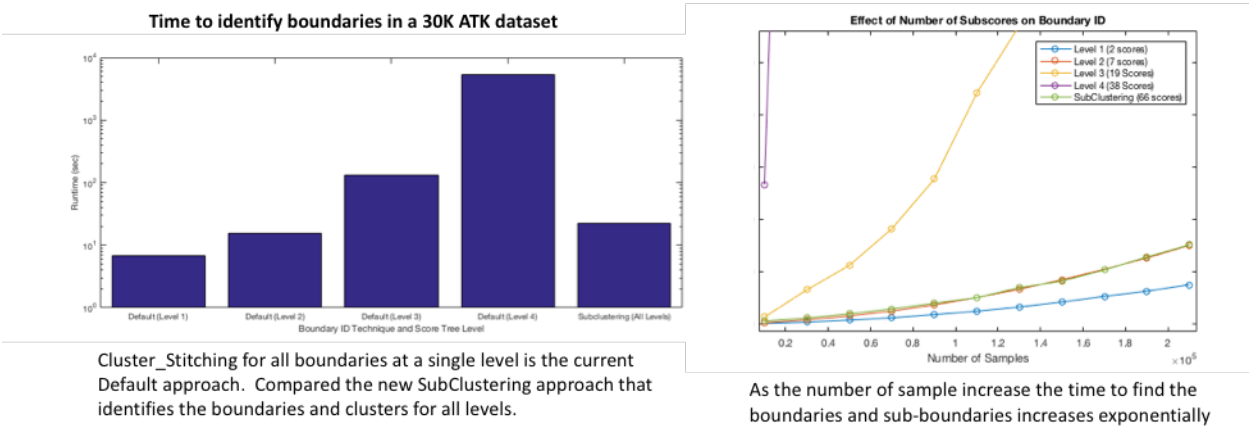


Figure 4.19: Timing comparison of the sub-boundary algorithm and previous algorithm. (Left) Bar chart showing the time to cluster a 30K dataset for increasing number of sub-scores with the new technique on the far right. (Right) Line plot showing the time to cluster a dataset for increasing numbers of samples. The new sub-clustering process is collinear with the results for clustering 7 scores simultaneously.

in the time compared to our prior approach.

The new sub-boundary algorithm successfully discovers all the boundaries that exist at all levels of the hierarchical score tree without significantly increasing the computational time. By applying our new algorithms, we can successfully reconstruct any sub-score boundary with minimal computational overhead. Even when we are dealing with data-sets with a large number of dimensions and nearly one million data-points.

4.3.6 Search Performance in the presence of sub-boundaries

While these sub-boundary techniques can be invaluable for finding previously unrecognized phenomenon in our system, we often have sparse coverage of these areas. In Figure 4.20 we show scatterplots of both the boundaries and sub-boundaries after applying our NNDV search method. One thing that is immediately apparent

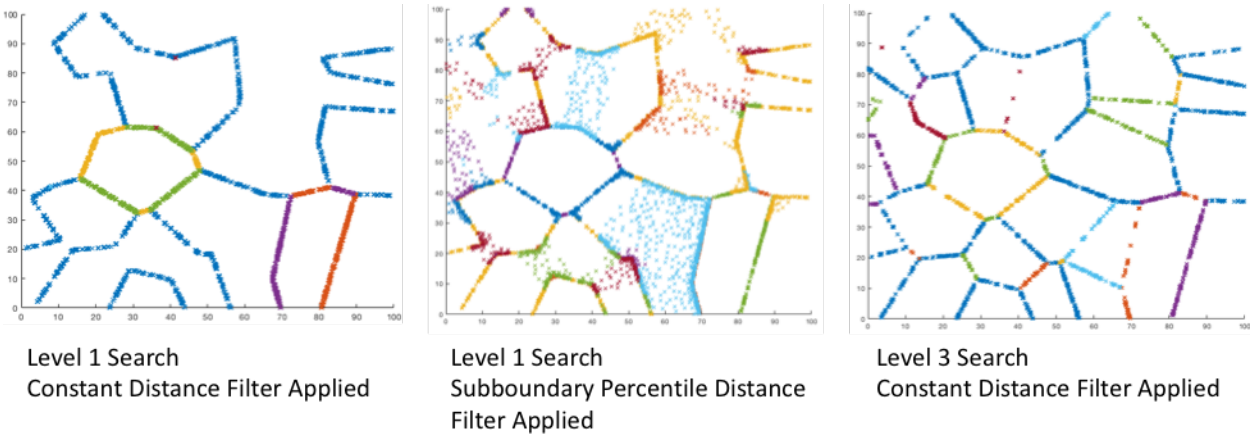


Figure 4.20: Scatterplots of the resulting boundaries from applying the adaptive search approach to just the root score metrics (Level 1) versus the leaf score metrics (level 3).

is that the distances between our sub-boundary pairs are large. This is because our NNDV algorithm is only operating on the root elements of our score-tree. It focuses all of its samples towards the regions of the root score boundaries and away from the interior regions where the sub-boundaries exist.

For this reason, our adaptive search approach can be detrimental when attempting to find sub-boundaries. As seen in Figure 4.20 when we apply our NNDV search to the Level 1 score elements we achieve high resolution sampling of the primary boundary while also having sparse sampling of the sub-boundaries. This can also be seen in our UUV example in Figure 20 where the sub-boundary pairs are somewhat dissimilar due to their distance from one another. This issue can be relieved by changing our search criterion such that we are searching over the level 3 elements of the score tree as seen in Figure 4.20. However this dilutes the search and does not work well when we scale the problem to higher dimensions.

What these results indicate is that we should take care when attempting to find

sub-boundaries from our NNDV generated data-sets. The low-resolution boundary pairs we achieve during a search of the root score elements can be used to inform the test engineer of the existence of the sub-boundary. If we want high-resolution of the sub-boundaries we must first choose the specific set of sub-scores we are interested in and then rerun the search using those score-metrics as input.

Therefore, we should consider the test design process to be an iterative approach with the following steps. First, we perform an adaptive search using the root score criterion. Second, we apply sub-boundary analysis to find test cases and boundary types of interest. Finally, we apply a second round of simulations in the regions of interest to obtain higher resolution of the selected sub-boundaries. This is the approach we will take in the next section as we discuss our test generation process.

4.4 Pre-Demonstration Analysis

In this section we will discuss how we applied our RAPT software to design and develop field tests which were executed in November 2017. We will describe the platform, how we adapted our mission to ensure successful recovery, analysis of simulation results, and the test scenario selection process.

4.4.1 Platform Description

The platform used was an OceanServer Iver2 unmanned underwater vehicle, Figure 4.21. The vehicle is approximately 60 inches in length, has a diameter of

Figure 4.21: A 3D rendering of the OceanServer Iver2 platform.

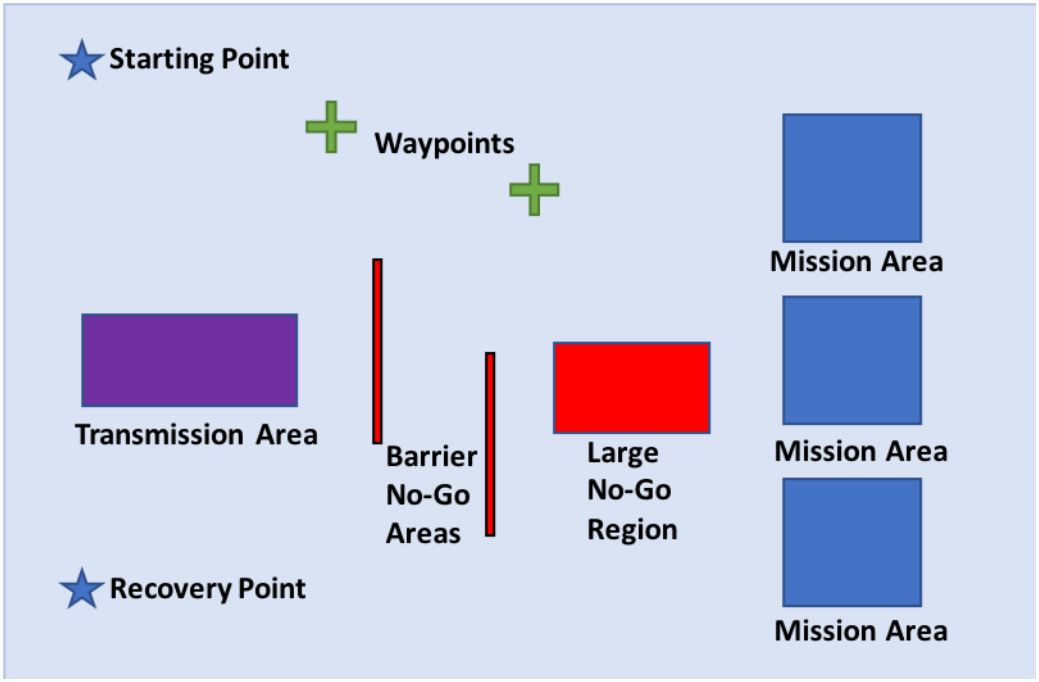


Figure 4.22: Diagram of the Phase 3 Mission

6 inches, and weighs approximately 60 pounds. It is controlled via a rear thruster and four external fins. The navigation sensors include a GPS, a Doppler velocity log (DVL), a compass, and an inertial measurement unit (IMU).

4.4.2 Mission Description

For this demonstration, the state space consisted of 13 variable parameters: (1) the start time; (2,3) the transmission window start time and duration; (4,5) the

latitude and longitude of the transmission area; (6,7) the latitude and longitude of the no-go region center; (8-11) the latitude and longitude of the 2 barrier no-go areas; (12) the priority order for the mission areas; and (13) the starting battery capacity for the IVER. These gave the representative cases of input states that have a strong effect on the autonomy, such as the tidal force imposed by the start time, and input states that have a weak effect, such as the priority order of the mission areas. The start time was varied in a 12-hour period between midnight and noon, the transmission window was set to open between 30 minutes after mission start time to an hour after mission start time with a duration between 30 minutes to an hour. Additionally, the transmission area was set to only vary in position on the western half of the operational area. The no-go region was set to vary only in position in the southeastern quadrant of the operational area. The no-go barriers were constrained to vary in position only within 400 meters of the operational area vertical centerline. And each scenario assigned a priority order for execution of the mission areas.

Additional state space element information is in listed in Table 4.1 below:

4.4.2.1 State Space Changes

The changes between the most recent state-space and the mission discussed in the previous chapter were made to speed up the search process and create scenarios that would be easier to execute on the water. One of the features of RAPT is the ability to analyze the contribution of each input variable to the resulting perfor-

mance modes of the system. We utilized this analysis along with our knowledge of the platform to make a more focused mission state space which would accurately represent what could be achieved on-water.

Our first change was to refine the state-space so that it only included elements which had significant impact on the system's performance. The original mission included a shoal obstacle located north of the waypoints, as well as three minor triangular obstacles. After performing a sensitivity analysis of all input variables, we concluded that the minor triangular obstacles did not significantly alter the behaviors frequently enough to justify the an additional 6 scenario parameters. This also prompted the addition of starting battery capacity as a state-space variable.

Our other changes were made due to constraints imposed by the testing range and hardware platform. The first was to make the operational area smaller and the scenario elements closer together. This allowed each scenario to be executed on-water in under 30 minutes. The second was to replace obstacles with no-go areas. We did so as there was no way to appropriately implement large obstacles in the water, and the IVER had no actual sonar sensor for detecting them. Therefore, the autonomy and simulation were modified to treat no-go areas as obstacles that could be detected by a simulated sonar sensor. The no-go areas, however, were still scored as their own category of performance. Also in the event the vehicle traveled into the no-go area, the simulation would continue the scenario allowing it to play out and RAPT would penalize the mission during scoring.

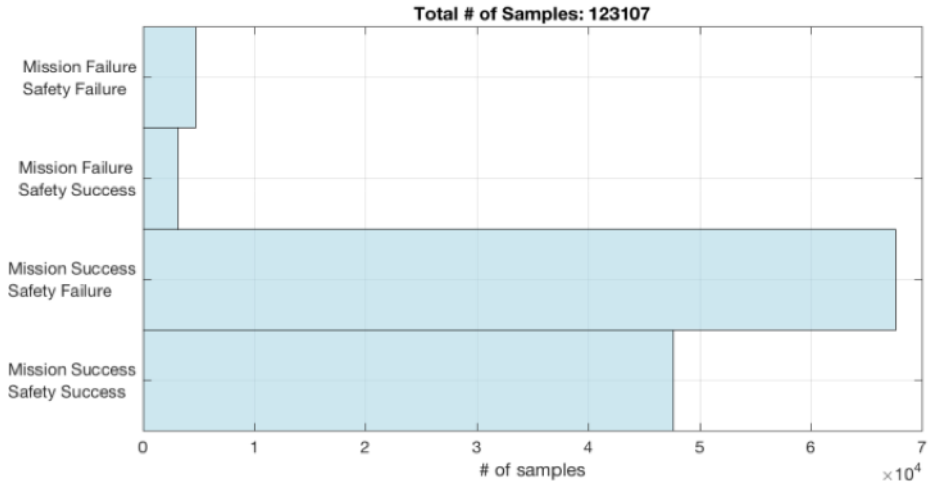
4.4.3 Scenario Generation and Pre-test Analysis

We generated 150k scenarios for the November Keyport Demonstration, allocating 30k to global search and 120k to adaptive search methods. Due to scenarios either being pruned for being invalid (e.g. no-go area generated on top of a waypoint) or to runs failing on the cluster, only 123k completed successfully and were available as part of the final dataset.

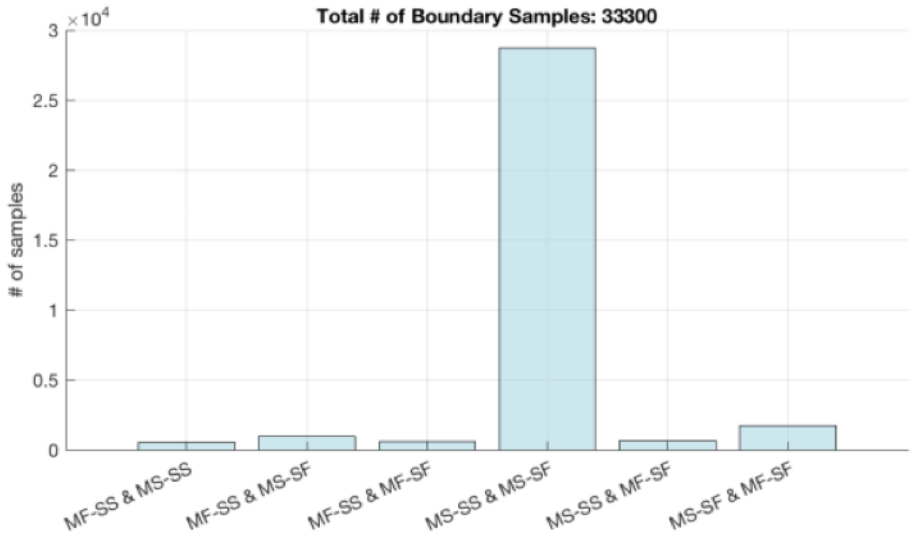
In this dataset the Mission Success performance mode dominated the scenario outcomes, with 93% of all scenarios successfully completing the Mission Area and Transmission Area objectives. In contrast, 57% of all scenarios failed on the safety criterion. This effect can be seen in Figure 4.23

The ease of the mission can be attributed to the reduced size of the operational area, which diminished the stress caused by the obstacle configuration and tidal current on the vehicle. As can be seen in Figure 4.24, the primary input variables which affected the vehicle's ability to complete the mission successfully were the timing of the transmission area and the amount of starting battery capacity. Once the vehicle had more than 16% battery capacity it nearly always completed the mission. The only exception was when the transmission window ended so early that it was impossible for the vehicle to arrive in time.

Although not surprising, it should also be noted that a majority of collisions occur near the waypoints and on the path between them, Figure 4.26. These result in scenarios where the mission objectives and the safety criteria create conflicts between reaching a waypoint and safely avoiding an no-go area. It also implies that

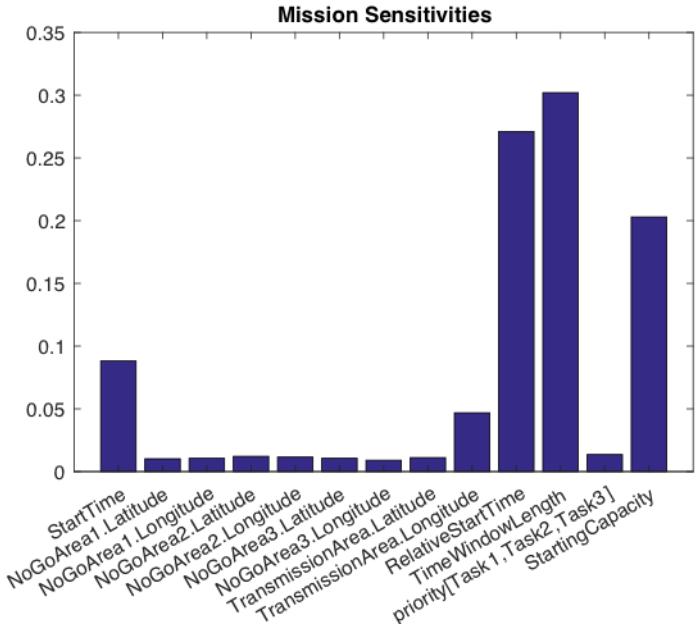


(a) Performance Mode Distributions

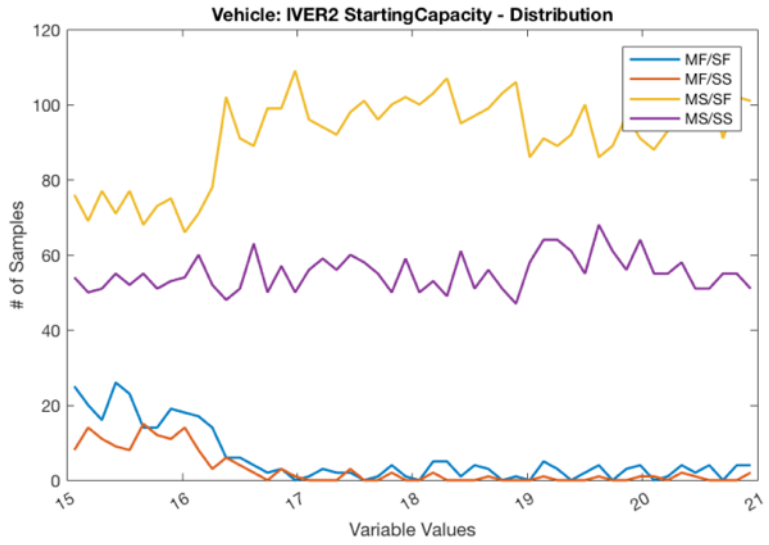


(b) Boundary type Distributions

Figure 4.23: Charts of number of completed runs in the pre-test dataset for each performance mode (above) and boundary type (below).



(a) Mission Parameter Response



(b) Individual Parameter Distributions

Figure 4.24: (Above) The estimated sensitivity response for each input variable on mission success. (Below) Distribution of performance modes for varying levels of starting battery capacity.

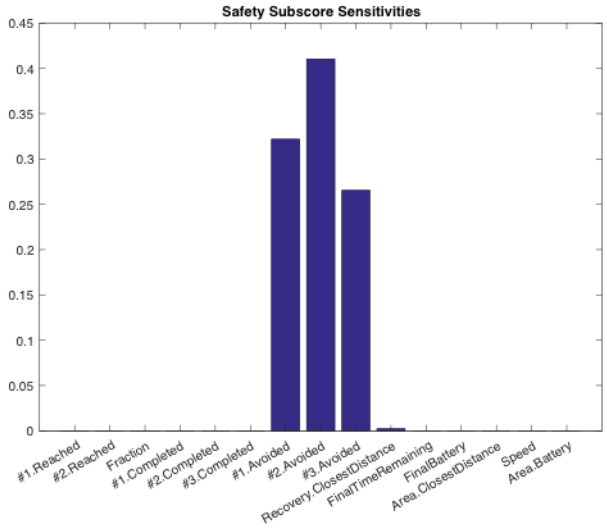


Figure 4.25: The estimated sub-score response for the safety success criterion

any time an no-go area is placed directly in the desired path the vehicle has a high probability of colliding.

In essence, this creates a "fuzzy" performance boundary where obstacles closer to the path result in higher collision probabilities due to deficiencies in the navigation software, errors in commanding the vehicle, deviations due to current, or inaccuracies in the estimated position. As scenarios approach the boundary, the uncertainty in the final score grows, with the scenario achieving either safety success or failure due to minor deviations in the trajectory, Figure 4.27. Thus, despite the prevalence of boundary pairs existing on a particular boundary, the majority of boundary pairs represent only minor variations in a trajectory rather than a change in the autonomy software's behavior.

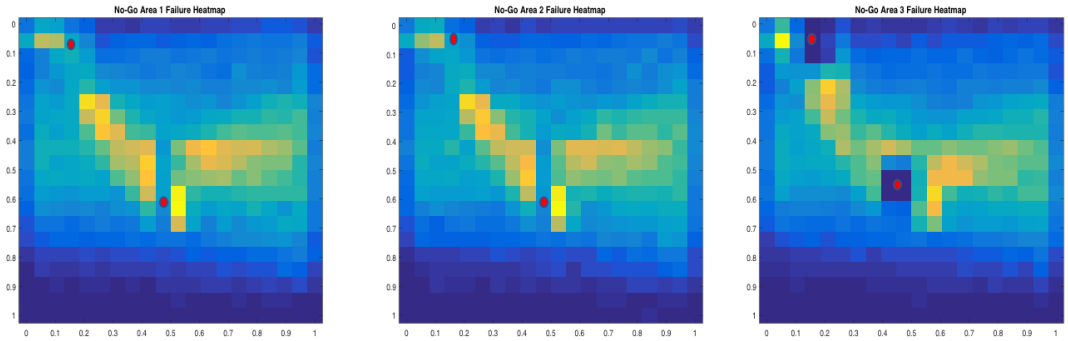


Figure 4.26: Heatmap showing which no-go area positions were most likely to cause collisions. Yellow is more probable, blue is less probable. Red markers indicate the positions of the waypoints. Cooler regions on top of the waypoints are result of these configurations being marked as "invalid" scenarios.

