2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Automated Generation of Diverse and Challenging Scenarios for Test and Evaluation of Autonomous Vehicles

Galen E. Mullins[1], Paul G. Stankiewicz [2], and
Satyandra K. Gupta[3], *Senior Member, IEEE*

***Abstract*— We propose a novel method for generating test scenarios for a black box autonomous system that demonstrate critical transitions in its performance modes. In complex environments it is possible for an autonomous system to fail at its assigned mission even if it complies with requirements for all subsystems and throws no faults. This is particularly true when the autonomous system may have to choose between multiple exclusive objectives. The standard approach of testing robustness through fault detection is directly stimulating the system and detecting violations of the system requirements. Our approach differs by instead running the autonomous system through full missions in a simulated environment and measuring performance based on high-level mission criteria. The result is a method of searching for challenging scenarios for an autonomous system under test that exercise a variety of performance modes. We utilize adaptive sampling to intelligently search the state space for test scenarios which exist on the boundary between distinct performance modes. Additionally, using unsupervised clustering techniques we can group scenarios by their performance modes and sort them by those which are most effective at diagnosing changes in the autonomous system's behavior.**

I. INTRODUCTION

Effective test and evaluation (T&E) of autonomous vehicles remains an open problem in the testing community. Traditional testing has focused on the physical components of a system to ensure safe and reliable operation. The very nature of autonomous systems, however, dictates that the reasoning component, i.e. the "brains", of the system must be effectively tested as well. A great deal of recent work has focused on applying software testing techniques to autonomous systems with an enormous amounts of inputs such as fault detection [1] or model checking [2] of the underlying decision engine. This can provide some runtime assurances for the robustness of the software, but it does not provide information about the how the autonomous system will perform when executing a mission. Additionally, these testing techniques do not provide insight into the environmental factors that contribute to the autonomous system's decision process.

For example, consider an unmanned underwater vehicle (UUV) tasked with a survey mission. The multiple subsystems and behavioral modes of the UUV must work in concert in the presence of competing priorities. For example, it must offset the risk of detection when surfacing with the need to localize itself via GPS. It also it must determine if it has enough fuel to complete the survey or if uncertainties in its environment make the effort too risky and it must return to the base early. This is of particular concern for long duration missions where the vehicle must transition among multiple mission objectives [3]. As such it can be difficult to provide guarantees of the system's decision-making capabilities without running extensive tests in a variety of challenging scenarios. This requires both a simulation framework capable of exercising the autonomous system realistically [4] and a suite of tests that provide coverage of the mission space [5].

To do this, we focus our attention on the regions in the configuration 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 vehicle to take a different path and fail to reach its goal. Understanding where these transitions occur is key to predicting the performance of the autonomous vehicles and is useful for both the design and validation of the system. They are particularly useful for identifying the triggers for specific behaviors, such as the strength of a current which overcomes an obstacle avoidance strategy. These can then be utilized for repairing bugs or simply understanding the likelihood of a certain behavior being triggered for that region of the configuration space.

One issue immediately encountered is that the number of configuration parameters for the scenario quickly increases when attempting to capture realistic missions. Moving and static obstacles, tidal and constant currents, time windows for objectives, and other environmental factors are just a few of the different parameters on which an engineer may wish to test a UUV. As the number of parameters increase the number of samples required to maintain the same resolution grows exponentially. In addition if the software being tested is not capable of running faster than real time a single scenario taking hours to run to completion.

Given the limitations of computational resources and the enormity of the testing space, we require a sampling method that maximizes the information returned given a limited number of runs. This involves both preferentially returning samples from performance boundary regions rather than spending resources exploring regions of stable performance and exploring along the entire performance boundary. This strategy is similar to using active learning to train a classifier,

[1]Galen E. Mullins is with the Department of Mechanical Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA gmullins@umd.edu

[2]Paul G. Stankiewicz is with the Johns Hopkins University Applied Physics Laboratory, 11100 Johns Hopkins Road, Laurel, Maryland 20723, USA paul.stankiewicz@jhuapl.edu

[3]S.K. Gupta is with the Department of Aerospace and Mechanical Engineering and Center for Advanced Manufacturing, University of Southern California, Los Angeles, California 90089, USA skgupta@usc.edu