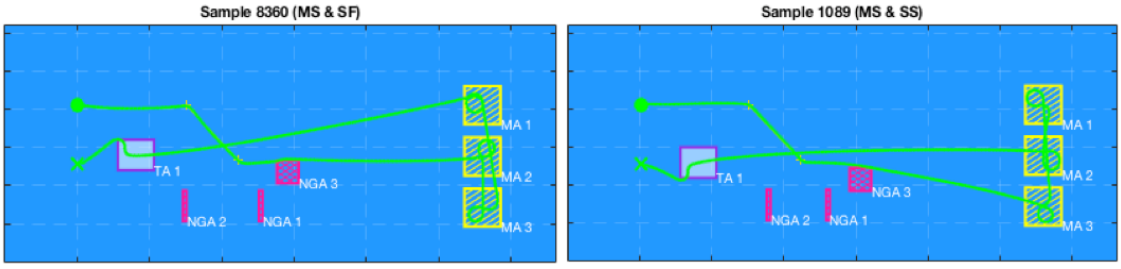


Figure 4.27: Example of boundary pair between safety success and failure, illustrating the "fuzzy" boundary.

4.4.4 Scenario Selection

Given the limited number of runs we could perform at the test-range we decided to create a testing suite of 10 scenarios consisting of 5 boundary pairs. Each boundary was designed to test a different behavior of the system as identified using our sub-boundary algorithm. We utilized sub-score boundary generation to test the sub-scores identified as the biggest contributors during the sub-score sensitivity analysis.

1. *Collision with No-Go Area 1.* This sub-boundary consists of the vehicle colliding with No-Go Area 1 when attempting to reach the first waypoint. In simulation if the no-go area was within a certain distance of the obstacle it would neither avoid the no-go area nor abort its attempt to reach the waypoint.

2. *Waypoint 2 Abort.* This sub-boundary demonstrates a major shift in the trajectory the vehicle takes depending on the distance between No-Go Area 3 and Waypoint 2. If No-Go Area 3 is too close to Waypoint 2 the vehicle will abort, otherwise it will attempt to reach Waypoint 2 but will collide with the no-go area.

3. *Mission Area 1 Abort.* This sub-boundary occurs within the individual mission area metrics and is meant to test how many mission areas will be completed during a single trial. Depending on the fuel levels and time windows the vehicle will either complete all the mission areas or abort early and return home.

4. *Return to Complete Mission.* This sub-boundary occurs when one of the mission areas has a much later completion time than the others. In one case the vehicle returns to complete the remaining mission area after traveling to the trans-

mission area. In the other it lacks the time and fuel to complete the last mission area and instead returns to the recovery point. It also included a placement of No-Go Area 3 that was considered to be particularly difficult.

5. *Early Transmission Area.* This sub-boundary occurs when the transmission area is completed before any of the mission areas. This can occur when the transmission window ends very early in the mission. This also results in a more efficient path which can mean the difference between mission success and failure.

Of all the sub-boundary types these 5 were considered to be the most relevant to the performance of the system and the most reproducible in the field. The remaining sub-boundaries were primarily permutations of the same behaviors (e.g. mission areas in different orders, collisions occurring at the same time as a mission abort) or were results of software faults which were based on non-deterministic effects. While the latter of these are an important phenomenon to test more thoroughly testing it in the water would require more time than we were allocated.

4.4.5 Dithering Study Results

As discussed in Section 4.3.6 our search techniques achieve high resolution sampling along the primary boundaries of our system (Mission Success, Safety Success) but low resolution in the interior regions where sub-boundaries occur. This means that we may have inaccurate estimates of how far our scenarios are from the desired sub-boundary or that there may even be sub-boundaries which we have failed to discover. Finally, uncertainty in the execution of the scenario may also

cause the simulated result to fail to match the real-world test. Therefore we need to explore the regions of the testing space near our selected test scenarios to ensure that we understand the level of uncertainty associate with each. As well as determining whether any of these scenarios exist on previously unidentified sub-boundaries.

To address these concerns we performed a dithering study of the selected test scenarios. In this study each scenario in the testing suite were dithered with Gaussian noise to generate an additional 2k scenarios in the neighborhood of test scenario. We then ran sub-clustering and sub-boundary identification scripts to identify all possible behaviors that could occur in the region of each boundary pair. The distances of the test scenarios to the boundaries discovered by the dithering study are listed in Table 4.2. For each boundary pair we provide the name of the sub-boundary type and the index for the score-tree element where the boundary occurs. For example our first boundary pair is a Collision sub-boundary which occurs on sub-score element 11, the metric which determines whether No-Go Area 1 was violated.

For a majority of the test scenarios we selected the dithering study revealed they were much closer to a sub-boundary than the original 120k dataset indicated. Boundary pairs 1 & 5 both resulted in the same sub-boundary type as indicated in the original set, simply with a closer distance. More concerning was the presence of previously undiscovered sub-boundaries in the vicinity of the other boundary pairs. We split these into two categories; orthogonal sub-boundaries and parallel sub-boundaries.

4.4.6 Orthogonal Sub-boundaries

Orthogonal sub-boundaries occur when multiple sub-clusters overlap and crossing one sub-boundary does not mean crossing the other. An example of this type of sub-boundary was found for boundary pair 2. The original sub-boundary we selected was the Waypoint 2 sub-score, which is element 15 in our sub-score tree. When we ran the dithering study we found that these test cases were also close to a Mission Abort sub-boundary, which is element 21 in our sub-score tree. Both the original boundary pair and the new sub-boundary pairs for these scenarios are shown in Figure 4.28.

In this example the dithering results indicated that the scenarios were incredibly sensitive to the perceived fuel levels of the system. If the vehicle had sufficient fuel it would return to complete the mission areas after surfacing in the transmission zone. Otherwise it would return to the recovery point immediately after transmitting the data. This decision to return to the mission is completely independent from the decision to abort the waypoint.

We selected this pair to test the sub-boundary caused by abandoning the waypoint, which still occurs even after the scenarios have been dithered. Therefore we were able to conclude that even if some bias occurred during test that we would still testing the correct sub-boundary. Albeit with the possibility that we would see trajectories similar to Dithering Scenarios (5922) and (89933) rather than the ones predicted by Scenarios 53451 and 55433.

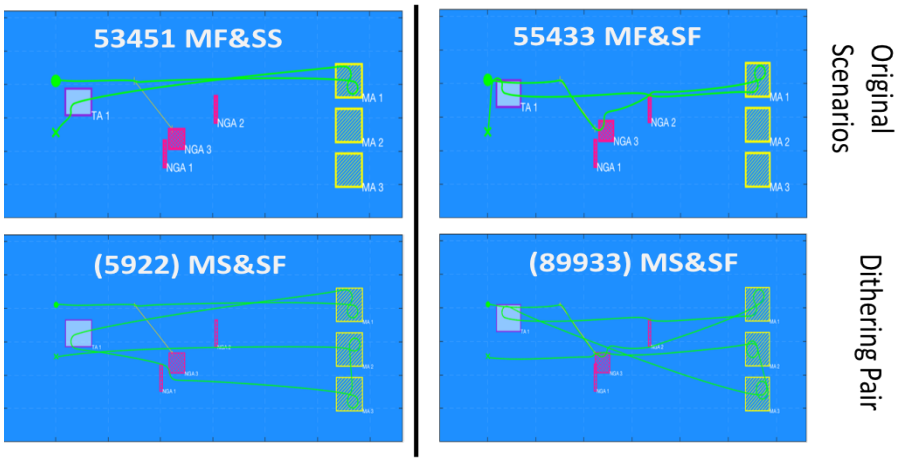


Figure 4.28: Example of orthogonal sub-boundaries discovered during the dithering analysis. The top two scenarios are the original boundary pair while the bottom two scenarios are the nearest neighbors discovered in the dithering analysis. While the right-left pairings provide an example of a Waypoint Abort sub-boundary the top-bottom pairings provide an example of a Mission Return sub-boundary.

4.4.7 Parallel Sub-boundaries

Parallel sub-boundaries occur when multiple thin sub-clusters border each other and their boundaries are aligned in the same direction in the state space. This is the case for boundary Pairs 3 & 4 which were selected to test two highly sensitive regions of our state-space. Our dithering study revealed that all of these boundary pairs were particularly close to the intended sub-boundary as well as nearby parallel sub-boundaries. Meaning that any bias in the execution of the scenario could easily push one of these cases across the boundary. The original scenarios and the dithering generated scenarios for boundary Pair 3 are shown in Figure 4.29

In this example a very small shift in the transmission window length causes the vehicle to complete anywhere from 1 to 3 of the mission areas. Our objective for this pair was to test the sub-boundary associated with the Mission Area 1 sub-score,

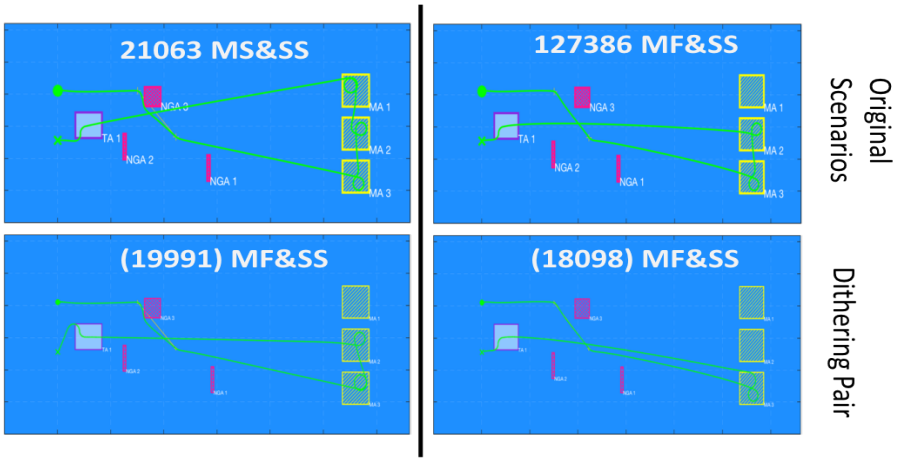


Figure 4.29: Example of parallel sub-boundaries discovered during the dithering analysis. The top two scenarios are the original boundary pair while the bottom two scenarios are the nearest neighbors discovered during the dithering analysis. These scenarios show examples from three different sub-clusters with varying levels of mission success.

which is element 26 in our sub-score tree. The dithering study revealed that not only is scenario 21063 substantially closer to that sub-boundary than we originally estimated but that scenario 127386 is adjacent to a parallel sub-boundary for the Mission Area 2 sub-score, which is element 28 in our sub-score tree.

These results indicate that the vehicle may exhibit behaviors from any of the sub-clusters represented in the dithering study for this pair. If the vehicle executes the mission more quickly or slowly than predicted in the simulation we may miss the sub-boundary entirely. Either testing the wrong sub-boundary, e.g. the mission area 2 sub-boundary, or both of the scenarios may exhibit the same behavior and no boundary will be demonstrated at all.

Therefore, in addition to running the selected scenarios for boundary pairs 3 & 4 we also developed modified versions of these scenarios which move the scenarios further apart in the state space. For boundary pair 3 this involved increasing the de-

creasing the transmission window length for scenarios 21063 and 127386 respectively. For boundary pair 4 this involved increasing and decreasing the starting battery for scenarios 61836 and 134243 respectively. Our test plan called for the unmodified scenarios to be executed first and if they did not match their predicted behaviors then the modified scenarios would be used in their place for any subsequent tests.

4.5 Demonstration Results

The Range Adversarial Planning Tool (RAPT) Phase 3 final demonstration was held in NUWC Keyport from November 14-17, 2017. Using the aforementioned JHU/APL assets of an IVER2 UUV running the JHU/APL Autonomy Toolkit for its surrogate autonomy.

The data-set utilized for the final test-plan and analysis consisted of 120,000 RAPT-generated scenarios. We were restricted to only 24 on-water tests for the field test period and thus were able to select 5 boundary pairs (corresponding test scenarios on opposite sides of a boundary) from the 120k runs for execution during the in-water portion of the demonstration. The intention was to run each scenario twice with backup runs reserved for scenarios where unexpected behaviors occurred.

When the on-water trials were performed at Keyport it was quickly discovered that the tidal files which were provided had almost no relationship to the actual currents experienced by the vehicle. In addition the IVER2 executed the scenarios at a faster speed than was commanded within the simulation. These two factors taken together meant there was a strong execution bias in the results where the

vehicle on the water behaved as if the time windows were later and the starting battery level was higher. Execution bias differs from stochastic effects as it causes all of the results to shift in a similar fashion. It can be treated as a translation of all of the performance boundaries in the testing space.

The rest of this section is devoted to a discussion of each of the boundary pairs.

4.5.1 Boundary 1 - No-Go Collision

This boundary pair tests the vehicle's navigation ability and how it handles waypoints near no-go areas and obstacles. In this boundary pair, the vehicle fails to abort a waypoint that is too close to a no-go area and ultimately violates the no-go region. This set of runs was an unmitigated success, both confirming the existence of the boundary and directly replicating the behavior of the simulation. A side-by-side comparison of the predicted results versus the on-water results are shown in Figure 4.30

4.5.2 Boundary 2 - Waypoint Abort

This boundary pair demonstrates the threshold at which the vehicle will abandon a waypoint due to the positioning of a No-Go Area. During the dithering study it was discovered that these test scenarios were incredibly close to another orthogonal sub-boundary, where the vehicle continues the mission after completing the transmission area. Execution bias in the on-water tests caused the vehicle to complete

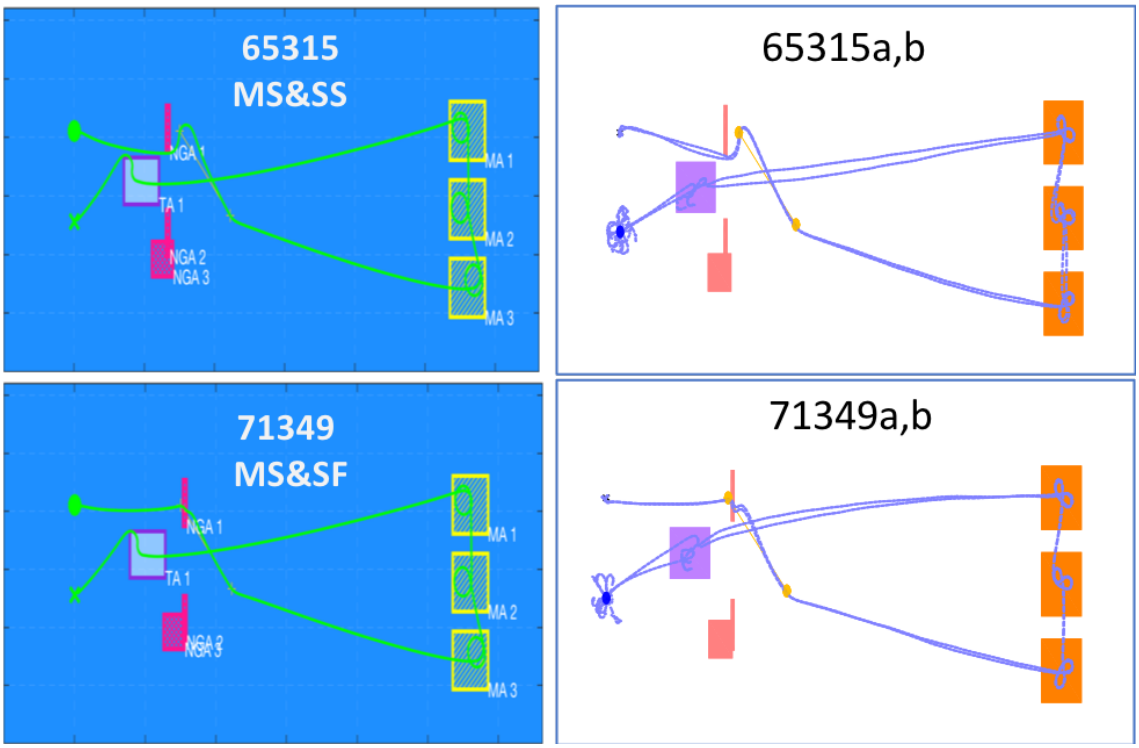


Figure 4.30: Boundary Pair 1 plots comparing the simulated result (left) to trajectory executed during in-water tests (right). Multiple blue lines are due to overlaying the results of all field tests on top of one another.

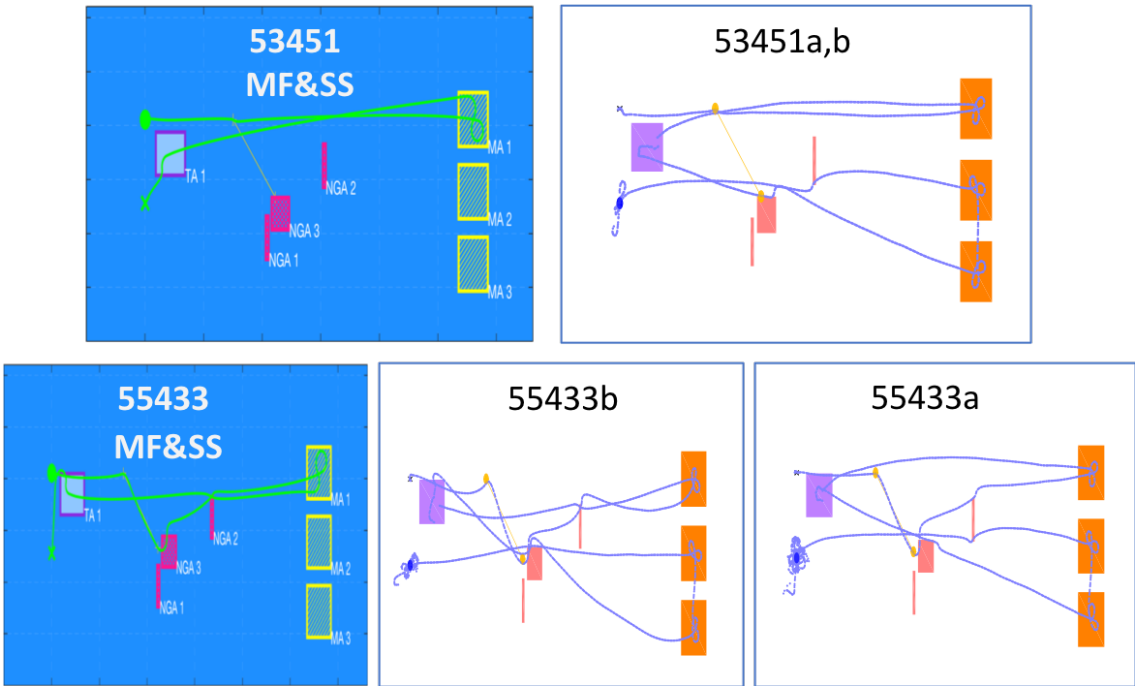


Figure 4.31: Boundary Pair 2 plots comparing the simulated result (left) to trajectory executed during in-water tests (right). Multiple blue lines are due to overlaying the results of all field tests on top of one another.

the mission more quickly and with more fuel than originally predicted. However this result was anticipated and these results still demonstrate the desired sub-boundary which occurs for the Waypoint 2 score element. A side-by-side comparison of the scenarios is shown in Figure 4.31

4.5.3 Boundary 3 - Mission Abort

This boundary pair demonstrates how the transmission window time affects transmission decisions by the autonomy. In the first scenario, the vehicle has plenty of time to complete the mission. In the boundary pair, the vehicle must return early to achieve the transmission window. In both cases the vehicle lacks the battery to continue the mission after completing the transmission window.

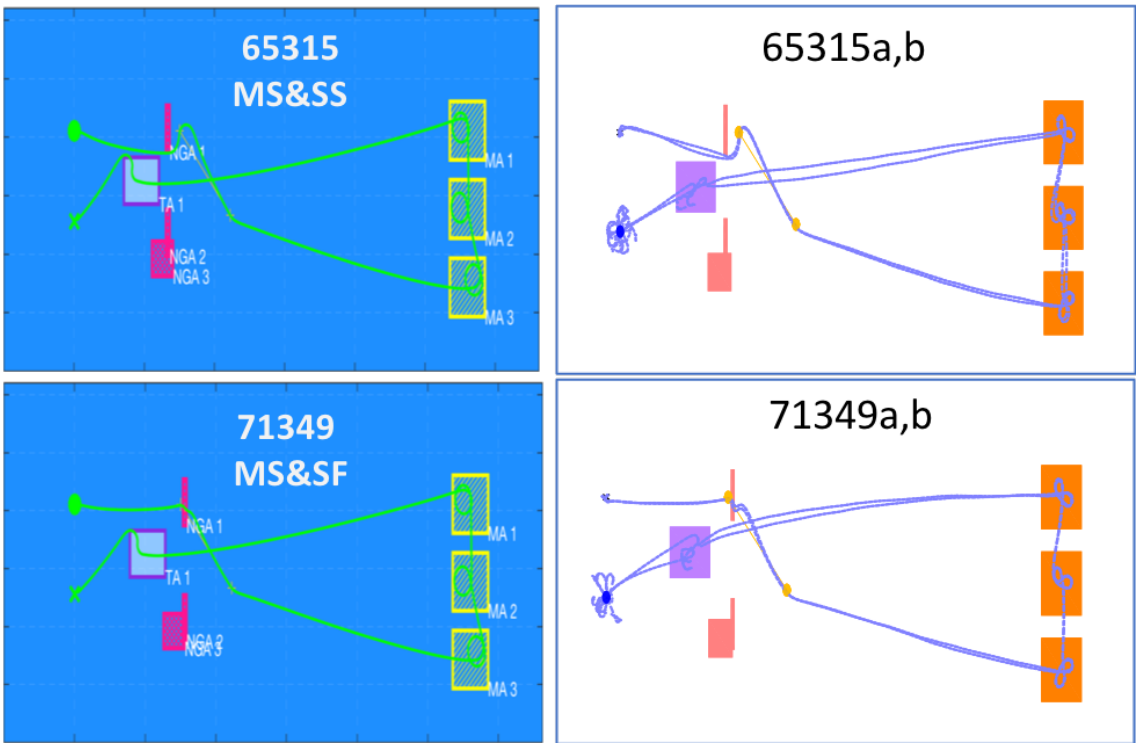


Figure 4.32: Boundary Pair 3 plots comparing the simulated result (left) to trajectory executed during in-water tests (right). Multiple blue lines are due to overlaying the results of all field tests on top of one another.

This boundary pair represents one of the cases where the dithering study indicated the final results would be uncertain. In all cases, the slower speed of the IVER in the water caused the vehicle to return to the transmission area earlier than predicted in the simulation. The scenarios designated with an "M" were modified manually to increase their distance from the boundary to achieve greater robustness. See Figure 4.32. In the case of scenario 21063, this meant increasing the length of the transmission window. Conversely, in scenario 127386 this meant decreasing the length of the transmission window.

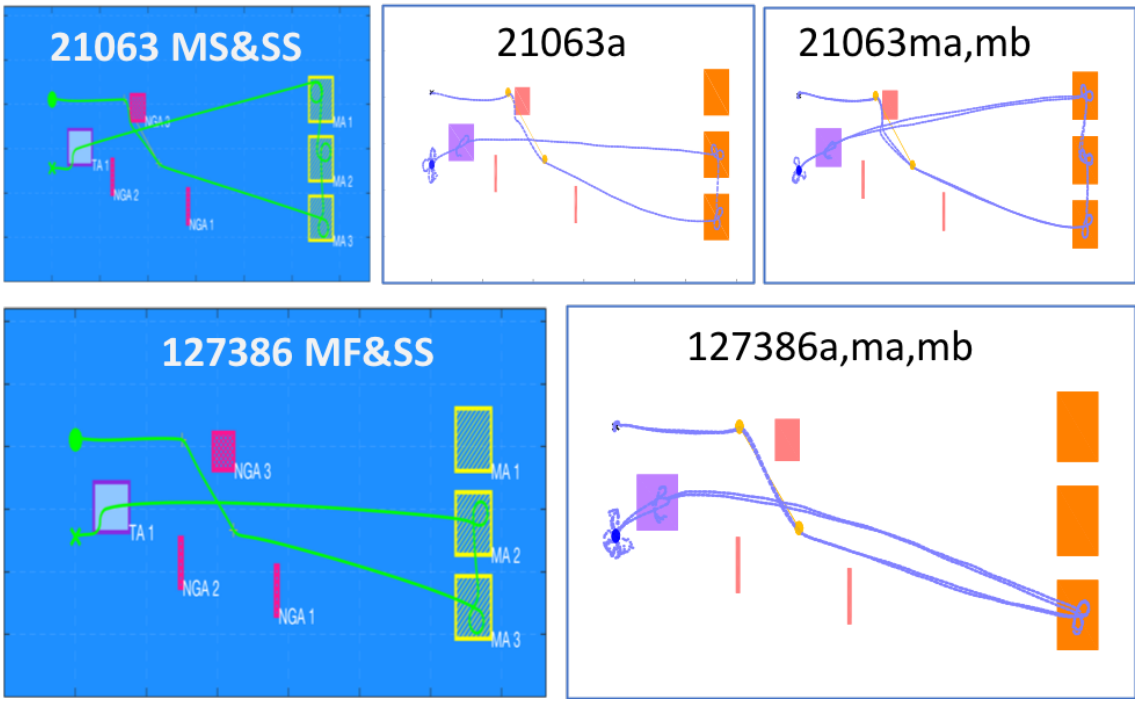


Figure 4.33: Boundary Pair 4 plots comparing the simulated result (left) to trajectory executed during in-water tests (right). Multiple blue lines are due to overlaying the results of all field tests on top of one another.

4.5.4 Boundary 4 - Mission Return

This boundary pair exercises the decision-making of the autonomy on whether to complete the mission after executing the transmission area. This is entirely determined by the starting battery level which is very close for both scenarios in the boundary pair. This boundary pair is incredibly close in all inputs, giving a good estimate of the threshold but also introducing risk due to noise and execution bias. A side-by-side comparison of the simulated result versus the on-water result is shown in Figure 4.33.

In this case execution bias cause the results to shift away from what was previously predicted. Since this possibility was predicted during our dithering study

of the selected runs we created a set of modified scenarios which are designated with an "M" in Figure 4.33 where the amount of fuel for scenario 13423 was reduced. However we underestimated the sensitivity of this boundary to both the transmission time and the starting fuel level and while the modification did move scenario 13423 to the correct side of the sub-boundary which we were testing it returned earlier than was predicted in the original simulation.

4.5.5 Boundary 5 - Transmission First

This boundary pair demonstrates vehicle behavior that will attempt the transmission area before completing all mission areas but always after the waypoints. In one scenario, this means that the vehicle achieves a total success since completing the transmission area first is the most fuel efficient. In the other, it fails the mission since there is not enough fuel to complete two full transits of the operational area. We were only able to perform a single run of this boundary due to unsafe weather conditions. This single pair managed to perfectly replicate the simulation results. A side-by-side comparison of the simulated result versus the on-water result is shown in Figure 4.34.

4.5.6 Demonstration Results Discussion

For all the scenarios tested we were able to successfully predict the behavior either during the initial search or during the dithering study. The execution bias meant that the performance boundaries predicted by the dithering study were exe-

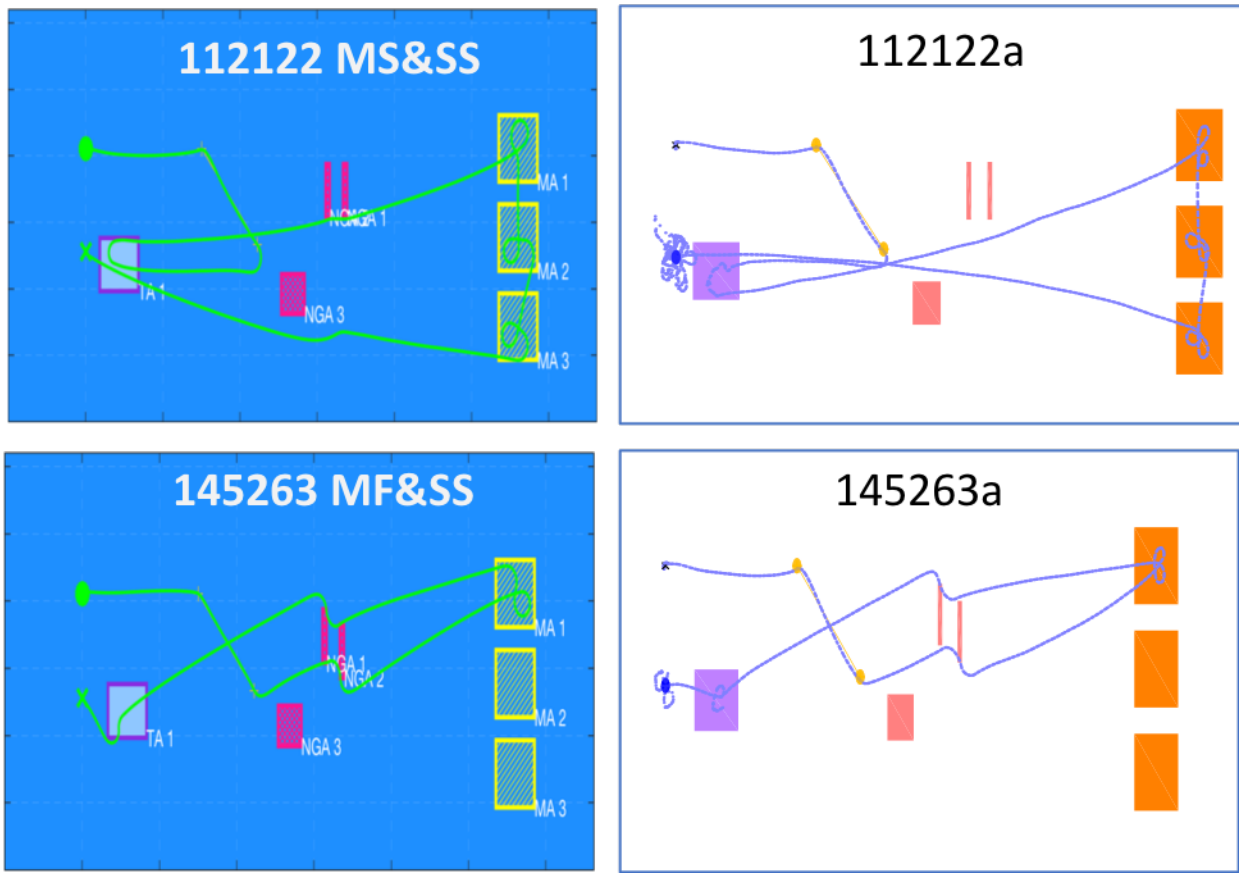


Figure 4.34: Boundary Pair 5 plots comparing the simulated result (left) to trajectory executed during in-water tests (right). Multiple blue lines are due to overlaying the results of all field tests on top of one another.

cuted instead of those predicted by the 120k dataset. For boundary pair 2 this meant that the behavior being tested, aborting a waypoint, was still executed correctly but the remainder of the trajectory executed on water differed from the simulation result. For boundary pairs 3 and 4 this execution bias meant that both scenarios would fall on the same side of the performance boundary. In order to account for this difference we had to manually adjust the scenarios to start with less fuel in order. Once this adjustment was made the original boundary was confirmed.

4.6 Discussion

Our objective for Phase 3 of the RAPT program was to demonstrate a TRL 6 system which could be deployed at NUWC Keyport and be used to generate test scenarios for in-water execution. At the November 14th -17th demonstration at Keyport we completed all of our stated objectives. We have successfully transitioned the software to Keyport and all components of test generation suite, simulation software, surrogate autonomy, and database are functioning as expected. This software was used to generate test scenarios which demonstrated different types of performance boundaries. These test scenarios were successfully executed in-water and validated the results of the simulation in that all the targeted behaviors were exhibited. Each of the research areas described in this Chapter contributed to this success.

There are several challenges which we encountered for the scenario generation and execution portions of the demonstration that are likely to occur in other domains. The first is a noisy performance surface which can throw off a search and

boundary identification process if the noise is too high. Despite this we were not only able to identify meaningful boundary cases but many of these cases were even closer to the boundary than originally predicted. The second was the transition of simulation results to in-water tests. Differences in both the dynamics and behavior of the hardware can skew the performance of the vehicle as was seen with the slower speeds on the IVER. However, by using a combination of highly refined sampling in the region of the selected tests and using sub-boundary analysis to identify a greater variety of performance modes we were able to predict the possible variations before execution in-water. As such we were able to execute the tests with confidence and the vehicle's performance on water aligned with our predictions.

The RAPT software suite achieved all of its objectives for a challenging problem, with the additional benefit of lessons learned that will guide our future efforts. The first lesson is that we should expect some amount of bias in the execution of our scenarios in-water. Developing an automatic way to identify and apply this bias to our predictions as we execute our tests would be a useful tool. The second is that there are many complex behaviors that can occur and identifying them requires careful processing of not just safety success and mission success but also all score metrics. Using a second pass of highly resolved tests and sub-boundary analysis was key to allowing us to predict changes in the autonomy's behavior before executing tests in-water.

This software suite is not just a tool for generating test scenarios but also a framework for analyzing the results and understanding the system under test. The software we delivered contains the tools and functions we found most useful for

both designing tests and processing the results. It is our hope that the users of the RAPT software suite will take the time to explore the capabilities of our software and understand that test design is an iterative process. The RAPT software suite enables not just the identification of relevant test cases but also provides information that can be used to design better tests, guiding a test engineer to the relevant regions of the state-space.

4.7 Summary

In this chapter, we took the framework introduced in chapter 3 and demonstrated the tests it generated could successfully be replicated in the field. There are two critical contributions that were necessary to complete demonstration. First, we established that our adaptive sampling algorithms were robust to the effects of uncertainty in the system under test. Second, we introduced a new sub-clustering algorithm which can identify boundary regions for a large number output dimensions simultaneously. Increasing the number of simultaneous outputs from 2 to 66.

Finally, this was the first study where search-based testing techniques were utilized to generate field tests. All prior research in this domain have stuck to strictly to simulation and only speculated on the possibility of transitioning to the real system. We not only predicted the behavior prior to execution on water but were able validate the existence of the performance boundaries on the hardware platform. The software has now been deployed at NSWC Keyport and will be used to design more UUV test suites in the future.

ACKNOWLEDGEMENT

This research was supported by the Department of Defense Test Resource Management Center under work contract W900KK-14-C-0004.

Element	**Description**
Waypoint	A recommended target to pass through if not constrained by other factors like no-go areas or time. It is not required for mission success.
Mission Area	A 500x500 meter region that the UUV must enter and remain inside for a predetermined amount of time. It must complete each mission area in the correct priority order. Completing all mission areas is required for mission success.
Transmission Area	A 700x750 meter region that the vehicle must enter and surface for the open transmission time window. Completing this objective is required for mission success. Surfacing outside of this time window or transmission area is a safety failure.
Large No-Go Area	A 400x500 meter region that the vehicle cannot enter. If the vehicle enters this region it will receive a safety failure but the simulation continues.
No-Go Barrier	If the vehicle collides with a barrier, it is considered a safety failure. The barrier areas are 700x40 meters.
Operational Area	A 3x3 kilometer region that the vehicle cannot leave. If the vehicle violates the boundary, it will receive a safety failure and the simulation ends.
Recovery Point	A target circle with a radius of 15 meters. The simulation ends when the vehicle reaches this point. If the vehicle does not reach this point, it receives a safety failure.

Table 4.1: State Space Element Details

Boundary Pair	Original Boundary Distance	Original Boundary Type	Scenario ID	Dithering Boundary Distance	Dithering Boundary Type
1	0.259	Collision (11)	65315	0.101	Collision (11)
			71349	0.256	Collision (11)
2	0.340	Waypoint (15)	55433	0.050	Mission Return (21)
			53451	0.042	Mission Return (21)
3	0.247	Mission Abort (26)	21063	0.050	Mission Abort (26)
			127386	0.042	Mission Abort (27)
4	0.2526	Mission Abort (35)	21063	0.052	Mission Abort (35)
			127386	0.073	Mission Abort (35)
5	0.2950	Transmission (8)	112122	0.054	Transmission (8)
			145263	0.073	Transmission (8)

Table 4.2: Dithering Results

Chapter 5: Adaptive Sampling as a validation method for UGV self-righting

5.1 Introduction

While much effort has been devoted to tip-over avoidance [156], robots operating in unknown, dynamic environments are likely to experience tip-over at some point during their operational lifetime. As most robots require a particular orientation for mobility, a single tip-over event can result in mission failure if the robot is unable to recover. Therefore, the ability to self-right is imperative for mission critical applications such as military [157] or law enforcement [158] operations, urban search and rescue (USAR) [159], [160], and planetary surface exploration [161]. Further, a tool that can independently assess and verify a robot's ability to self-right under various circumstances could be extremely valuable for agencies seeking to design, evaluate, and/or compare robots for such critical missions.

Previous approaches for self-righting have typically used hand-designed trajectories for specific robot morphologies [162]–[164]. These plans use a combination of active reorientation (i.e. using their actuators to push them into unstable configurations), followed by passive rolling to complete proper reorientation. Controlling the

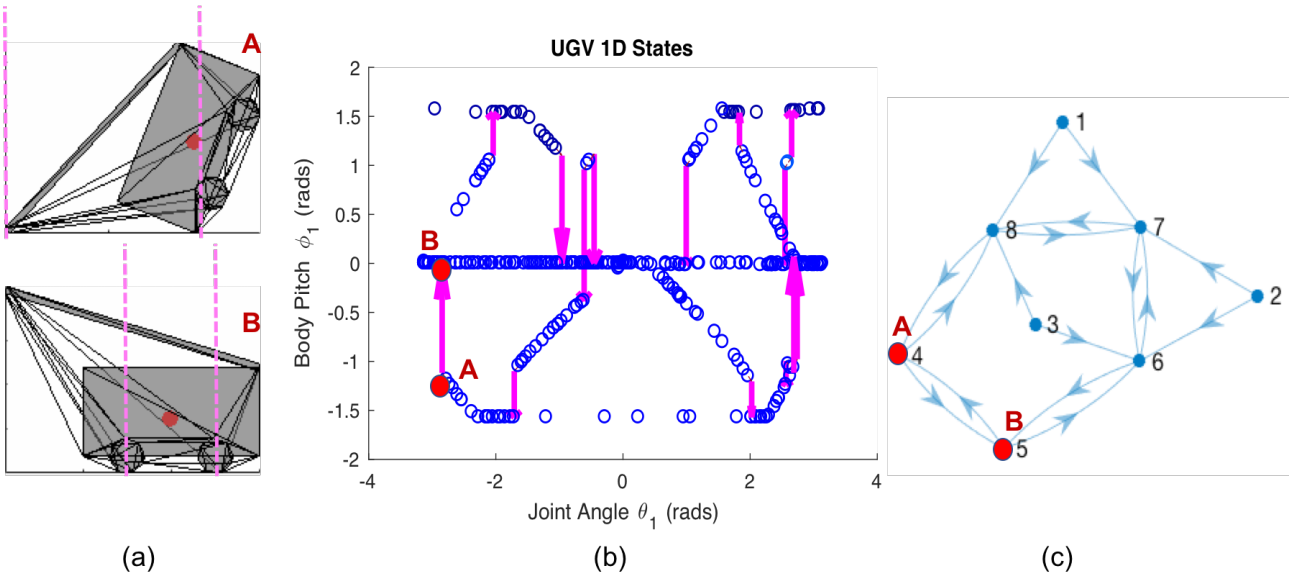


Figure 5.1: Overview of the stability framework. (a) Illustration of a robot with a single arm and 1 DOF tipping over (b) A state plot of the robot where blue markers represent states in C-space and magenta arrows showing where transitions occur. (c) A Node plot is created by compressing the stable regions of the state space into nodes and their transitions into edges.

impact of a polygonal object and a plane has been solved for a variety of dynamic conditions [165] but has not been explored as a potential path for self-righting. Finally, while there are a variety of general motion planners that consider stability constraints, it has been to prevent tipping rather than to exploit it [118], [156]. What has not been addressed in prior research is a general motion planner which can handle the type of hybrid locomotion seen in self-righting plans.

Therefore, we seek to develop a general framework which supports the generation of self-righting plans for any robot morphology which can be defined using a rigid body model. Further, we want the capability to quantify any given robot's ability to self-right under varying circumstances. Rather than focusing on determining a single optimal self-righting trajectory between two states, we want to discover

how much of the robot's configuration space (C-Space) is capable of self-righting. This would allow the retrofit of existing or future robots while also providing insight to improve future designs.

To achieve this goal, we build upon a previously developed framework [166] for generating self-righting road-maps, which was successfully applied to robots with 1, 2, and 3 degrees of freedom [167] in a 2-dimensional environment. This sampling-based approach has similarities to Probabilistic Road-maps (PRM)[168], a popular motion-planning approach that builds a graph of the connected regions of C-space through the use of a constraint checker and local planner. Where the self-righting framework differs is its focus on finding and exploiting unstable transitions between stable regions of C-space thereby creating hybrid directional graphs which encode the necessary motions for both active reorientation and passive rolling.

The issue with prior approaches is that both grid-based sampling and random sampling are insufficient to find the critical transitions of the system. As the number of degrees of freedom (DOF) increase, the number of samples required to search the space at the same resolution increases exponentially. Lowering the resolution can lead to incorrectly identifying the transitions and failing to properly estimate the motion necessary to initiate the transition. Therefore, we require two new technologies to support a sampling-based framework for self-righting. The first is a technique that provides high resolution samples in the critical regions where transitions occur while only sparsely sampling the stable regions of C-space. The second is a graph generation technique that can both identify the stable regions of C-Space and the transitions between them.

In this chapter, we introduce new methods of adaptive sampling and generating connectivity graphs of the configuration space. These are based upon algorithms we had previously developed for identifying critical testing cases for underwater vehicles. [142] These techniques bias sampling towards regions of the system where sharp transitions in the output occur, allowing us to achieve higher resolution in the areas of interest while not wasting samples in areas that have already been characterized. Using these algorithms, we are able to address some of the scaling issues we encountered in the prior self-righting approach [166].

However, neither of the previous approaches could be directly applied to the analysis of robotic morphologies in three-dimensions. Achieving our objective of analyzing high-fidelity robot models in 3D required three critical changes in our approach: (1) We had to update our framework to support 3D dynamics and high-resolution robot models, (2) We needed a new method for identifying connected regions of configuration space, and (3) we had to change our query generation process to support constrained sampling of manifolds in high dimensional space. The final result of these improvements is a system which can take a 3D rigid body model of any robotic system and generate a self-righting transition graph spanning C-space using significantly fewer queries than previous grid-based or random-sampling approaches.

5.2 Self-Righting Framework

In this section we describe the self-righting framework initially developed in [166] as well as how it was updated to work with our new simulation and sampling

method. This framework evaluates the self-righting capability of a robot by creating a state space map and analyzing its connectivity. To create the map and identify transitions, the framework iterates through all possible joint configurations (previously using grid-based sampling with uniform resolution), determining the convex hull of the robot for each configuration. Since we assume sloped planar ground, only points on the convex hull of the robot can contact the ground. Identifying the convex hull produces a list of faces, which we define as the body rotations required to align that face with the ground normal. We then assess the stability of each face by determining whether the robot's center of mass would be located between the extents of the support face, if the robot were resting on the given face. A visualization of this assessment can be found in Fig. 5.1(a,b). The red dot is the robot's center of mass, and the extents of the support face are the vertical dotted lines. When the robot's center of mass is no longer contained by the the support polygon in state A, it transitions to state B.

By iterating through all possible joint configurations and faces, transitions where a face changes from stable to unstable (or ceases to exist on the convex hull) are identified. When a face is found to be unstable, the resulting stable face on which the robot would come to rest is determined. Next, sets of continuously stable states whereby the robot remains supported on the same face are grouped into "nodes". These nodes form a directed graph with edges corresponding to transitions. An example of these graphs is depicted in Figure 5.1(c). Since most robots are sensitive to physical shocks, we chose the change in potential energy across a transition as the cost metric for the node graph. In this manner, we are able to generate path

plans using the usual tools, and also assess the robot's ability to self-right for various ground angles by examining the reachability of the goal from the rest of the graph. Further, this analysis can be performed a priori using large computational resources, but be distilled into directed graphs with low memory requirements and requiring little computational resources for on-board use.

For this work we had to improve this framework in two significant ways. The first is that in the original framework, the contact face was described as a pair of unique vertices. However this proved to be insufficient for our purposes. As the collision geometry for the robot became more complex, it could potentially have thousands of unique faces which, in addition to number of joints, causes the search of this space to become intractable. To solve this, we no longer track unique faces, but rather query states based on body angle and find the face most nearly parallel to the ground given that body angle. This allows us to treat nearby faces as similar for the purposes of guiding our search.

The second change is that the original framework marked any states that were adjacent in the grid search and on the same resting face as connected. Since the number of possible resting faces were small and easily computed, these also sufficed for identifying the nodes of the system. However, due to the varying distance between states in our adaptive search, we needed an unsupervised method for identifying connectivity between states. We also needed methods for identifying the nodes of our system given our use of continuous body angles to describe the faces of the convex hull. Thus we developed new clustering algorithms for identifying the manifolds of the system and the boundaries between them.

5.3 Problem Formulation

In section 5.2, we discussed our general framework for analyzing a robot's ability to self-right based on connectivity among regions of continuous stable states. Note: throughout this chapter we define stability to mean quasi-static stability. The robot is quasi-statically stable whenever the center of gravity is within the bounds of the support polygon and the robot has zero angular velocity. Here we define the problem more formally for the purposes of our adaptive search method.

5.3.1 Definition of terms

(i.) The robot joint state space Θ^m defines the range of possible joint configurations for a robot with of m degrees of freedom. A joint state is defined as $\theta \in \Theta^M$

(ii.) The robot body angle state $\phi \in SO(3)$ which defines the orientation of the base frame for the robot. We define $Z_\theta = [\phi_{\theta,1}, ..., \phi_{\theta,k}]$ as the set of body angles for a convex hull with k different faces. This hull is defined by robot's collision geometry for the joint configuration θ. In addition, we denote a set of body angle sets $Z^N = [Z_{\theta_1}, ..., Z_{\theta_N}]$.

(iii.) A robot configuration state is defined as the vector $x = [\theta, \phi]$ where $\phi \in Z_\theta$. This is the combination of a joint configuration and body angle which is given to the stability analysis function. We denote the configuration space-space as $x \in \mathcal{X} = [\Theta, SO(3)]$. We denote a set of configuration states as $X^N = [x_1, ...x_N]$.

(iv.) A robot stability function $\mathcal{F}(X^N) = [Y^N, W^N]$. It accepts a set of N estimated states X^N and returns sample set of N output stability vectors Y^N, as well as N output information vectors W^N. The binary stability output $y_i \in [0, 1]$ has a value of 0 when the state x_i is unstable and 1 if it is stable.

(v.) We define the information vector as $W^N = [\phi_S^N, \phi_R^N, Z^N]$. Where ϕ_S^N are starting body angles; ϕ_R^N are resting body angles; and Z^N defines the convex hull faces for each state. These are discussed in more in Section 5.4.2.

(vi.) A *node* is a region of continuous stability on an m dimensional manifold in the configuration state space $\mathbb{R}^{m+3}$ where m is the number of joints. Two states x_1 and x_2 are members of the same node if they exist on the same manifold and there exists a path between them where all points along the path are stable.

(vii.) A robot transition graph G_N, where each node in the graph represents a continuous region of stability and each edge represents a transition between these regions.

5.3.2 Problem Statement

Our problem is: given a robot stability function $\mathcal{F}$ and the state space $\mathcal{X}$, generate the transition graph G that groups regions of continuously connected stable states as nodes and tracks the transitions between them as edges. From this we can determine how many of the possible starting states can reach the desired upright home state x_{home}.

Our goal is to demonstrate two primary improvements over the previous framework. The first is to improve sampling efficiency over the base-line grid approach. This will be measured using the following metrics.

Query Efficiency. This is measured as the number of queries to the simulator required to fully explore the system.

Edge length. This is measured as the distance in joint space between a pair of states. It represents the minimum amount of actuation necessary to move between those states.

Transition Energy. This is defined as the amount of potential energy lost by the robot when it enters an unstable state and tips over. It is measured by the change in height of the center of gravity and is a measure of the physical shock inherent in the transition.

Our second goal is to show that this framework can be used for self-righting in three-dimensions for robots models with a high number of facets and degrees of freedom. We will demonstrate this by applying our system to a high-fidelity model of an actual bomb-defusing robot with 5 DOF. For this case-study, we measure the effectiveness of the search and the resulting transition graph on the following metrics; the number of samples for each search method, the percentage of states which can reach the home state of the robot, and the mean total transition energy required to reach the home state. This last metrics is the sum of the transition energies for all edges in the graph between the starting state and home state. A

lower total transition energy means less powerful impacts as the robot tips over during the self-righting process.

5.4 Approach

5.4.1 Overview of Approach

Our objective is to discover all node transitions with the same or better resolution as the previous grid-based approach, while utilizing fewer queries to the simulator. Our approach iterates upon adaptive sampling techniques originally developed to generate test scenarios for autonomous vehicles [142]. This framework breaks the problem into two steps; adaptive sampling and boundary detection. During the adaptive sampling step we try generate queries and submit them to the system in an attempt to maximize samples in the transitional regions of the system. During the boundary detection step we attempt to cluster the contiguous regions of stable C-space and identify how they are connected via unstable transitions.

The states where the convex hull formed by the robot has a face resting on the ground exist on a discrete set of m dimensional manifolds in $\mathbf{R}^{m+3}$ space. For maximum efficiency we only want to submit queries to the system along these manifolds. However, we do not know a priori the number or shape of these manifolds. In addition we require a way to extract clusters of samples that share the same manifold in order to generate our transition graphs.

To achieve this, we have adapted both of our techniques to use additional information output by the simulation. The first is that our stability function does

not return results for the queried states. Instead, it returns a set of starting states along with the outputs of the simulation for those starting states. Any starting state which does not result in a face of the convex hull resting on the ground is rotated onto the closest face to the ground plane. This step is similar to a projection operation [169] which is commonly used for sampling motion plans with constraints. This allows us to utilize estimates of the manifolds as inputs to our system and and the stability function will attempt to correct any error in our estimate so we still receive an output as if we had submitted a valid manifold state. A valuable feature as it reduces the total number of times we must compute the convex hull for potential joint angles.

Our second change is to utilize save all convex hulls that have been computed with each query. We use these to create a set of all known combinations of joint states and body angle states for our clustering algorithms. Creating a larger set of points which can be used for manifold identification than using the queried states alone.

Our final change is to account for the fact our C-space is defined in joint angles and body angles. We adapt our distance metrics to be the cosine distance between two states, defined as $dist_{cos} = |cos(x_0) - cos(x_1)|$.

We take this approach rather than using more standard manipulation motion planners such as AtlasRRT or CBiRRT2 [115][117] for two reasons. The first is that these algorithms do not provide any bias that would lead it to discover the "bridge" or transitional states between the manifolds of the system. The second is that we are attempting to achieve a global road-map rather than simply plan between two

selected states.

5.4.2 Robot Stability Function

We developed a 3d simulation environment which uses the MATLAB Robotic System Toolbox to simulate the dynamics of the rigid-body on a sloped planar surface. The collision geometry of the robot is defined using Unified Robot Description Format (URDF) to define the rigid-body properties and STL files to define meshes of each individual link.

This simulation takes a robot joint configuration and body angle and outputs the valid faces of the convex hull formed by that configuration and whether the input body angle is stable. We are only interested in body angles which have the possibility of stability, which are those where a face of the convex hull is in contact with the ground. To ensure this, we "snap" the robot to the nearest face of the convex hull before simulating the dynamics and only consider stability from this state.

To turn this simulation into a queryable function for our adaptive sampling methods we need to wrap it such that it could accept a vector of states X^Q as input and return as output a sample set $\mathcal{L} = [X, Y, W]$ where X is the vector of starting states that were utilized by the simulation, Y is the the vector of binary stability values for each of the starting states in X, and W is a vector containing information about the starting pose, ending pose, and convex hull. This allows us to use an adaptive sampling approach as if we had a system with $\mathbf{R}^{m+3}$ inputs and a binary

Algorithm 5 StabilityFunction(X)

Input: A set of states X^Q
Output: A set of samples $\mathcal{L}$
  $[\Theta, \Phi^Q] = X^Q$
  for each $\theta_i \in \Theta$ **do**
    Compute vertices for the robot $P_i = RobotModel(\theta_i)$
    Compute vertices and face normals for the convex hull $[P_{Ci}, V_{Ci}] = ConvexHull(P_i)$
    Compute body angle set $Z_{\theta,i} = Vec2Euler(V_{Ci})$
    Find $\phi_{S,i} \in Z_{\theta,i}$ which minimizes $dist(\phi_i, \phi_{S,i})$
    Rotate the robot such that its body angle is $\phi_{S,i}$
    Simulate dynamics until the robot is stable
    Set $\phi_{R,i}$ as the final body angle
    $X^\frown x_i$ where $x_i = [\theta_i, \phi_{S,i}]$
    $Y^\frown y_i$, where $y_i = (distance(\phi_{S,i}, \phi_{S,i}) \leq \epsilon_\phi)$
    $W^\frown w_i$ where $w_i = [\phi_{S,i}, \phi_{R,i}, Z_i]$
  end for
  return $\mathcal{L} = [X, Y, W]$

output and automatically corrects for any error in our estimate of the position of the valid manifolds. The formal definition for this function is given in Algorithm 5.

5.4.3 Adaptive Sampling

Adaptive sampling is an iterative process consisting of submitting queries to the system, using the returned outputs to generate a meta-model, and then applying an information metric to the meta-model to generate a new set of queries. The typical application of adaptive sampling is to minimize error between the meta-model being built and the true system. However, our goal is to generate samples along the transitions between the stable manifolds of our system.

Here, we tailor our adaptive sampling framework from [142] to our new system in several ways. This new approach is described in Algorithm 6. First, we assume that the test function $\mathcal{F}$ outputs feedback as to the starting states and their stability values. Therefore, instead of constructing our labeled data set from our queries X^Q, we instead construct it from the starting states X' output by $\mathcal{F}$. The second change

Algorithm 6 AdaptiveSampling($\mathcal{F}, \mathcal{X}, N$)

Input: A function $\mathcal{F}$, a state space $\mathcal{X}$, and a maximum number of queries N
Output: A set of labeled samples $\mathcal{L} = [X^N, Y^N, W^N]$

```
Randomly sample a set of initial queries X^0 ∈ X from a uniform distribution
Evaluate initial queries [X^N, Y^N, W^N] = F(X^0)
while size(X) < N do
    Train M on [X, Y]
    X^C = generateCandidates(X, X, W)
    [I, σ, V] = M(X^C)
    Remove x_i ∈ X^C where σ_i = 0 and V_i < r_max
    Remove x_i ∈ X^C where σ_i > 0 and V_i < r_min
    if X^C = ∅ then
        BREAK
    end if
    Set X^Q to the n elements of X^C with the highest I
    [X', Y', W'] = F(X^Q)
    Concatenate samples, X^N‖X', Y^N‖Y', W^N‖W'
end while
return L = [X^N, Y^N, W^N]
```

is that we have made the generation of the candidate states X^C into a modular component that allows us to use a domain specific algorithm tailored to $\mathcal{F}$. The third change is that while we are still driving our search using the binary stability output Y, we allow additional information to be saved in the vector W. Finally, we introduce cutoff conditions of minimum and maximum resolution $[r_{min}, r_{max}]$ of the search. The search ends when all volumes that lie upon the transition boundary are smaller than the maximum resolution, and all other volumes are smaller than the minimum resolution.

We utilize interchangeable meta-model evaluators designated as $\mathcal{M}$ for selecting which candidate states X^C to select as queries. A meta-model evaluator trains a meta-model of the function $F(X) = Y$ using all previously labeled samples $[X^N, Y^N]$. It then computes an information gain metric I along with a density V and variance σ for all candidates states. Currently we utilize a K-Nearest Neighbors Density and Variance (NNDV) estimation method, which was designed to sample the transition regions of systems. The procedure for the NNDV evaluator, $\mathcal{M}_{NNDV}$,

Algorithm 7 GENERATECANDIDATE($\mathcal{X}, X, W$)

Input: A state space $\mathcal{X}$, a set of known states X, and their respective body information W
Output: A set of candidate states X^C

$[\Theta, \Phi] = X$
$[\Phi_S, \Phi_R, Z] = W$
Train convex hull meta-model $\Pi(\Theta, Z)$
Generate random set $\Theta^C \subset \Theta$
$Z^C = \Pi(\Theta^C)$
$X^C = [\theta_i, \phi_j], \forall \theta_i \in \Theta^C$ and $\forall \phi_j \in Z_i^C$
return X^C

is as follows:

1 **Input** a set of sampled states $[X^N, Y^N]$, and query states X^Q

2 Find the K nearest neighbors in X^N using cosine distance

3 Compute variance for each state $\sigma_i = Var(Y_i^K)$

4 Compute volume for each state $V_i = max(|X_i - X_j^K|)$

5 Compute information $I = \alpha * \sigma + \beta * V$

6 **return** $[I, \sigma, V]$

We use α and β as weights to tune the search. A high α biases the search towards sampling regions with high variance, exploiting the evaluator's estimate of where the boundary is likely to occur. A high β biases the search towards sampling regions with high volume, exploring regions that have not been sampled before.

As mentioned in the beginning of the section we do not know the number or shape of the R^m manifolds a priori. There is no known analytical representation for many types of constraint manifolds (including pose constraints) and the high dimensional C-spaces of most practical robots make representing the manifold through grid-based sampling sampling prohibitively expensive.[117]. While we can

Algorithm 8 TRANSITIONGRAPH($\mathcal{L}$)

Input: A sample set $\mathcal{L} = [X^N, Y^N, \Phi_S^N, \Phi_R^N, Z^N, J^N]$
Output: A transition graph G_N
Create set of all known states X^Z
Create graph $G_0 = gabrielGraph(X^Z)$
Set $Y^Z = Y^N(nearestNeighbor(X^Z, X^N))$
Create $G_S \subset G$ by removing all nodes where $y_i = 0$
Create set of clusters $[L, L^Z] = stronglyConnected(G_S)$
Create transition edges E_T
Convert $E_T = \{[i, j]\}$ into $E_L = \{[l_i, l_j]\}$
Remove duplicates from E_L, keeping the lowest energy edges
return $G_N = (L, E_L)$

compute the location of the manifold for any given joint state θ, it requires calculating the convex hull which is computationally expensive. This means we must estimate where the manifolds are when generating our candidate states X^C and rely on the projection step within $\mathcal{F}$ to correct for any error in our estimates.

Past efforts for constrained sampling have researchers in the past have created parametric or piecewise continuous models of the manifolds [170] [115] to drive their search. Here we do the same, by training a meta-model of the convex hull function $\hat{Z}_\theta = \Pi(\theta)$, which takes a vector of joint angles and returns a set of the predicted body angles of the convex hull faces. To minimize computational overhead, we selected a nearest neighbor classifier for Π such that $\Pi(\theta_i) = Z_j$ where θ_j is the nearest neighbor to θ_i in Θ^N. We more formally describe this component in Algorithm 7

5.4.4 Graph-based Clustering

In our previous work [142] we developed a boundary identification technique which relied upon Mean Shift clustering to find the performance modes of the target system. However, that algorithm is not appropriate for identifying contiguous stable

regions of C-space. Therefore, we developed a novel set of graph-based clustering algorithms. The purpose of these is to identify the stable manifolds that represent the resting faces of our system, then identify the transitions between them to create our self-righting graph. This is analogous to the connection step of the PRM process, except we are eschewing the use of a local planner to generate validated connections between states and instead using trying to determine connectivity via identification of the manifolds.

In order to identify the manifolds of our system we utilize all known valid points that are identified in our generated map $[\Theta^N, Z^N]$ to create X^Z where $X^Z_{ij} = [\theta_i, \phi_j] \forall \theta_i \in X^N, \forall \phi_j \in Z_i$. This gives a significantly denser set of points of which our labeled states X^N is a subset, filling in the regions where the adaptive search explored more sparsely.

We then construct a Gabriel graph of these points. The criterion for two points $x_i, x_j \in X$ being connected via a Gabriel graph are $d(x_i, x_j) \leq \sqrt{d^2(x_i, x_k) + d^2(x_j, x_k)}$ and $d(x_i, x_j) \leq d_{max}$ where $k \neq i, j$ and $x_k \in X$. We select d_{max} as our maximum edge length criterion. This requirement removes any outliers and prevents connections between parallel manifolds.

Once we have our initial Gabriel graph G_0, we assign each the stability values in Y^N to each node in X^N. We then assign values to the remaining elements in X^Z based upon their closest neighbor in X^N. Next we create the subgraph of stable states G_S by removing all nodes where $Y^Z = 0$. Finally, we identify all strongly connected regions L in the subgraph and assign each a unique label $l_i \in L$ to form our clusters, with L^Z being the label for each node in G.

These clusters constitute the nodes of our transition graph. The next step is to generate the directional edges of our transition graph which are unstable transitions between the continuously stable nodes. We do this by creating $E_T = \{[i, j]\}$ where $x_i = [\theta_i, \phi_{i,S}], x_j = [\theta_i, \phi_{i,R}]$ $\forall xi, xj \in X^N$ s.t. $y_i = 0$. In other words connecting each unstable state in X^N to the node representing its final resting state.

For the transition graph we want to compress each strongly connected region in G_0 into a node and use the edges in E_T to define the transitions between them. We do this by converting $E_T = \{[i, j]\}$ into $E_L = \{[l_i, l_j]\}$, replacing the node indices with their cluster labels. As this can create duplicate edges between nodes, we remove all duplicates except the one with the lowest transition energy. This leaves us with the final graph $G_N = (L, E_L)$. The full process is detailed Algorithm 8.

The advantage of using graph-based clustering instead of using a local planner to perform edge checks is a reduction computational time. To compare these two techniques we created a simple local planner which checks the linear path between two states for stability. We then applied the PRM algorithm for connecting states in C-space using our linear local planner.In Table 5.1 we show the results for applying both techniques to sample sets generated our 1,2, and 3 DOF robots.

		Gabriel Clusters	PRM
	# of states	Time (sec)	Time (sec)
1DOF	443	0.162	2.34
2DOF	1,686	1.5809	33.44
3DOF	16,964	481	2,249

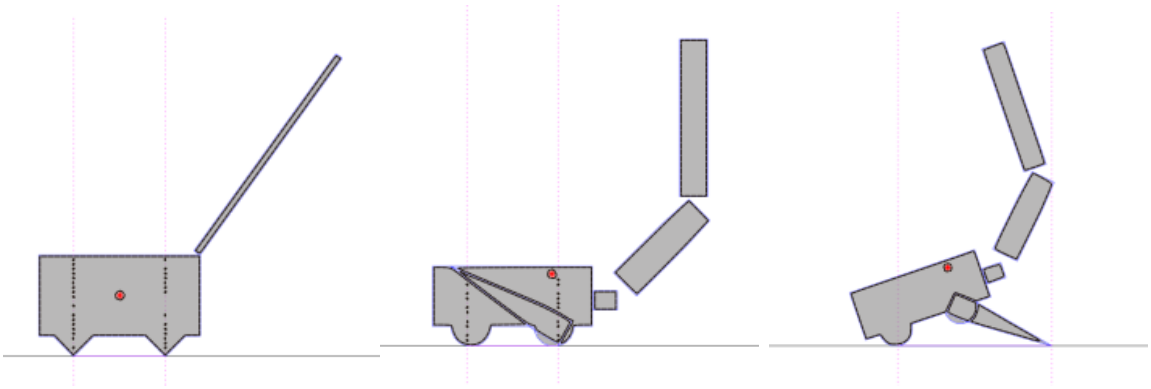
Table 5.1: Comparison of computational time between our Gabriel clustering method versus graph generation using PRM edge checking

5.5 Results

5.5.1 Comparison with Previously Validated Results

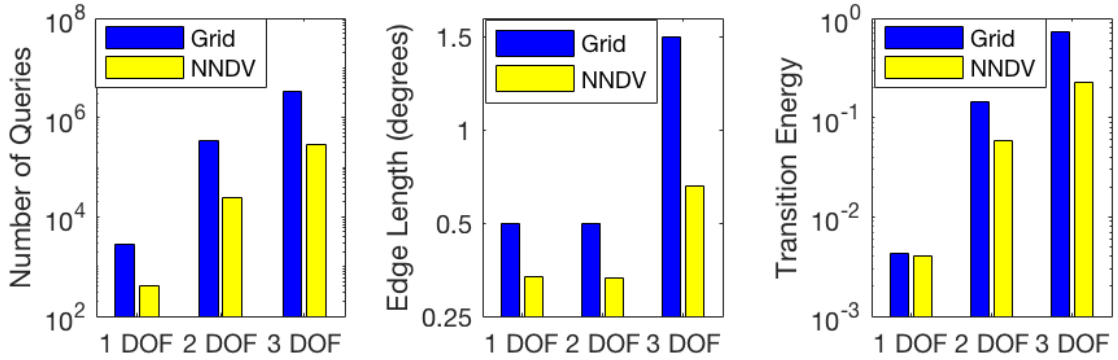
The self-righting framework was previously validated on three systems consisting of 1, 2, and 3 degrees of freedom (DOF) [167]. In this section, we compare the results of applying our newly introduced search techniques to those previously published. For each system, we measure the number of queries to fully explore the space, the mean distance between boundary pairs, and the mean energy of the identified transitions.

The systems tested consist of a 1 DOF system with a single shoulder joint attached to a massless arm (Fig. 5.2A), a two DOF system with a shoulder joint and elbow joint with a fifth of the robot's mass residing in the arm (Fig. 5.2B), and a three DOF system that is identical to the two degree system but has an additional flipper located in-line with the front wheels (Fig. 5.2C). The 1 DOF robot was run with a ground angle of 20 degrees, while the physically realized 2 and 3 DOF



(a) 1DOF (b) 2DOF (c) 3DOF

Figure 5.2: Illustrations of the 1, 2, and 3 DOF robots



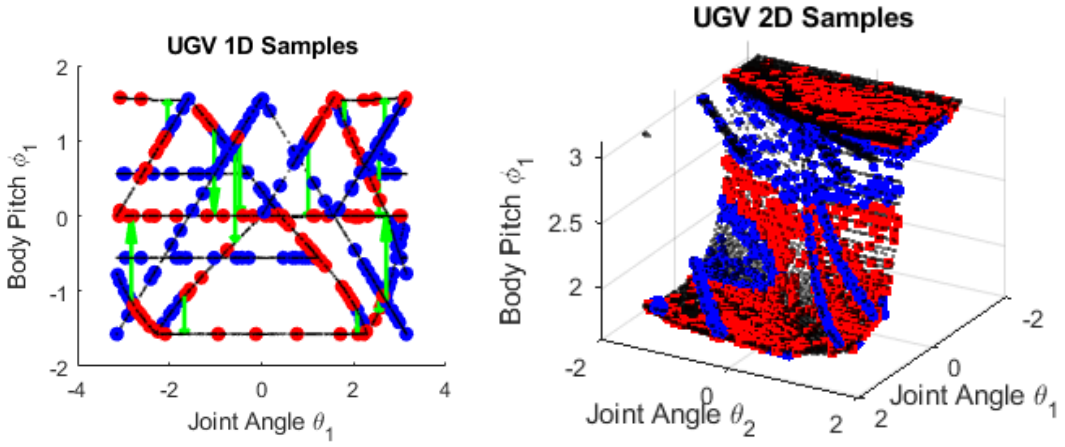
(a) Number of Queries (b) Length (c) Energy

Figure 5.3: Comparison of the three search methods for the validated systems

robots were analyzed for a ground angle of 0 degrees. As the previous study only considered the dynamics in a two-dimensional plane, we fixed the roll and yaw to zero in our simulation for the comparison.

For comparison, the grid search was run at 1 degree resolution for the 1 and 2 DOF system, and 3 degree resolution for the 3 DOF system. The results of this comparison are shown in Figure 5.3.

For all of the systems, both the grid and adaptive searches resulted in the same node transition graphs which had been previously validated on hardware. See Figure



(a) 1DOF (b) 2DOF

Figure 5.4: State plot comparing the grid search and adaptive search. The black lines indicate the samples generated from grid search; the blue makers indicate stable samples and red markers indicate unstable samples queried by the adaptive search.

5.1c for a visualization of the 1DOF graph. The adaptive search technique required an order of magnitude fewer queries to fully explore the system. For the 2 DOF system this reduced the total simulation time from 550 seconds to 75 seconds, with the additional time attributed to computational overhead. As seen in Figure 5.4, the adaptive search was successful at preferentially sampling near the boundary regions between nodes while minimizing the number of samples in the interior regions.

The higher resolution in the transition regions meant it was able to identify boundary pairs that were closer together. Which meant more confidence about the actual transition point. These pairs also had a lower transition energy, in other words less energy had to be dissipated due to impact with the ground. These qualities mean that we could potentially reduce the number of samples required even further if we are willing to accept higher impact transitions that are associated with lower resolution in the transition regions.

5.5.2 Case Study: AEODRS Increment 1 Robot

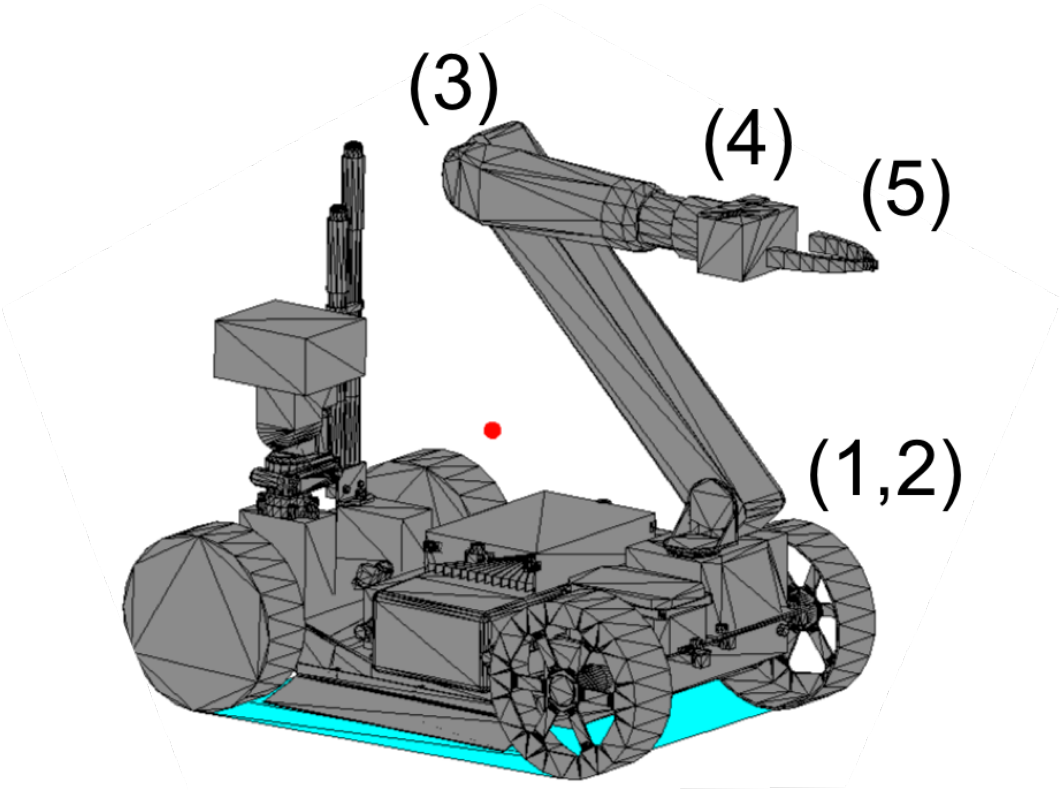


Figure 5.5: Illustration of the 5 DOF robot model. Its degrees of freedom are (1) Shoulder Yaw, (2) Shoulder Pitch, (3) Elbow angle, (4) Wrist rotation, (5) Jaw Angle

As this improved framework now supports the analysis of three dimensional models we were interested in how these techniques would apply to a robot where both rolling and pitching is part of the self-righting process. For this purpose, we selected a robot from the Advanced Explosive Ordnance Disposal Robotic System (AEODRS) family of platforms. AEODRS is a US Navy-sponsored effort to create open standards for bomb defusal robots. The ability to self-right is one of the many requirements of the final system [171]. The Increment 1 robot is illustrated in Figure 5.5 and has a manipulator arm with 5 degrees of freedom. It has a total

of 9 degrees of freedom if its wheels and gimbaled camera system are included but neither of those are considered part of the self-righting process for this study. Of the 5 degrees of freedom provided by the manipulator and end effector we were able to show that only 3 of these (shoulder pitch, shoulder yaw, and elbow angle) are necessary for self-righting. We analyzed the robot as a 1, 2, and 3 degree of freedom system. As proceeding to any higher dimensions using the grid search required more computational time than we could reasonably allow. The 1DOF version only utilizes shoulder pitch, and the 2DOF version utilizes shoulder pitch and shoulder yaw. In both of these configurations, the elbow was locked at full extension (180 degrees). For all of these analyses, we only considered a ground angle of zero, allowing the effects of yaw in the body angle to be ignored.

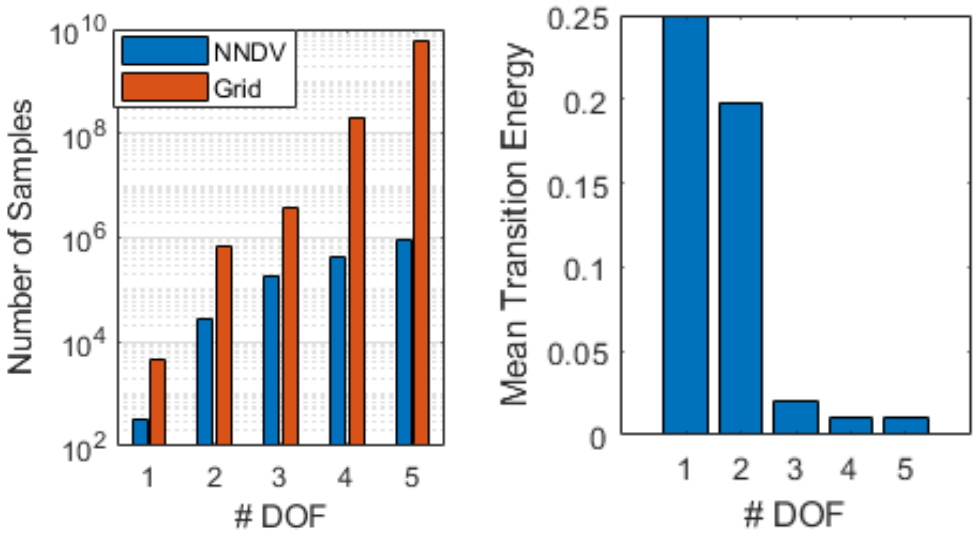
The results of applying our algorithms to AEODRS robot are shown in Figure 5.6. However in this case we did not actually run the grid-search as it would have been prohibitively expensive. Instead we used the manifold meta-model generated by the adaptive search to estimate the number of queries required by the grid search to achieve the same transition resolution. As with the previous validations, the adaptive sampling search proved to be significantly faster at generating the transition graph. This is not surprising as the previous-grid search approach required a number of queries equal to $||Z_\theta|| * (\pi/r)^m$, where $||Z_\theta||$ is the average number of unique faces for a single joint angle and r is the sampling resolution. For the complex geometry of the AEODRS robot most configuration states had approximately 200 unique faces. Combine this with the exponential number of samples required as the number of joint angles increases and it becomes immediately apparent that grid-

sampling cannot be applied beyond 2 or 3 dimensions. For example at 4 dimensions a grid-search with 5 degree resolution would take over a month to complete. What is interesting is that the adaptive sampling approach does not exhibit the same exponential growth at 4 and 5 dimensions. This is primarily due to the fact that the wrist and gripper jaws have almost no effect on the self-righting capability of the robot. Thus the adaptive sampling method was able to ignore those dimensions to a large degree and still create the appropriate self-righting path.

In addition, we discovered that 78% of states were self-rightable for the 1 DOF system and 100% of states were self-rightable for 2 DOF and above. However, when all three joints were utilized, we could find a stable transition directly to the upright configuration from almost any state as indicated in Figure 5.6b as the transition energy drops to nearly zero. The states for the 1 DOF setup are shown in Figure 5.7. It is immediately apparent there are 1 dimensional manifolds in the 3 dimensional space. Where the robot undergoes both changes in roll and pitch as the shoulder joint moves.

5.6 Summary

In this chapter, we introduced a simulation-based framework for evaluating the self-righting capabilities of a robot in a 3-dimensional environment. The primary intellectual contributions were a manifold constrained NNDV search and new methods for graph generation. By using adaptive sampling we achieved the same or better resolution of the transitional regions as the prior work with an order of



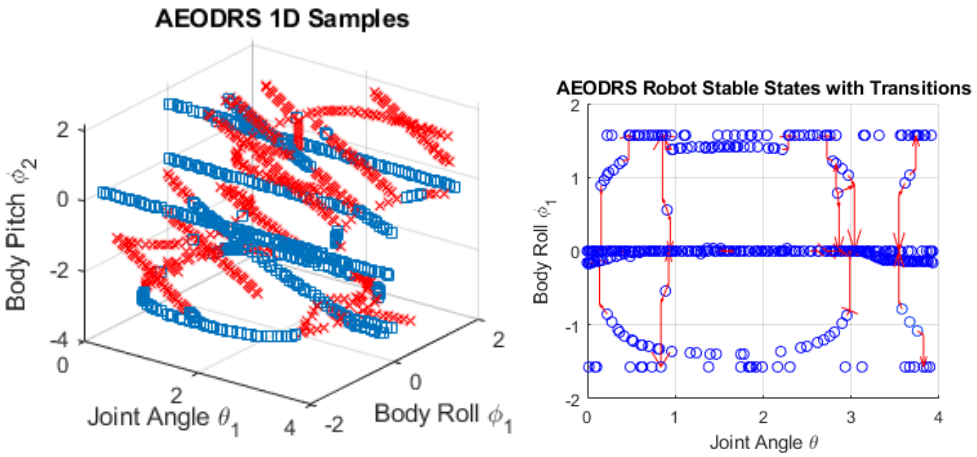
(a) Number of Samples (b) Energy

Figure 5.6: Charts showing (a) the number of samples required to fully explore the AEODRS C-space and (b) the resulting mean transition energy it takes to reach the home state for increasing degrees of freedom

magnitude fewer samples. This is a critical factor in our ability to scale the system up to systems with higher degrees of freedom, as the number of samples required scale exponentially as the number of dimensions increase.

In addition, our graph-based clustering methods for identifying contiguous regions of stable space were more computationally efficient than the PRM approach of validation using a local planner. They also support our new approach of representing our system as stable manifolds in C-Space which can be reached through unstable transitions. The result was an increase both the number of degrees of freedom and number of geometry facets for the robot models. Significantly increasing the fidelity of the robot models which could be verified.

Our test system for this study assumed a quasi-static model for robot stability and the simulator used a coefficient of restitution of 1. The black-box treatment



(a) Stability Plot (b) Transition plot

Figure 5.7: Scatter plots for the 1DOF AEODRS robot with only the shoulder joint actuated. (left) An isometric view showing how stable states in blue and unstable states in red. (right) A top down view with red arrows showing transitions between nodes.

of the simulation by the sampling algorithms means we can change these assumptions by updating the model in our simulator. Similarly, it would be straightforward to add additional complexity to the simulation environment in the form of rugged terrain. Two aspects for future work that will require changes to the framework are torque limits and dynamic limb motions. Torque limits can cause our assumption of full bi-direction connectivity within the stable manifolds to break down, requiring an additional step to be applied to generate valid road-maps within these manifolds. Dynamic limb motions have been addressed by previous research into individual transitions but add an extra level of dimensionality which is not currently incorporated into our planner.

Chapter 6: Accelerated Testing and Evaluation of Autonomous Vehicles via Imitation Learning

The work in this chapter was published in the following venues,

G. E. Mullins, A. G. Dress, J. D. Appler, P. G. Stankiewicz, and S. K. Gupta, "Accelerated testing and evaluation of autonomous vehicles via imitation learning," in *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, IEEE, 2018

6.1 Introduction

Correctly predicting how an autonomous vehicle will behave in new scenarios is an unsolved problem within the testing and evaluation community. Evaluating blackbox autonomy software via simulation-based testing requires an enormous amount of runs to provide assurances about the system's behaviors and decision-making processes [25].

One way of reducing the number of required runs is to intelligently select experiments via adaptive sampling methods that utilize a surrogate model of the System Under Test (SUT).

In our previous work [142], we detailed an adaptive sampling framework that

identifies critically ranked test scenarios for evaluation of autonomous vehicles. However, even when intelligently selecting test scenarios, the dimensionality of a realistic testing space (i.e. obstacles, missions, environments, etc.) can quickly require millions of simulations to be run. In addition, predicting the performance of untested scenarios through traditional classification techniques is limited due to the highly nonlinear performance landscape of autonomous systems.

Imitation learning [123] methods aim to mimic the behaviors and actions of the agent under observation. In the past, imitation learning has primarily been viewed as a replacement or supplement for reinforcement learning [128], [173], where the goal is to develop an optimal behavior by starting from a known expert policy. For this chapter, we are interested in the subdomain of behavioral cloning [123], where the imitator learns policies that replicate the target agent as closely as possible, even if those behaviors are sub-optimal. Thus, just as physics-based models are used to approximate physical systems, we aim to use behavioral cloning techniques to approximate the "brain" of the system, i.e. its decision-making processes and behavioral modes.

In this chapter we introduce a new method for generating test scenarios for autonomous vehicles by utilizing imitation learning surrogates to guide an adaptive search. Our approach iteratively trains an imitator agent that is then used in place of the real agent in simulation. Over time it generates increasingly accurate predictions of scenario performance as seen in Figure 6.1. These predictions are then used to estimate where performance boundaries may exist for the both the real and surrogate agents. The remainder of the chapter is organized as follows. In Section 6.2 we

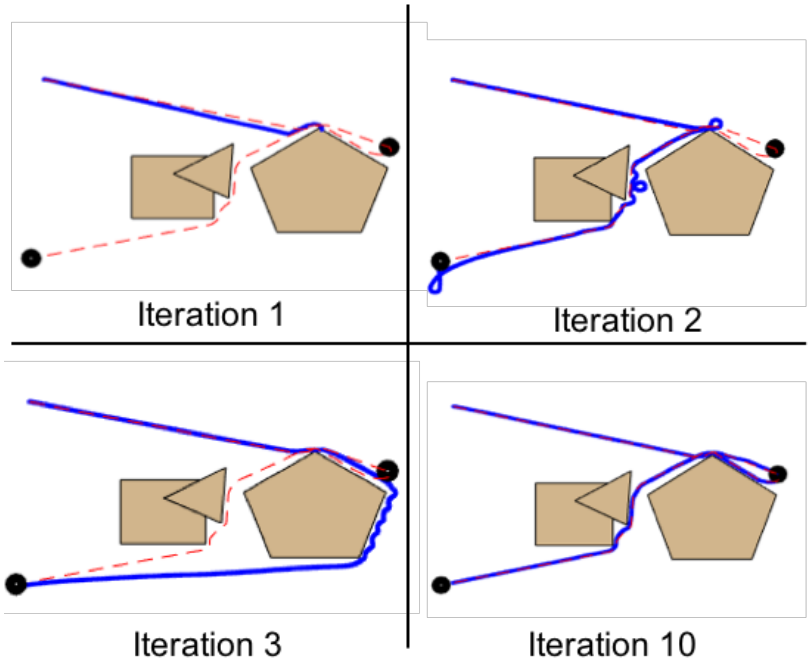


Figure 6.1: In this example scenario the imitator (blue line) rapidly converges to the expert agent's path (red line) despite never experiencing this scenario during training.

discuss the problem of scenario generation for autonomous systems. The problem formulation is given in Section 6.3. In section 6.4 we describe the imitation learning problem and the Q-DAgger algorithm. In Section 6.5 we describe our autonomous underwater vehicle (AUV) target expert and scenario. In Section 6.6 we provide the results of replicating the AUV's navigation behaviors. Finaly, in Section 6.7 we summarize our findings and discuss future work.

6.2 Test Generation Process

Test scenario generation techniques aim to identify inputs that will trigger specific behaviors in an autonomous agent. Often, this focuses on environments that will trigger a fault or violation of the system's requirements. In this section we

will quickly review the work in Chapter 3 in generating challenging and diverse test scenarios for autonomous vehicles and how this approach can be augmented with imitation learning.

6.2.1 Performance Boundaries

In attempting to characterize the performance landscape of an autonomous system, we previously introduced the concept of *performance boundaries* [142]. These are transitional regions in the testing space where small changes in scenario parameters cause large changes in system performance. In our prior work we outlined methods for identifying scenarios in the regions of these performance boundaries. As an example, Figure 6.2 illustrates a simple performance boundary on a simulated autonomous agent, where a small change in the position of the pentagonal obstacle means the difference between obstacle avoidance and a collision.

Identifying performance boundaries in a testing space is useful not only for better characterization of the performance landscape, but also for providing an estimate for the certainty of the vehicle's performance. In other words, the outcome of scenarios that lie near performance boundaries are less robust than those that lie away from the performance boundaries. Another purpose for boundary identification is that they can serve as delta-tests to infer which environmental factors trigger specific performance modes or decisions by the autonomy.

6.2.2 Range Adversarial Planning Tool

To address the problem of discovering challenging test scenarios with minimal simulations, we developed the Range Adversarial Planning Tool (RAPT) [142]. This simulation-based framework accepts a testing space, generates and runs scenarios within the testing space in simulation, and ultimately provides a ranked test suite to be executed on hardware. The test generation algorithms within RAPT can be broken into two phases: adaptive search and boundary identification.

To mitigate the high dimensionality of a realistic testing space, the adaptive search phase intelligently generates the scenarios that are run in simulation. During this search, scenarios are chosen so as to achieve a balance between exploring the full testing space while also preferentially selecting scenarios near the inferred

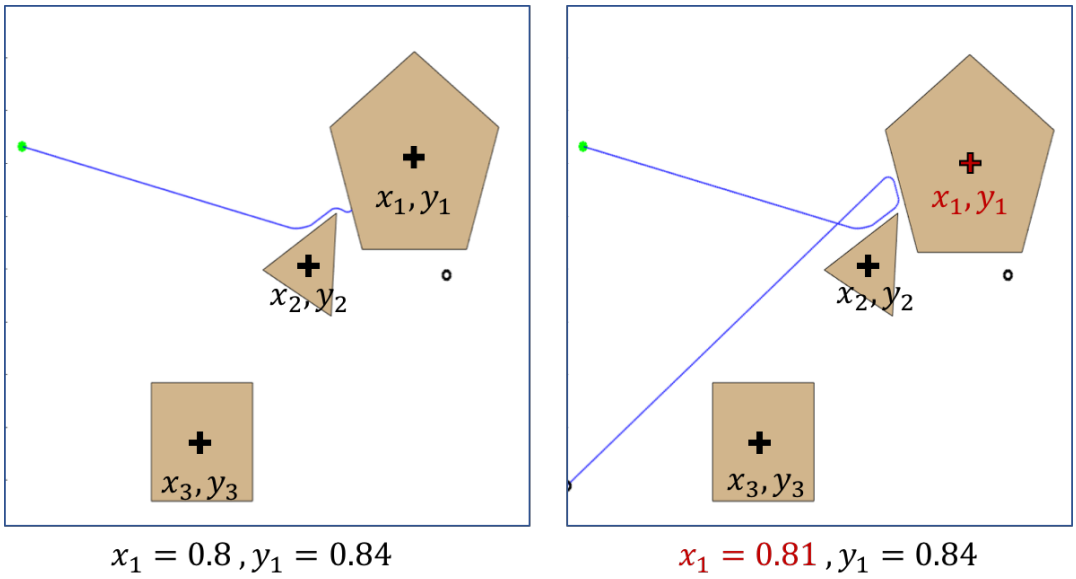


Figure 6.2: Example of a performance boundary - a small change in x_1 results in a new trajectory (blue line) that now avoids collision (i.e. a change in performance).

performance boundaries.

The second phase of test generation within RAPT is boundary identification. Here, we use adjacency clustering of the resulting scenario set from the adaptive search to identify pairs of scenarios which straddle performance boundaries. Scenario pairs that lie closest to the boundary are ranked and delivered to the test engineer for evaluation. The end result is a set of test scenario pairs which represent areas of vehicle performance uncertainty, where the source of performance change can be inferred from the scenario difference across the boundary pair.

6.2.3 Surrogate Agent Accelerated Generation

Given the expense of running simulations against the expert agent, we have designed a modified architecture that will allow us to identify the performance boundaries by using a surrogate imitator agent. Under this framework, instead of directly querying the expert system, the RAPT software generates results using the imitator agent running in a faster-than-real-time simulation. This results in a set of scenarios which describe the performance boundaries of the system. Subsequently, because these scenarios have high performance uncertainty, they are then run against the expert agent to validate the results. This process is summarized below.

1. Use **expert** to train imitator [50-100 runs]

2. Use **imitator** to discover performance boundaries [6000 runs]

3. Run **expert** against predicted performance boundaries [100 runs]

4. Return the set of performance boundaries confirmed by the expert

The proposed framework would reduce the number of runs required by the expert agent from several thousand to a few hundred. The remainder of this chapter will explore whether an imitator agent is accurate enough to enable this proposed framework. In particular, we want to answer the following questions.

- Can we train an imitator agent that exhibits similar performance modes and boundaries as the expert?
- How many annotated trajectories are required to train the imitator agent?

6.3 Problem Formulation

In this section we will define the terminology used throughout the rest of the chapter. First, we will provide our definitions of terms required to describe the imitation learning process. Then we will discuss how this relates to our target problem of test scenario generation.

6.3.1 States and Policies

In this chapter we will use the common conventions of defining the state of our world as $s \in S$ with the agent capable of taking actions $a \in A$. The state transition function is defined as $\delta(a, s) = s'$ where $s' \in S$ is another valid state.

A trajectory $\mathcal{T}^M = \{s_0, ..., s_{M-1}\}$ in this context is the sequence of states of length M that is created by running the action set $\mathcal{A}^M = [a_0, ..., a_{M-1}]$ through the state transition function.

Given a state s, we are capable of taking an observation $\psi(s) = f$ where f is some set of features in the world that the agent is capable of sensing or has *a priori* knowledge (e.g. maps or mission objectives).

A policy $\pi(f) = a$ is a function that maps observed features into actions.

Finally we define a simulator as a function $\Phi(\pi, s_0) = [\mathcal{T}^M, \mathcal{A}^M]$ which takes an initial state and policy and generates a trajectory by continuous execution of the policy and state transition function $s_{i+1} = \delta(s_i, \pi(\psi(s_i)))$ until it reaches a terminal state $s \in S_{terminal}$ or reaches an end time M_{max}.

6.3.2 Imitator Policy

Imitation learning is the process of training the policy π such that it closely replicates the performance of original expert policy π^*. During training we designate the imitator policy as π_i to indicate the current iteration of the training process.

The imitator policy is trained on the training set $\mathcal{D}^k = [\mathcal{F}^k, \mathcal{A}^k]$, which is the combined feature history and action history for k simulations, where the feature history $\mathcal{F} = \psi(\mathcal{T})$is defined as the state history passed through the observation function. We denote $\mathcal{L}(\mathcal{D})$ as the loss function of π with respect to π^* for all states in $\mathcal{D}$.

6.3.3 Scenarios and Mission Performance

Here we define a scenario input state as the vector $X = [x_1, x_2, ..x_n]$. Each element in the state space vector represents a variable in the environment, mission,

or vehicle parameters (obstacle positions, time windows, mission priorities, etc.). We assume the scenario configuration state X is related to the initial world state s_0 by some scenario generator function $\mathcal{K}(X) = s_0$.

The performance score is defined as the vector $Y = [y_1, y_2, ..y_m]$. Each element in the score vector represents a performance metric by which the autonomy is evaluated, such as percentage of fuel consumed or number of way points reached.

For each mission there is an evaluator function $Z(\mathcal{T}) = Y$ which takes the simulation history and returns the performance scores for that trajectory and scenario.

The performance boundaries of a system are defined as a set of paired scenarios $B_\pi = [(X_{1,a}, X_{1,b}), ..., (X_{n,a}, X_{n,b})]$ where $X_{i,a}$ and $X_{i,b}$ are each the others nearest neighbor to the other for which $Y_{i,a} \neq Y_{i,b}$.

6.3.4 Objectives and Metrics

Our unique objective in the field of imitation learning is to determine whether the simulated imitator agent results in the same performance mode as the expert. To measure this we will be using the following accuracy metrics of the imitator agent's behaviors.

- **Score Error**: $\frac{1}{N}\sum_{i=1}^{N} ||Z(\Phi(\pi, s_i))) - Z(\Phi(\pi*, s_i)))||$
- **Path Error**: $\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} ||s_{i,j,\pi} - s_{i,j,\pi*}||$

In addition, we are interested in determining if the performance boundaries of the imitator agent are accurate enough (compared to those of the expert) to

be used to guide test selection. To measure this we also introduce the following boundary-based metrics.

- **Boundary Accuracy**: $\frac{1}{N}\sum_{i=1}^{N}\delta_B(x_i)$ where $\delta_B(X_i) = 1$ when $X_i \in B_\pi \cap B_{\pi*}$ and 0 otherwise
- **Distance Error**: $\frac{1}{N}\sum_{i=1}^{N}||D_{B_\pi}(x_i) - D_{B_{\pi*}}(x_i)||$ where D_{B_π} is the distance from scenario x_i to its closest neighbor in B_π.

6.4 Imitation Learning

It is generally accepted that training a controller using only pre-recorded data will not lead to a stable policy. The primary limitation of this approach is that any error in the learned policy can lead to states not seen in the training set, ultimately leading to unstable behavior.

This problem can be solved by utilizing online-learning, where the imitator is allowed to control the vehicle while the expert records the correct control actions for all states in the trajectory. This technique, known as DAgger [135] and has seen recent success in several behavioral cloning applications.

Often, the optimal trajectories for an autonomous vehicle involve long periods of constant velocity and heading. Because state information and control actions are used to train the imitator, these periods dominate the training sets. Under these circumstances, a network can minimize loss by always returning the same velocity command regardless of the state. To avoid this we have modified the method for aggregating the training set such that it over-samples the major decision points of

the trajectories and under-samples the majority of null control actions. Instead of aggregating the entire data-set we only retain the worst predictions from $\mathcal{D}$. This is similar to the EP-Opt procedure used to train robust reinforcement learning policies [174]. We refer to this method as Q-DAgger which is defined in more detail in Algorithm 9.

Algorithm 9 Q-DAGGER

Create initial dataset $\mathcal{D}_0$ from a set of k scenarios running π^*
Train π_0 from $\mathcal{D}_0$
for $i \leq N$ **do**
  Generate dataset $\mathcal{D'}_i$ from a set of k scenarios running π_i
  Create annotated dataset $\mathcal{D}_i$ by running π^* against $\mathcal{D'}_i$
  Compute $Q_\epsilon = \epsilon$-percentile of $\mathcal{L}(\mathcal{D})$
  Select $\mathcal{D}_\epsilon = \ \tau_j : \mathcal{L}(\tau_j) \leq Q_\epsilon$
  Aggregate $\mathcal{D} = \mathcal{D}_i \cup \mathcal{D}_\epsilon$
  Update policy $\pi_i = PolicyUpdate(\pi_{i-1}, \mathcal{D})$
  i++
end for

By applying quantile down-sampling, we not only achieve our goal of over-sampling the decision points of the trajectory, but also significantly reduce the amount of training samples used during each update step. This allows Q-DAgger to update the policy much faster than if each full epoch was performed with all collected data. The network architecture is shown in Table 6.1. The network contains 400 input nodes, two hidden layers of size 150 and 50, and an output layer for predicted trajectory. We used a mean squared error loss function and ADAM [175] algorithm for optimization.

Layer Type	Size	Activation
Input	400	Relu
Fully Connected	150	Sigmoid
Fully Connected	50	Sigmoid
Output	2	N\A

Table 6.1: Neural Network Architecture

The imitator network was implemented in Tensorflow and executed on a machine with an NVIDIA 980 GTX GPU. This allowed us to generate control actions for a given state significantly more quickly than our prior autonomy solutions. We had two prior controllers which we used for the comparsion. The first was a model predictive controller (MPC) which used non-linear optimization to generate high-fidelity trajectories and was implemented in C++. The second was a UUV Autonomy which used a tangent bug controller [176] which was implemented in Matlab. A comparison of the computational speed for each of these methods is shown in Table 6.2.

Controller Type	Time (sec) per 100 Evals
Model Predictive Controller	5.87
UUV Autonomy	0.487
Imitator Network	0.00421

Table 6.2: Comparison of Controller Speeds

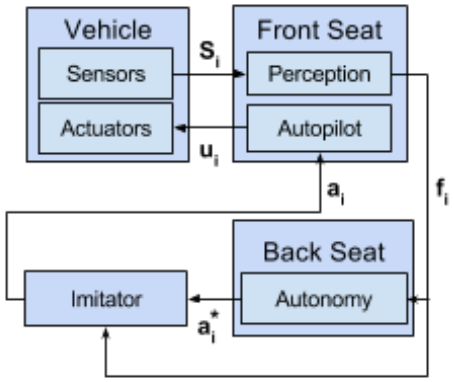


Figure 6.3: Flowchart outlining the autonomous vehicle testbed used for imitation learning.

6.5 Autonomous Vehicle Testbed

6.5.1 Testbed Framework

To enable the application of Q-DAgger to a real system, we propose the platform-agnostic autonomous vehicle testbed of Figure 6.3. This testbed is composed of a front-seat computer which runs a low-level autopilot and a back-seat computer which runs the decision-making software of the autonomy. Separating these systems allows us to directly intercept the communications between the front-seat and back-seat, ultimately giving the testbed access to perceptual information and control commands. We can then either let the expert's commands be passed directly to the front-seat or send imitator commands instead.

6.5.2 Mission Description

While the methods of this chapter are applicable to autonomous systems of all domains, we focus our experimental results here as a case study on the underwater domain. An OceanServer IVER [177] autonomous underwater vehicle (AUV) was selected as the system under test. This platform was chosen because it conforms to the framework of Figure 6.3 and is widely used in the underwater robotics community. While the training and validation of our imitation learning framework is performed here in simulation, the autonomy software being studied is the same version that we utilize on the hardware platform.

The mission scenario applied to this vehicle, Figure 6.4, contains a launch location, a mission waypoint, and a recovery point. While traversing the mission area, the agent must avoid obstacles and monitor its remaining battery. If at any time the system believes it cannot achieve both the mission waypoint and the recovery waypoint, it will abort the mission and return to the recovery. The testing space

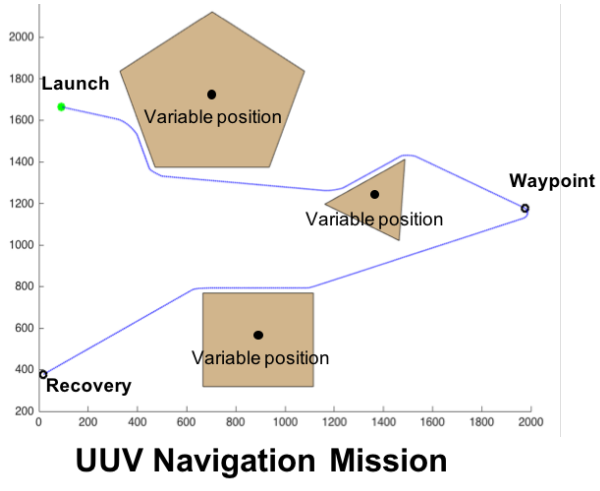


Figure 6.4: Example UUV mission scenario

within this mission consists of the (X,Y) position of each obstacle as well as the starting battery level, resulting in a seven-dimensional testing space.

The performance of the vehicle is evaluated by how much of the mission it completes. Successfully reaching only the mission waypoint is scored as a Mission Success (MS), while successfully reaching only the recovery point is scored as a Safety Success (SS). Completing both objectives is scored as a Total Success (TS), and completing neither is scored as a Total Failure (TF).

6.5.3 Feature Space and Action Space

Our simulated vehicle model uses a 3-degree of freedom hydrodynamics model typical of torpedo-shaped Unmanned Underwater Vehicle (UUV)s (surge, sway, and yaw). The simulated autopilot turns desired vector commands into low-level control actions (rudder angle, thrust) and is outfitted with a sensor package including an Inertial Navigation System (INS) and SONAR. We convert these sensor inputs and *a priori* information about the environment into the following features:

- f_{fuel} measures the remaining battery percentage.
- $f_{map} = [\bar{dx}_{way}, \bar{dy}_{way}, \bar{dx}_{rec}, \bar{dy}_{rec}]$ contains the relative vectors from the vehicle to the mission waypoint and recovery waypoint.
- $f_{sonar} = [r_0, ...r_{360}]$ contains SONAR readings subject to 100 meters of range and 360 degrees of coverage with 1 degree resolution.

We transform all of these features into the vehicle's local frame (x aligned in surge),

then normalize all outputs to a range of [-1,1]. Finally, we concatenate all of these to get our final feature vector. $f_j = [f_{fuel,j}, f_{map,j}, f_{sonar,j}]$

The actions of the system are the desired speed and heading commands represented as a vector relative to the vehicle's local frame, $a = [dx_{des}, dy_{des}]$. These are similarly normalized such that $0 <= ||a|| <= 1$, where an action of magnitude one represents maximum velocity.

6.6 Results

6.6.1 Experimental Setup

In this section we analyze the results of the trained imitator agent on the AUV simulation of Section 6.5. We populate our initial training set $\mathcal{D}_0$ with 5 trajectories generated by the expert policy π^*. Following this initial training set, subsequent training sets $\mathcal{D}_i$ are generated using 5 additional trajectories from the imitator policy π_i. We trained the imitator policy for 50 iterations with each iteration consisting of 5 randomly generated scenarios, leading to 250 unique scenarios in total. All simulations were run and the neural networks were trained on a workstation with a Intel Xeon E5-2600 processor, 32 GB of RAM, and a Nvidia 980 GTX GPU.

In our first set of experiments we measure the performance of the imitator policy on two metrics: path error and score accuracy. As defined mathematically in Section 6.3, path error is the mean error between all (X,Y) locations of the expert trajectory and the imitator trajectory. When these trajectories have different lengths, the shorter trajectory is extended by repeating the last position. Score

accuracy measures the ability of the imitator to correctly achieve the same performance mode (MS, SS, TS, TF) of the real agent. The results of these criteria were compared for Q-DAgger, DAgger, and a standard multilayer perceptron controller trained using supervised learning as a naïve baseline.

In our second set of experiments we explored how well the imitator policy performed in the boundary regions, in terms of both its accuracy in predicting the correct performance modes and how the boundary locations changed between the expert and imitator. For these experiments we only present the results for the Q-DAgger(50) agent.

6.6.2 Prediction Accuracy

While all of the imitator policies achieved at least 97.5% accuracy at predicting individual control actions by the first iteration, this was due to the fact that a majority of the control actions simply involved tracking a straight path. The critical actions such as obstacle avoidance maneuvers and transition from the mission waypoint to the recovery points make up only a small fraction of the overall trajectory. This means the regression loss during training is not an accurate predictor of how the imitator will perform at replicating the target behaviors. This is reflected by the performance of the baseline supervised learning algorithm in Figure 6.5.

By comparison, the DAgger family of algorithms performed substantially better. As can be seen in Figure 6.1, these algorithms required minimal training iterations in order to achieve accurate predictions of both trajectories and performance

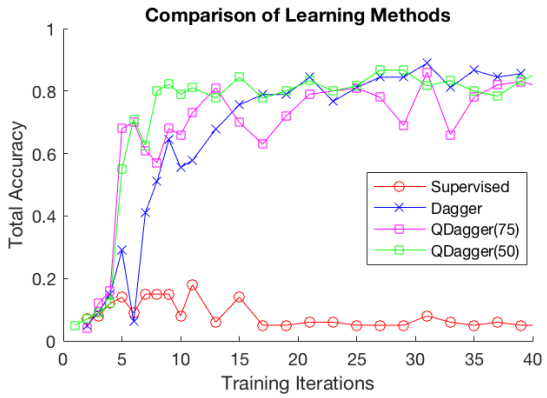


Figure 6.5: Comparison of standard supervised learning with DAgger and Q-DAgger at 50th percentile and 75th percentile downsampling.

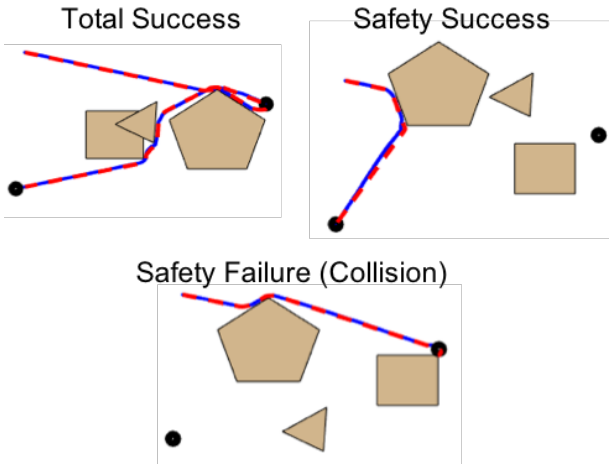


Figure 6.6: The imitator (blue line) is able to reproduce the paths of expert (red line) for multiple different performance modes

modes. All of these techniques generally converged to an average of 85% overall prediction accuracy for replicating the correct performance mode. The models tended to converge after 15 iterations, meaning only 50-75 annotated trajectories were required to achieve reasonable performance. The detailed results from our experiments are documented in Table 6.3.

The reader should note that the Q-DAgger methods introduced in this chapter converge slightly more quickly than standard DAgger, but in the end the result is comparable. The true benefit of the Q-DAgger method is the increase in training speed over standard DAgger. The total training time to complete 40 iterations was *7 hours and 41 minutes* for DAgger whereas the Q-DAgger(50) training time was *2 hours and 32* minutes. This is due to the fact that the Q-DAgger method managed to discard up to 75% of all training data while retaining the same predictor performance. This strongly indicates that a vast majority of trajectory information is not important to the training process.

	Supervised	Dagger	Q-Dagger	
Metrics			(50)	(75)
Run Time	6h 12m	7h 41m	2h 32m	2h 12m
Score Accuracy	6%	85%	84%	81%
Path Error (m)	525m	70m	61 m	87m

Table 6.3: Comparision between imitation learning methods

6.6.3 Predicting Multiple Behaviors

One of the primary objectives of this work was determining if our system could encode multiple behaviors in an imitator policy. Three of the primary performance modes resulting from these behaviors are depicted in Figure 6.6. We are particularly concerned with how well the algorithms learn failure modes such as the collision mode depicted. To determine how well the imitator was learning each of these behaviors, the resulting performance modes for the expert system were compared to the predicted performance mode by the imitator in the form of a confusion matrix (Figure 6.7). In other words, this matrix shows the success of the imitator system in predicting the same performance mode that the expert system experiences.

Figure 6.7 reveals that the imitators had the most difficulty predicting SS cases, i.e. predicting when the autonomy should abort the mission and return early. They also tended to trend towards safety failure, indicating the imitator agents experienced collisions slightly more often than the true agent. While this is not unexpected from the overall imitator accuracy, it is useful to know what behaviors the imitators may have trouble replicating. Overall this data confirms that they are able to learn all the major performance modes of the expert.

6.6.4 Performance Boundary Accuracy

In addition to measuring our ability to predict the correct performance mode, we are also interested in how well the performance boundaries predicted by the imitator agent align with those of the true agent.

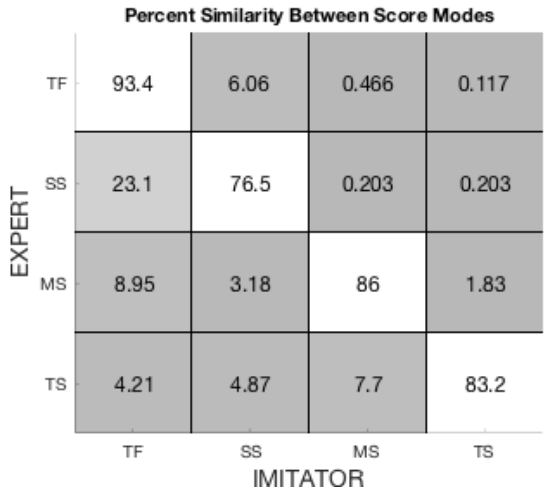


Figure 6.7: Confusion Matrix for the Q-DAgger(50) showing performance accuracy of the imitator for each of the performance modes.

For this comparison, we created a validation set of 5000 scenarios using a Sobol space-filling design. We then extracted the performance boundaries of both the expert agent and each of the imitator agents using the techniques from our previous work [142]. For this study, we did not include the baseline agent trained using only supervised learning as it did not have sufficient performance for the experiment. We computed the error in boundary distance prediction and looked at the correlation between the performance boundary regions and the locations where incorrect behaviors occurred.

We discovered that 86.8% of the imitator performance boundary regions were the same as the boundaries of the true system. The mean error in boundary distance prediction was 0.0096. The shift in performance boundaries between the true system and the imitator agent for a 2D slice of the testing space is shown in Figure 6.8. From these results we can have high confidence that any scenario that is near the performance boundary of the imitator agent is also in the vicinity of the performance

boundary for the true system.

Additionally, we can see that that nearly all of the scenarios (95%) where inaccurate behaviors occur are in the vicinity of the boundary regions, shown in Figure 6.8 by plotting the cases where the imitator did not have the same performance as the expert.

These results lead to two useful conclusions. The first is that the locations of the performance boundaries of the imitator agent are sufficiently accurate to support the approach discussed in Section 6.2.3. Using only 250 training scenarios where the expert system was in the loop we were able to locate the performance boundaries of a system that previously required 6000 simulations by the expert. The second conclusion is that the high correlation between prediction error and the location of the imitator's performance boundaries means that proximity to the boundaries can be used as a confidence measure regarding the imitator's behavior. Ultimately, this informs the user where additional runs by the expert system may be required.

6.7 Summary

In this chapter, we explored the application of imitation learning to the creation of surrogate agents for validation of autonomous vehicles in simulation. The first intellectual contribution was a demonstration that the Q-DAgger imitation learning algorithm can successfully train a surrogate agent to accurate replicate multi-objective UUV behaviors. The second is a demonstration that an imitation network is hundreds of times faster than the original controller, significantly increas-

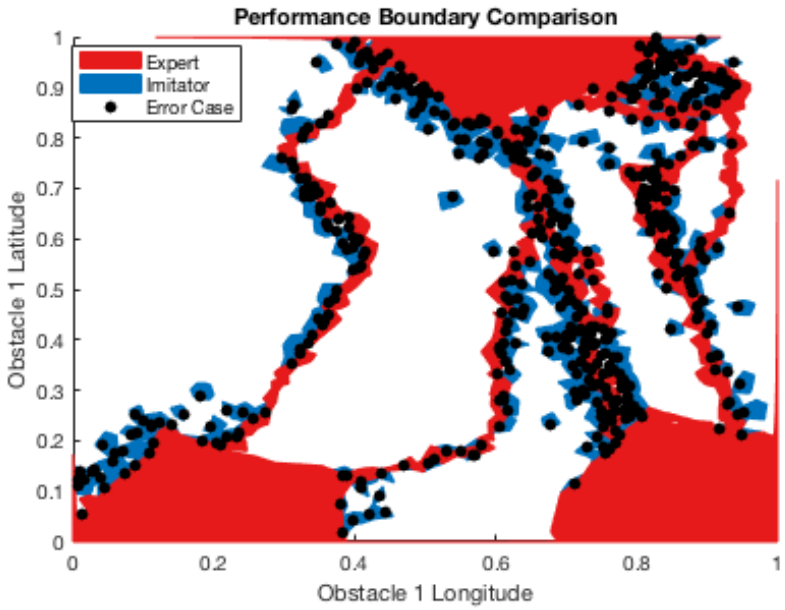


Figure 6.8: Comparison of the performance boundary regions of the imitator (blue) versus the expert (red) for a 2D slice of the scenario space. White regions represent areas of stable performance for both agents. The black dots are scenarios where the imitator performance mode does not match that of the expert.

ing the rate at which runs can be performed.

Behavioral cloning using DAgger can successfully create surrogate agents which accurately encode both the decision-making of the real agent and its resulting performance landscape. A single *multilayer perception* architecture is capable of not only learning a regression of the control actions, but can also encode multiple mission-level behaviors. By utilizing a shallow network we maximize the rate at which predictions can be made. Additionally, by applying percentile down-sampling as part of our Q-DAgger algorithm we can reduce the time it takes to train the imitation agent.

Chapter 7: Conclusions

This chapter presents the expected intellectual contributions of this dissertation and the anticipated benefits to the larger community. Further research directions to continue addressing the presented issues and questions are also discussed.

7.1 Intellectual Contributions

The main intellectual contributions include the following:

7.1.1 Automatic generation of challenging and diverse test scenarios for autonomous vehicles

This dissertation introduces a novel approach of using adaptive sampling to discover and identify the performance boundaries of an autonomous vehicle. This study posits that scenarios that lie upon these boundaries are the most informative test cases for characterizing the behaviors of an autonomous system. As applied in Chapter 3 and 4 this testing approach can be used to discover the performance boundaries of an autonomous robot completing a complex mission with multiple competing objectives. More generally the contributions from this work are as follows,

- An adaptive sampling algorithm, NNDV, which outperforms all standard baselines for the task of finding performance boundaries. This newly introduced technique generates tests which are both close to the performance boundaries and provide good coverage of the boundary regions. The NNDV algorithm does not require a system specific objective function like prior optimization based algorithms, enabling it to be applied to a wider range of system. It also is robust to discontinuities and noise in the output of the simulation.

- An unsupervised clustering approach that uses pair-wise metrics to identify transitions in scattered data-sets with an arbitrary number of dimensions. This approach describes the transition regions of a system by identifying pairs of cases that belong to different performance modes. As it is a density based technique it requires almost no-tuning or a priori knowledge of the distribution of the data or the size of the state-space. The output is a set of transition pairs categorized by the type of performance boundary they describe and sorted by predicted distance from the boundary.

- A scaled neighborhood density and variance estimation algorithm, S-NDV, which scales the neighborhood calculations of the NNDV algorithm based upon variable importance. This algorithm significantly reduces the impact of non-contributing inputs and allows the search to remain focused as the number of input dimensions increase. Outperforming non-scaling methods when applied to systems with a large number of dimensions. Where as prior research in search-based testing only focused on 2 to 6 dimensions I demonstrated the

new algorithm can search over 18 dimensions.

- Hierarchical clustering algorithms for analyzing high-dimensional outputs. I demonstrated that this technique can extend the maximum of number of output dimensions for the system from 2 to 66. This approach was predicated on the development of a hierarchical scoring method and is well suited to many autonomous vehicle missions. By combining hierarchical scoring with the sub-clustering algorithms it is possible to cluster and analyze dozens of different output metrics simultaneously. By using these methods an engineer can quickly select the scoring criterion that are most likely to reveal performance boundaries caused by changes in the decision making process of the system.

- The first study where test scenarios for an autonomous vehicle which were generated via search-based testing and then executed in the field. All prior research into search-based test generation methods have been restricted to simulation environments. I was able to validate performance boundaries on the hardware platform despite large amounts of execution error and uncertainty. Demonstrating the test scenario generation process I developed is capable of creating actionable field tests.

These contributions are combined to create RAPT, A modular software toolbox for adaptive sampling and performance boundary identification of black-box systems. The modular nature of this toolbox means that a designer can experiment with a variety of search methods, including commercial products such as SUMO

[79]. In addition, the toolbox is designed for easy integration with a variety of simulation environments and software components. Beyond the simulation environments discussed in this dissertation plug-ins currently exist for a variety of standard optimization test functions [178], the MATLAB Robots System Toolbox [179], and a UGV simulation for reinforcement learning via PROPS [180]. It is capable of being complied as as a standalone tool and is currently being installed at the Keyport Underwater Testing Facility (Technology Readiness Level of 6).

7.1.2 Adaptive sampling framework for generating self-righting paths for generic robot geometries

The algorithms introduced in this dissertation have applications to any system where the transition regions hold special significance. In Chapter 5 I extend the work performed in Chapter 3 to sample along constrained manifolds in a high-dimensional configuration space. Specifically I demonstrate how the adaptive sampling framework can be integrated with a simulation which verifies stable and unstable UGV configurations as a method for generating self-righting plans. This effort resulted in the following contributions,

- A manifold-constrained version of the NNDV adaptive sampling algorithm. This algorithm allows us to search the C-space of a ground robot for locations where it transitions from stable to unstable. This search method offers an order of magnitude reduction in the number of simulations required to generate self-righting graphs for ground robots.

- A continuity-based clustering method for identifying contiguous performance regions. This algorithm utilizes Gabriel graph connectivity to establish when samples should be connected as members of the same cluster. Allowing us to create self-righting graphs from scattered data, which was not supported by prior frameworks. It also is faster than edge checking techniques used in traditional PRM generation.

- An improved simulation-based framework for validating the self-righting capabilities of ground robots. This removes the need to simplify the robot models to two-dimensional models with dozens of facets and instead utilize full resolution three-dimensional models with thousands of facets. When combined with the adaptive sampling algorithm it enables the analysis of robots with greater degrees of freedom. Allowing users to perform validation on realistic systems without simplifying assumptions.

These techniques were successfully applied to generate self-righting graphs for robots with arbitrary morphologies. Previously, only a grid-based sampling method had been successfully applied to compute these topological graphs. By applying new adaptive sampling algorithms I can both reduce the amount of time it takes to generate graphs with similar resolution by an order of magnitude. Making it possible to apply this method to robots with significantly higher degrees of freedom. Finally, all of these new improvements enable the analysis of robot models in three-dimensions. Improving upon the previous approaches which were only capable of analysis in two-dimensions.

7.1.3 Deep imitation learning for creating surrogate agents

Surrogate behavioral meta-models make it possible to perform fast predictions of an agent's performance in unknown scenarios. In Chapter 6 I developed a deep imitation learning framework and demonstrated how it can be applied to autonomous unmanned systems. It was successfully applied to a single agent UUV navigation mission and was able to model multiple behaviors using a single MLP network. These models run significantly faster than the original autonomy controllers, allowing for faster simulations. In addition, these models were leveraged as a method for predicting scenario similarity that can be applied to the scenario generation process discussed in Chapter 3.

In short, this study lead to the following contributions.

- A Q-Dagger imitation learning algorithm which can successfully train a surrogate agent which accurately replicates a UUV autonomy with multi-objective behaviors. Previous behavioral cloning efforts had only demonstrated success at learning simple obstacle avoidance and waypoint strategies. The imitator network learned additional policy decisions such as when to turn back when it was low on fuel and when the true agent would collide with an obstacle. This allows for greater predictive ability than had been demonstrated to this point.

- A several orders of magnitude increase in the speed of the surrogate agent over the autonomy under test. The shallow MLP network is sufficiently accurate to replicate the behavioral modes of the autonomy while being up to 100

times faster than even the optimized implementations. This will allow for greater number of simulations to be performed and faster identification of the performance boundaries.

- A new quantile data-aggregation (Q-DAgger) algorithm which reduces the computational time required to perform imitation learning. This algorithm works by reducing the training set to only those samples which maximize the loss of the imitator network. This helps bias the training set towards the important decision points of the trajectory. This algorithm trains in fewer iterations than the original DAgger algorithm while also taking significantly less computational time to train.

These advances offer a new approach for predicting the performance boundaries of autonomous vehicle software using an imitation agent. The performance boundaries of the imitator agent prove to be co-located with the performance boundaries of the actual system. Thus an estimate of boundary distance generated by the imitator system can be taken with relative confidence. In addition most of the inaccurate behaviors between the imitator and true system occur at the boundary region. Allowing boundary distance to be used as a stand-in for imitator prediction confidence.

7.2 Anticipated Benefits

This dissertation covers the problems of efficiently generating test-cases for autonomous vehicles using adaptive sampling and surrogate models. It introduces a

novel search-based testing approach which eschews typical hand-designed objective functions in favor of finding performance boundaries across all behavioral modes. This framework has been developed as a MATLAB toolbox which is deployed at government test-ranges. In addition, it provides a method for generating transition graphs for hybrid systems. Which has many applications including the ability to verify a ground robot's ability to self-right. These techniques have been successfully demonstrated on systems with random effects, indicating they can be used to cover either uncertainty in system performance or stochastic decision making processes. Finally, this work explores the problem of generating surrogate agents which can be used in place of the real system for faster prediction of performance.

7.3 Future Work

7.3.1 Meta-modeling and Manifold Learning

The NNDV algorithm introduced in Chapter 3 was selected because it provided similar performance to Gaussian Process Regression but with substantially better scaling. The choice to prioritize training time over accuracy was driven by the faster-than-real time performance of the UUV simulation. As time spent training a meta-model was potentially time that could have been spent running more simulations. This coupled with the difficulty of fitting meta-models to the high-dimensional non-linear state-space meant that efforts for developing better meta-modeling strategies were dropped. Yet there are many cases where having accurate meta-models would be beneficial for providing assurances of reliability and accelerating the search of

the system.

In particular, many autonomous vehicle simulators are not capable of running faster than real-time. Limiting the amount of runs which can be performed even with high powered computing resources. When this is the case having strong meta-modeling techniques which can guide surrogate optimization approaches will be critical. Manifold learning techniques which can describe the performance boundaries of the system could simultaneously drive the search and provide stronger assurances for the volume of space where stable performance can be expected. The one benefit a slower simulation provides is that there is more time to train a meta-model while awaiting results. Opening up the possibility of using methods which are more expensive to train, such as artificial neural networks. It is my strong recommendation that any follow-up to the work in this dissertation explores more accurate and advanced meta-modeling approaches.

7.3.2 Testing Adaptive and Learning Systems

The verification and validation of learning systems is still an open problem. Verifying the performance of autonomous vehicles capable of learning will require advances in formal methods. In particular, the development of proofs capable of showing that the entire policy space that the autonomous system can occupy will never violate its operational requirements. Additionally, many of the autonomous vehicles of the future will likely be highly complex systems consisting of neural networks, model predictive controllers, and rule-based behaviors. Potentially making

white-box and grey-box testing entirely infeasible. Current research efforts are currently concerned with proving a single safe controller as safe then developing it as a watchdog mechanism the regulates the behavior of the learning system, rather than attempt to verify the entire policy space directly.

Therefore, evaluating these systems will require black-box testing and falsification approaches. All of the adaptive search and optimization approaches for developing testing scenarios discussed in this dissertation make the assumption of a static policy. Applying these strategies to a learning system will require a better understanding of how a learning system will adapt its policy as it is presented with test scenarios. In the development of this dissertation I performed experiments where the RAPT software was applied to a vehicle capable of reinforcement learning. During these experiments I discovered that depending on the method of generating the tests was highly influential on the policy that was learned by the vehicle. However, the effects were highly dependent on the initial scenario distribution, how samples were selected from that distribution, and the final target testing distribution. To the point where different testing strategies could create either a more robust system or one that fails to converge. More research is required on establishing these relationships between the training data, the testing regime, and the resulting learned policy. This will necessitate development of new problem definitions for what it means to test a learning autonomy and new frameworks for simulation-based testing.

7.3.3 Imitation Learning and World Modeling

The imitation learning approach discussed in Chapter 6 was incredibly successful at encoding the behaviors of a target UUV controller. This work acts as an effective proof of concept for using surrogate agents for simulation-based testing, but a great deal of additional research is required before a real test-bed can be developed. The first problem that needs to be addressed is extracting relevant features from raw sensor data streams. In Chapter 6 the features were hand-designed and directly fed to the surrogate agent from the simulation. This approach works when the designer is familiar with the autonomy, mission, and simulation but require a perception component be designed specifically for that system. In order to apply the proposed framework to a wide variety of black-box systems it will be necessary to automatically identify the sensor data which is being utilized by the autonomy and turn it into features to feed to the imitator network.

Another intriguing area of research is creating surrogate simulators using neural networks. Surrogate agents alone enable faster than real-time operation as they can be directly integrated with the simulator and run on the same computer. This allows the engineer to bypass issues software-in-the-loop testbeds have with the synchronization of multiple processes across several computers. However, the surrogate agent is still limited by the speed of the simulator itself. By creating a surrogate simulation using deep network world modeling [108] it may be possible to run approximate simulations orders of magnitude faster than the original simulation. In addition, surrogate simulations may make it easier to isolate and run sub-simulations

on specific trajectory segments that are identified as interesting. There are many questions that will need to be addressed about whether the latent space of the auto-encoder is expressive enough to capture the true state or if recurrent networks are capable of describing complex dynamical systems.

Bibliography

[1] B. A. Weiss and L. C. Schmidt, "Multi-relationship evaluation design: Formalization of an automatic test plan generator," *Expert Systems with Applications*, vol. 40, no. 9, pp. 3764–3774, Jul. 2013.

[2] B. Weiss, "Multi-relationship evaluation design (MRED): An interactive test plan designer for advanced and emerging technologies," PhD thesis, University of Maryland, College Park, 2012.

[3] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Vehicle Automation 3*, G. Meyer and S. Beiker, Eds., Cham: Springer International Publishing, 2016, pp. 91–109.

[4] M. Wagner and P. Koopman, "A philosophy for developing trust in self-driving cars," in *Road Vehicle Automation 2*, G. Meyer and S. Beiker, Eds., Cham: Springer International Publishing, 2015, pp. 163–171.

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

[6] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167–199, Sep. 2005.

[7] P. McMinn, "Search-based software test data generation: A survey," *Software Testing Verification and Reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[8] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg, "Does automated unit test generation really help software testers? A controlled empirical study," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 4, pp. 1–49, Sep. 2, 2015.

[9] P. Gandhi, "A survey on prospects of automated software test case generation methods," presented at the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016.

[10] A. Harris and J. M. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *Southeastcon, 2011 Proceedings of IEEE*, IEEE, 2011, pp. 243–249.

[11] C. Pepper, S. Balakirsky, and C. Scrapper, "Robot simulation physics validation," in *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, ACM, 2007, pp. 97–104.

[12] M. Torres-Torriti, T. Arredondo, and P. Castillo-Pizarro, "Survey and comparative study of free simulation software for mobile robots," *Robotica*, vol. 34, no. 4, pp. 791–822, Apr. 2016.

[13] X. Zou, R. Alexander, and J. McDermid, "Safety validation of sense and avoid algorithms using simulation and evolutionary search," in *Computer Safety, Reliability, and Security*, Springer, 2014, pp. 33–48.

[14] ——, "Testing method for multi-UAV conflict resolution using agent-based simulation and multi-objective search," *Journal of Aerospace Information Systems*, vol. 13, no. 5, pp. 191–203, May 2016.

[15] X. Zou, "Supporting validation of UAV sense-and- avoid algorithms with agent-based simulation and evolutionary search," PhD thesis, University of York, Aug. 2016.

[16] H. Beglerovic, M. Stolz, and M. Horn, "Testing of autonomous vehicles using surrogate models and stochastic optimization," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2017, pp. 1–6.

[17] C. E. Tuncali, S. Yaghoubi, T. P. Pavlic, and G. Fainekos, "Functional gradient descent optimization for automatic test case generation for vehicle controllers," in *Automation Science and Engineering (CASE), 2017 IEEE International Conference on. IEEE*, IEEE, Aug. 2017, pp. 1059–1064.

[18] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," *ARXIV:1804.06760 [cs]*, Apr. 18, 2018. arXiv: 1804.06760.

[19] A. Tetlay and P. John, "Determining the lines of system maturity, system readiness and capability readiness in the system development lifecycle.," presented at the 7th Annual Conference on Systems Engineering Research 2009, 2009.

[20] R. Kuhn, "Introducing combinatorial testing in large organizations," in *ASTQB Software Testing Conference*, 2014.

[21] M. Steinberg, J. Stack, and T. Paluszkiewicz, "Long duration autonomy for maritime systems: Challenges and opportunities," *Autonomous Robots*, vol. 40, no. 7, pp. 1119–1122, Oct. 2016.

[22] A. Schwartz and H. Do, "Cost-effective regression testing through adaptive test prioritization strategies," *Journal of Systems and Software*, vol. 115, pp. 61–81, May 2016.

[23] T. Sotiropoulos, J. Guiochet, F. Ingrand, and H. Waeselynck, "Virtual worlds for testing robot navigation: A study on the difficulty level," in *12th European Dependable Computing Conference (EDCC 2016)*, IEEE, 2016, pp. 153–160.

[24] S. Dogramadzi, M. E. Giannaccini, C. Harper, M. Sobhani, R. Woodman, and J. Choung, "Environmental hazard analysis - a variant of preliminary hazard analysis for autonomous mobile robots," *Journal of Intelligent & Robotic Systems*, vol. 76, no. 1, pp. 73–117, Sep. 2014.

[25] R. Alexander, H. R. Hawkins, and A. J. Rae, "Situation coverage a coverage criterion for testing autonomous robots," 2015.

[26] P. J. Durst, W. Gray, A. Nikitenko, J. Caetano, M. Trentini, and R. King, "A framework for predicting the mission-specific performance of autonomous unmanned systems," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, IEEE, 2014, pp. 1962–1969.

[27] V. V. Fedorov, *Theory of optimal experiments*. Elsevier, Jan. 1, 1972, 307 pp.

[28] L. Eriksson, E. Johansson, N. Kettaneh-Wold, C. Wikstrom, and S. Wold, "Design of experiments: Principles and applications," *Principles and Applications, Learn ways AB, Stockholm*, L. Eriksson and U. AB, Eds., 2000, OCLC: 249982676.

[29] A. Dean, D. Voss, and D. Draguljic, *Design and analysis of experiments*, Second edition, ser. Springer texts in statistics. New York: Springer, 2017, 840 pp., OCLC: 992474888.

[30] J. A. Jacquez, "Design of experiments," *Journal of the Franklin Institute*, vol. 335, no. 2, pp. 259–279, 1998.

[31] D. M. Titterington, "Optimal design: Some geometrical aspects of d-optimality," *Biometrika*, vol. 62, no. 2, p. 313, Aug. 1975.

[32] F. Mentre, A. Mallet, and D. Baccar, "Optimal design in random-effects regression models," *Biometrika*, vol. 84, no. 2, pp. 429–442, 1997.

[33] A. Saltelli, K. Chan, and E. M. Scott, *Sensitivity analysis*. Wiley New York, 2000, vol. 1.

[34] W. Chen, R. Jin, and A. Sudjianto, "Analytical variance-based global sensitivity analysis in simulation-based design under uncertainty," *Journal of Mechanical Design*, vol. 127, no. 5, p. 875, 2005.

[35] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, IEEE, 2002, pp. 91–95.

[36] L. S. Ghandehari, J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn, "BEN: A combinatorial testing-based fault localization tool," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, IEEE, 2015, pp. 1–4.

[37] N. Mahadevan, M. Lowry, J. Schumann, and G. Karsai, "DVER: A tool chain for cross-validation and perfection of discrete model-based diagnostic systems," in *Aerospace Conference, 2016 IEEE*, IEEE, 2016, pp. 1–15.

[38] T. Mahmoud and B. S. Ahmed, "An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8753–8765, Dec. 2015.

[39] J. Choi, "Model checking for decision making behaviour of heterogeneous multi-agent autonomous system," PhD thesis, Cranfield University, 2012.

[40] M. O'Brien, R. C. Arkin, D. Harrington, D. Lyons, and S. Jiang, "Automatic verification of autonomous robot missions," in *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, 2014, pp. 462–473.

[41] K. Meinke and P. Nycander, "Learning-based testing of distributed microservice architectures: Correctness and fault injection," in *Software Engineering and Formal Methods*, Springer, 2015, pp. 3–10.

[42] M. Leucker, "Learning meets verification," in *Formal Methods for Components and Objects*, Springer, 2007, pp. 127–151.

[43] T. Dang and T. Nahhal, "Coverage-guided test generation for continuous and hybrid systems," *Formal Methods in System Design*, vol. 34, no. 2, pp. 183–213, Apr. 2009.

[44] D. Araiza-Illan, A. G. Pipe, and K. Eder, "Model-based test generation for robotic software: Automata versus belief-desire-intention agents," *ARXIV preprint arXiv:1609.08439*, 2016.

[45] J. B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "Formal verification of ACAS x, an industrial airborne collision avoidance system," in *Proceedings of the 12th International Conference on Embedded Software*, IEEE Press, 2015, pp. 127–136.

[46] C. Fan, B. Qi, and S. Mitra, "Road to safe autonomy with data and formal reasoning," *ARXIV preprint arXiv:1704.06406*, 2017.

[47] N. Walkinshaw, K. Bogdanov, J. Derrick, and J. Paris, "Increasing functional coverage by inductive testing: A case study," in *Testing Software and Systems*, Springer, 2010, pp. 126–141.

[48] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, and M. Roper, "Reformulating software engineering as a search problem," *IEE Proceedings-software*, vol. 150, no. 3, pp. 161–175, 2003.

[49] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, Sep. 2011.

[50] L. T. M. Hanh, N. T. Binh, and K. T. Tung, "A novel fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis," *Journal of Systems and Software*, vol. 120, pp. 17–30, Oct. 2016.

[51] M. Patrick, R. Alexander, M. Oriol, and J. A. Clark, "Subdomain-based test data generation," *Journal of Systems and Software*, vol. 103, pp. 328–342, May 2015.

[52] J. Kim, J. M. Esposito, and V. Kumar, "Sampling-based algorithm for testing and validating robot controllers," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1257–1272, Dec. 1, 2006.

[53] J. Kempka, P. McMinn, and D. Sudholt, "Design and analysis of different alternating variable searches for search-based software testing," *Theoretical Computer Science*, vol. 605, pp. 1–20, Nov. 2015.

[54] Y. Qin, C. Xu, P. Yu, and J. Lu, "SIT: Sampling-based interactive testing for self-adaptive apps," *Journal of Systems and Software*, vol. 120, pp. 70–88, Oct. 2016.

[55] L. S. de Souza, R. B. Prudêncio, F. d. A. Barros, and E. H. d. S. Aranha, "Search based constrained test case selection using execution effort," *Expert Systems with Applications*, vol. 40, no. 12, pp. 4887–4896, Sep. 2013.

[56] A. Piziali, *Functional verification coverage measurement and analysis*. Springer Science & Business Media, May 8, 2007, 222 pp.

[57] B. Wile, J. C. Goss, and W. Roesner, *Comprehensive functional verification: The complete industry cycle*. Morgan Kaufmann, 2005, 703 pp., Google-Books-ID: XB91TWOtPAkC.

[58] D. Araiza-Illan, D. Western, A. Pipe, and K. Eder, "Coverage-driven verification," in *Hardware and Software: Verification and Testing*, ser. Lecture Notes in Computer Science, Springer, Cham, Nov. 17, 2015, pp. 69–84.

[59] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, pp. 1–42, Feb. 1, 2013.

[60] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, Jan. 2010.

[61] A. Panichella, R. Oliveto, M. D. Penta, and A. D. Lucia, "Improving multi-objective test case selection by injecting diversity in genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 358–383, Apr. 2015.

[62] P. G. Sapna and H. Mohanty, "Clustering test cases to achieve effective test selection," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, ACM, 2010, p. 15.

[63] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, ACM, 2009, pp. 201–212.

[64] M. Zalmanovici, O. Raz, and R. Tzoref-Brill, "Cluster-based test suite functional analysis," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, New York, NY, USA: ACM, 2016, pp. 962–967.

[65] Y. Miao, Z. Chen, S. Li, Z. Zhao, and Y. Zhou, "Identifying coincidental correctness for fault localization by clustering test cases.," in *SEKE*, 2012, pp. 267–272.

[66] D. Nurmuradov, R. Bryce, S. Piparia, and B. Bryant, "Clustering and combinatorial methods for test suite prioritization of GUI and web applications," in *Information Technology - New Generations*, ser. Advances in Intelligent Systems and Computing, Springer, Cham, 2018, pp. 459–466.

[67] M. Dimjašević and D. Giannakopoulou, "Test-case generation for runtime analysis and vice versa: Verification of aircraft separation assurance," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ACM Press, 2015, pp. 282–292.

[68] S. Alam, H. A. Abbass, C. J. Lokan, M. Ellejmi, and S. Kirby, "Computational red teaming to investigate failure patterns in medium term conflict detection," in *8th Eurocontrol Innovation Research Workshop, Eurocontrol Experimental Center, Brtigny-sur-Orge, France*, 2009.

[69] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *Computer Safety, Reliability, and Security*, Springer, 2013, pp. 33–44.

[70] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems*, vol. 36, no. 6, pp. 45–64, Dec. 2016.

[71] S. Alam, C. Lokan, G. Aldis, S. Barry, R. Butcher, and H. Abbass, "Systemic identification of airspace collision risk tipping points using an evolutionary multi-objective scenario-based methodology," *Transportation Research Part C: Emerging Technologies*, vol. 35, pp. 57–84, Oct. 2013.

[72] D. Thipphavong, "Accelerated monte carlo simulation for safety analysis of the advanced airspace concept," American Institute of Aeronautics and Astronautics, Sep. 13, 2010.

[73] D. A. Hsu, "The evaluation of aircraft collision probabilities at intersecting air routes," *The Journal of Navigation*, vol. 34, no. 1, pp. 78–102, Jan. 1981.

[74] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2011, pp. 254–257.

[75] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Jul. 15, 2010, pp. 167–170.

[76] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *NASA Formal Methods*, ser. Lecture Notes in Computer Science, Springer, Cham, Apr. 27, 2015, pp. 127–142.

[77] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Sim-ATAV: Simulation-based adversarial testing framework for autonomous vehicles," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*, ser. HSCC '18, New York, NY, USA: ACM, 2018, pp. 283–284.

[78] S. Shan and G. G. Wang, "Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions," *Structural and Multidisciplinary Optimization*, vol. 41, no. 2, pp. 219–241, Mar. 2010.

[79] K. Crombecq, E. Laermans, and T. Dhaene, "Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling," *European Journal of Operational Research*, vol. 214, no. 3, pp. 683–696, Nov. 2011.

[80] K. Crombecq, L. De Tommasi, D. Gorissen, and T. Dhaene, "A novel sequential design strategy for global surrogate modeling," in *Winter Simulation Conference*, Winter Simulation Conference, 2009, pp. 731–742.

[81] T. Hall, M. M. Dabbeeru, and S. K. Gupta, "A new approach for explicit construction of moldability based feasibility boundary for polymer heat exchangers," in *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, 2011, pp. 863–871.

[82] J. Cevallos, S. Gupta, and A. Bar-Cohen, "Incorporating moldability considerations during the design of polymer heat exchangers," *Journal of Mechanical Design*, vol. 133, no. 8, p. 081 009, 2011.

[83] J. Eason and S. Cremaschi, "Adaptive sequential sampling for surrogate model generation with artificial neural networks," *Computers & Chemical Engineering*, vol. 68, pp. 220–232, Sep. 2014.

[84] A. G. Banerjee, A. Balijepalli, S. K. Gupta, and T. W. LeBrun, "Generating simplified trapping probability models from simulation of optical tweezers system," *Journal of Computing and Information Science in Engineering*, vol. 9, no. 2, p. 021 003, 2009.

[85] J. P. Kleijnen, W. v. Beers, and I. v. Nieuwenhuyse, "Constrained optimization in expensive simulation: Novel approach," *European Journal of Operational Research*, vol. 202, no. 1, pp. 164–174, Apr. 2010.

[86] A. Ajdari and H. Mahlooji, "An adaptive exploration-exploitation algorithm for constructing metamodels in random simulation using a novel sequential experimental design," *Communications in Statistics - Simulation and Computation*, vol. 43, no. 5, pp. 947–968, Jan. 2014.

[87] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, 3. print, ser. Adaptive computation and machine learning. Cambridge, Mass.: MIT Press, 2008, 248 pp.

[88] E. Snelson, "Tutorial: Gaussian process models for machine learning," *Gatsby Computational Neuroscience Unit, UCL*, 2006.

[89] B. C. Shah, P. Švec, I. R. Bertaska, A. J. Sinisterra, W. Klinger, K. von Ellenrieder, M. Dhanak, and S. K. Gupta, "Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic," *Autonomous Robots*, vol. 40, no. 7, pp. 1139–1163, 2016.

[90] J. D. Langsfeld, K. N. Kaipa, and S. K. Gupta, "Selection of trajectory parameters for dynamic pouring tasks based on exploitation-driven updates of local metamodels," *Robotica*, vol. 36, no. 1, pp. 141–166, 2018.

[91] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, May 27, 2015.

[92] A. M. Kabir, J. D. Langsfeld, K. N. Kaipa, and S. K. Gupta, "Identifying optimal trajectory parameters in robotic finishing operations using minimum number of physical experiments," *Integrated Computer-Aided Engineering*, pp. 1–25, Preprint 2018.

[93] J. D. Langsfeld, A. M. Kabir, K. N. Kaipa, and S. K. Gupta, "Integration of planning and deformation model estimation for robotic cleaning of elastically deformable objects," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 352–359, 2018.

[94] Z. Wang and M. Ierapetritou, "A novel feasibility analysis method for black-box processes using a radial basis function adaptive sampling approach," *AICHE Journal*, vol. 63, no. 2, pp. 532–550, Feb. 2017.

[95] A. Basudhar and S. Missoum, "An improved adaptive sampling scheme for the construction of explicit boundaries," *Structural and Multidisciplinary Optimization*, vol. 42, no. 4, pp. 517–529, Oct. 2010.

[96] J. Fan, Y. Fan, and Y. Wu, "High-dimensional classification," *High-dimensional Data Analysis*, pp. 3–37, 2011.

[97] S. Shan and G. G. Wang, "Metamodeling for high dimensional simulation-based design problems," *Journal of Mechanical Design*, vol. 132, no. 5, p. 051 009, 2010.

[98] X. Cai, H. Qiu, L. Gao, P. Yang, and X. Shao, "An enhanced RBF-HDMR integrated with an adaptive sampling method for approximating high dimensional problems in engineering design," *Structural and Multidisciplinary Optimization*, vol. 53, no. 6, pp. 1209–1229, Jun. 2016.

[99] Z. Huang, H. Qiu, M. Zhao, X. Cai, and L. Gao, "An adaptive SVR-HDMR model for approximating high dimensional problems," *Engineering Computations*, vol. 32, no. 3, pp. 643–667, May 5, 2015.

[100] G. Li, J. Hu, S.-W. Wang, P. G. Georgopoulos, J. Schoendorf, and H. Rabitz, "Random sampling-high dimensional model representation (RS-HDMR) and orthogonality of its different order component functions," *The Journal of Physical Chemistry A*, vol. 110, no. 7, pp. 2474–2485, Feb. 2006.

[101] W. Liao, M. Maggioni, and S. Vigogna, "Learning adaptive multiscale approximations to data and functions near low-dimensional sets," in *Information Theory Workshop (ITW), 2016 IEEE*, IEEE, 2016, pp. 226–230.

[102] O. Pujol and D. Masip, "Geometry-based ensembles: Toward a structural characterization of the classification boundary," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 6, pp. 1140–1146, Jun. 2009.

[103] O. Day and T. M. Khoshgoftaar, "A survey on heterogeneous transfer learning," *Journal of Big Data*, vol. 4, no. 1, Dec. 2017.

[104] J.-B. Mouret and K. Chatzilygeroudis, "20 years of reality gap: A few thoughts aboutm simulators in evolutionary robotics," ACM Press, 2017, pp. 1121–1124.

[105] S. Koos, J. B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, Feb. 2013.

[106] A. Boeing and T. Bräunl, "Leveraging multiple simulators for crossing the reality gap," in *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, Dec. 2012, pp. 1113–1119.

[107] A Ligot, "On mimicking the effects of the reality gap with simulation-only experiments," *IRIDIA – Technical Report Series*, Mar. 2018.

[108] D. Ha and J. Schmidhuber, "World models," *ARXIV:1803.10122 [cs, stat]*, Mar. 27, 2018. arXiv: 1803.10122.

[109] W. Honig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 5382–5387.

[110] D. H. Scheidt, "Organic persistent intelligence, surveillance, and reconnaissance," *Johns Hopkins APL Technical Digest*, vol. 31, no. 2, pp. 167–174, 2012.

[111] Z. Kingston, M. Moll, and L. E. Kavraki, "Decoupling constraints from sampling-based planners," in *International Symposium of Robotics Research 2017*, 2017.

[112] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM - IEEE conference publication," in *Robotics and Automation (ICRA), 2000 IEEE International Conference on*, vol. 1, 2000, pp. 521–528.

[113] B. Burns and O. Brock, "Single-query entropy-guided path planning," in *Robotics and Automation (ICRA), 2005 IEEE International Conference on*, Apr. 2005, pp. 2124–2129.

[114] V. Boor, M. H. Overmars, and A. F. v. d. Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *Robotics and Automation (ICRA), 1999 IEEE International Conference on*, vol. 2, 1999, 1018–1023 vol.2.

[115] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[116] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Robotics and Automation (ICRA), 2009 IEEE International Conference on*, IEEE, May 2009, pp. 625–632.

[117] D. Berenson, T. Siméon, and S. S. Srinivasa, "Addressing cost-space chasms in manipulation planning," in *Robotics and Automation, 2011. ICRA 2008. IEEE International Conference on*, IEEE, May 2011, pp. 4561–4568.

[118] M. Norouzi, J. V. Miro, and G. Dissanayake, "Planning stable and efficient paths for reconfigurable robots on uneven terrain," *Journal of Intelligent & Robotic Systems*, vol. 87, no. 2, pp. 291–312, 2017.

[119] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, Jan. 1, 2014.

[120] D. J. Kriegman, "Let them fall where they may: Capture regions of curved objects and polyhedra," *The International Journal of Robotics Research*, vol. 16, no. 4, pp. 448–472, Aug. 1, 1997.

[121] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason, "Planar manipulation on a conveyor with a one joint robot," in *Robotics Research*, Springer, London, 1996, pp. 265–276.

[122] T. Abell and M. Erdmann, "Stably supported rotations of a planar polygon with two frictionless contacts," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 3, 1995, pp. 411–418.

[123] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–35, Apr. 6, 2017.

[124] J. D. Langsfeld, "Learning task models for robotic manipulation of nonrigid objects," PhD thesis, University of Maryland, College Park, 2017.

[125] C. Finn, P. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," *ARXIV preprint arXiv:1611.03852*, 2016.

[126] J. D. Langsfeld, K. N. Kaipa, R. J. Gentili, J. A. Reggia, and S. K. Gupta, "Incorporating failure-to-success transitions in imitation learning for a dynamic pouring task," presented at the Workshop on Compliant Manipulation: Challenges and Control, Chicago, IL, 2014, p. 4.

[127] M. Hausknecht, Y. Chen, and P. Stone, "Deep imitation learning for parameterized action spaces," presented at the AAMAS Adaptive Learning Agents (ALA) Workshop, Singapore, May 2016.

[128] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 4565–4573.

[129] H. M. Le, P. Carr, Y. Yue, and P. Lucey, "Data-driven ghosting using deep imitation learning," p. 15, 2017.

[130] B. Hoehn, "The effectiveness of opponent modelling in a small imperfect information game degree," PhD thesis, 2006.

[131] F. Southey, M. P. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' bluff: Opponent modelling in poker," *ARXIV preprint arXiv:1207.1411*, 2012.

[132] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust, "Opponent behaviour recognition for real-time strategy games.," *Plan, Activity, and Intent Recognition*, vol. 10, p. 05, 2010.

[133] G. E. Mullins and S. K. Gupta, "Adversarial blocking techniques for autonomous surface vehicles using model-predictive motion goal computation," in *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, Sep. 2015, pp. 2272–2278.

[134] H. He, U. EDU, J. Boyd-Graber, and H. Daumé III, "Opponent modeling in deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 1804–1813.

[135] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[136] K. Kelchtermans and T. Tuytelaars, "How hard is it to cross the room? – training (recurrent) neural networks to steer a UAV," *ARXIV:1702.07600 [cs]*, Feb. 24, 2017. arXiv: 1702.07600.

[137] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," *ARXIV:1707.02515 [cs]*, Jul. 8, 2017. arXiv: 1707.02515.

[138] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *ARXIV:1605.06450 [cs]*, May 20, 2016. arXiv: 1605.06450.

[139] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive UAV control in cluttered natural environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 1765–1772.

[140] A. Jacoff, E. Messina, B. A. Weiss, S. Tadokoro, and Y. Nakagawa, "Test arenas and performance metrics for urban search and rescue robots," in *Intelligent Robots and Systems (IROS 2003), 2003 IEEE/RSJ International Conference on*, vol. 4, IEEE, 2003, pp. 3396–3403.

[141] R. A. Paielli, "Automated generation of air traffic encounters for testing conflict-resolution software," *Journal of Aerospace Information Systems*, vol. 10, no. 5, pp. 209–217, May 2013.

[142] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 1443–1450.

[143] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, J. D. Appler, M. H. Biggins, K. Chiou, M. A. Huntley, J. D. Stewart, and A. S. Watkins, "Delivering test and evaluation tools for autonomous unmanned vehicles to the fleet," *Johns Hopkins APL technical digest*, vol. 33, no. 4, pp. 279–288, 2017.

[144] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *Journal of Systems and Software*, vol. 137, pp. 197–215, 2018.

[145] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[146] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, Jun. 2016.

[147] J. Yu, "Nearest-neighbor density estimation," *Encyclopedia of Statistical Sciences*, 1998.

[148] N. Wang and A. E. Raftery, "Nearest-neighbor variance estimation (NNVE)," *Journal of the American Statistical Association*, vol. 97, no. 460, pp. 994–1019, 2002.

[149] A. Agresti and M. Kateri, *Categorical data analysis*. Springer, 2011, pp. 206–208.

[150] C. Strobl, *Statistical issues in machine learning: Towards reliable split selection and variable importance measures*. Cuvillier Verlag, 2008.

[151] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.

[152] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.

[153] D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq, "A surrogate modeling and adaptive sampling toolbox for computer based design," *Journal of Machine Learning Research*, vol. 11, pp. 2051–2055, Jul 2010.

[154] R. Chalmers, D. Scheidt, T. Neighoff, S. Witwicki, and R. Bamberger, "Cooperating unmanned vehicles," in *AIAA 1st Intelligent Systems Technical Conference*, 2004, pp. 1–8.

[155] T. I. Fossen, *Guidance and control of ocean vehicles*. John Wiley & Sons Inc, 1994.

[156] P. R. Roan, A. Burmeister, A. Rahimi, K. Holz, and D. Hooper, "Real-world validation of three tipover algorithms for mobile robots," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 4431–4436.

[157] H. Schempf, E. Mutschler, C. Piepgras, J. Warwick, B. Chemel, S. Boehmke, W. Crowley, R. Fuchs, and J. Guyot, "Pandora: Autonomous urban robotic reconnaissance system," in *Robotics and Automation (ICRA), 1999 IEEE International Conference on*, vol. 3, IEEE, 1999, pp. 2315–2321.

[158] H. G. Nguyen and J. P. Bott, "Robotics for law enforcement: Applications beyond explosive ordnance disposal," in *Enabling Technologies for Law Enforcement*, International Society for Optics and Photonics, 2001, pp. 433–454.

[159] R. Murphy, J. Casper, J. Hyams, M. Micire, and B. Minten, "Mobility and sensing demands in USAR," in *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE*, vol. 1, IEEE, 2000, pp. 138–142.

[160] R. R. Murphy, "Trial by fire [rescue robots]," *IEEE Robotics & Automation Magazine*, vol. 11, no. 3, pp. 50–61, 2004.

[161] E. Tunstel, "Evolution of autonomous self-righting behaviors for articulated nanorovers," in *Artificial Intelligence, Robotics and Automation in Space*, vol. 440, 1999, p. 341.

[162] M. I. Wallace, J. F. Burn, R. A. Hyde, C. Melhuish, and T. Pipe, "Biologically inspired solutions for compliant limb UGVs," in *2nd SEAS DTC Technical Conference*, Citeseer, 2007.

[163] P. Ben-Tzvi, A. A. Goldenberg, and J. W. Zu, "Design, simulations and optimization of a tracked mobile robot manipulator with hybrid locomotion and manipulation capabilities," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 2307–2312.

[164] S. Peng, X. Ding, F. Yang, and K. Xu, "Motion planning and implementation for the self-recovery of an overturned multi-legged robot," *Robotica*, vol. 35, no. 5, pp. 1107–1120, May 2017.

[165] N. B. Zumel and M. A. Erdmann, "Balancing of a planar bouncing object," in *Robotics and Automation (ICRA), 1994 IEEE International Conference on*, IEEE, 1994, pp. 2949–2954.

[166] C. C. Kessens, D. C. Smith, and P. R. Osteen, "A framework for autonomous self-righting of a generic robot on sloped planar surfaces," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, 2012, pp. 4724–4729.

[167] J. Collins, C. C. Kessens, and S. J. Biggs, "Proprioceptive sensing for autonomous self-righting on unknown sloped planar surfaces," in *Robotic and Sensors Environments (ROSE), 2013 IEEE International Symposium on*, IEEE, 2013, pp. 160–165.

[168] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[169] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 951–958, Dec. 2001.

[170] M. E. Henderson, "Multiple parameter continuation: Computing implicitly defined k-manifolds," *International Journal of Bifurcation and Chaos*, vol. 12, no. 3, pp. 451–476, Mar. 1, 2002.

[171] M. A. Hinton, J. M. Burck, K. R. Collins, M. S. Johannes, E. W. Tunstel Jr, and M. J. Zeher, "Integration of advanced explosive ordnance disposal robotic systems within a modular open systems architecture," *Johns Hopkins APL technical digest*, vol. 32, no. 3, p. 595, 2013.

[172] G. E. Mullins, A. G. Dress, J. D. Appler, P. G. Stankiewicz, and S. K. Gupta, "Accelerated testing and evaluation of autonomous vehicles via imitation learning," in *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, IEEE, 2018.

[173] A. Santara, A. Naik, B. Ravindran, D. Das, D. Mudigere, S. Avancha, and B. Kaul, "RAIL: Risk-averse imitation learning," *ARXIV:1707.06658 [cs]*, Jul. 20, 2017. arXiv: 1707.06658.

[174] A. Rajeswara, S. Ghotr, B. Ravindra, and S. Levin, *EPOPT: Learning robust neural network policies using model ensembles*. Upper Saddle River, NJ: Prentice Hall, 2016, 596 pp., OCLC: 32698477.

[175] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980.

[176] I. Kamon, E. Rimon, and E. Rivlin, "Tangentbug: A range-sensor-based navigation algorithm," *The International Journal of Robotics Research*, vol. 17, no. 9, pp. 934–953, 1998.

[177] S. Petillo, A. Balasuriya, and H. Schmidt, "Autonomous adaptive environmental assessment and feature tracking via autonomous underwater vehicles," in *OCEANS 2010 IEEE-Sydney*, IEEE, 2010, pp. 1–9.

[178] S. Surjanovic and D. Bingham. (Jan. 2015). Optimization test functions and datasets, Virtual Library of Simulation Experiments, [Online]. Available: http://www.sfu.ca/~ssurjano/optimization.html (visited on 05/12/2017).

[179] P. I. Corke, "Robotics toolbox," *Obtained from Peter O. Corke site: Http://www. petercorke. com/Robotics% 20Toolbox. html*, 2002.

[180] M. Sheckells, G. Garimella, and M. Kobilarov, "Robust policy search with applications to safe vehicle navigation," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 2343–2349.