978-1-5090-4633-1/17/$31.00 ©2017 IEEE

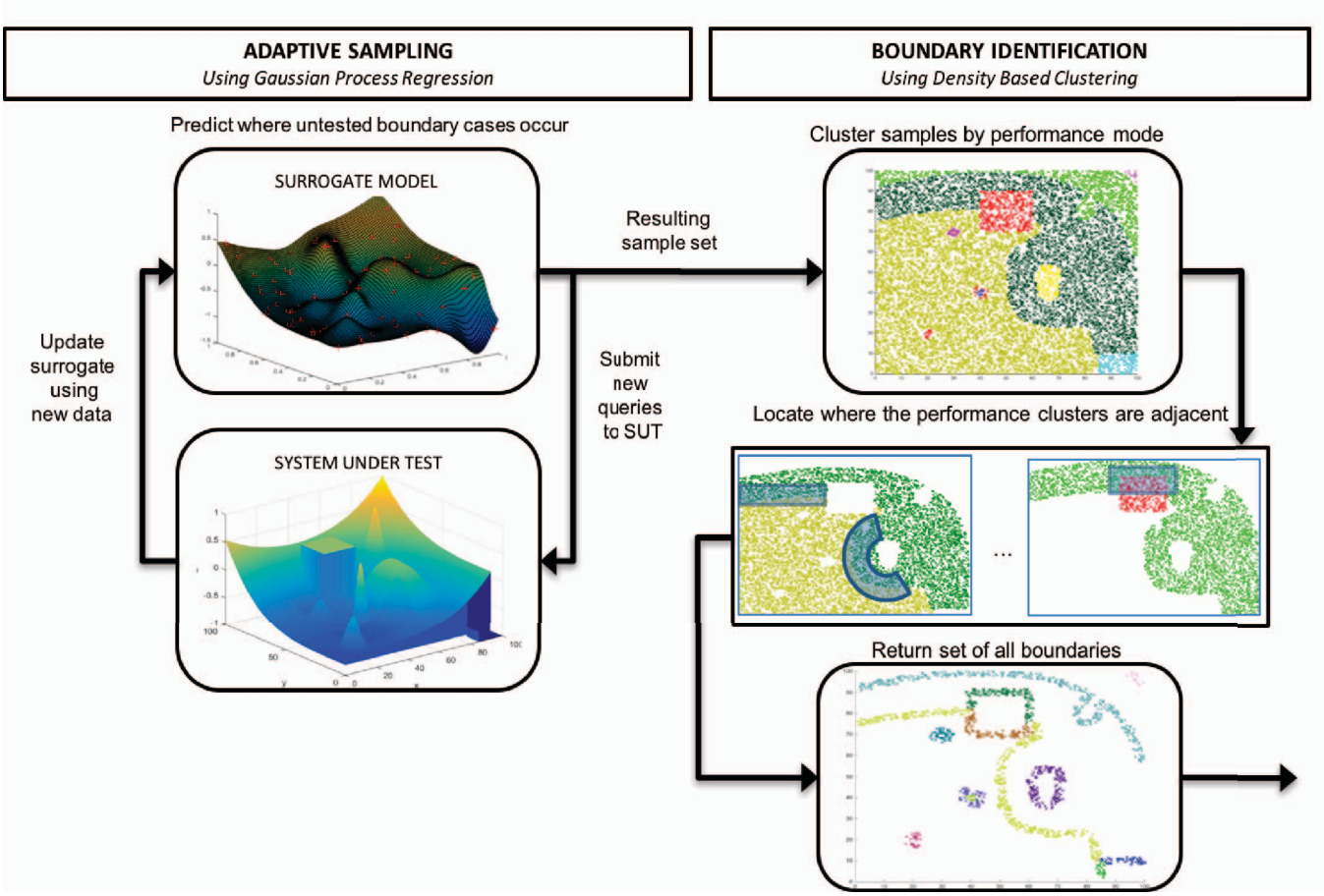


Fig. 1: An overview of the test generation cycle for a system with continuous unlabeled outputs.

where the highest information cases lie along the decision boundary and thus the algorithm will preferentially sample in these regions [6]. Furthermore given that running tests is restricted by time rather than number of queries, this search technique cannot incur a significant amount of overhead above and beyond what the simulations already demand. This means we require a method that scales well with both number of samples and number of dimensions in order to properly capture the complexity of realistic missions. In this paper, we introduce a novel adaptive search technique designed for the purpose of discovering performance boundaries of a system that scales better than previous approaches with number of samples and number of dimensions. We also introduce a technique for identifying performance boundaries through unsupervised clustering and adjacency tests.

In this paper we refer to the regions in the parameter space where transitions in behaviors occur as the performance boundaries of the autonomous system. Our objective is to utilize active learning methodology to effectively sample the scenario parameter space and automatically identify the performance boundary cases. In this paper we discuss the two parts of this learning process - first, using adaptive sampling to search the parameter space followed by unsupervised learning to determine the performance modes and the cases that make up the performance boundaries. An overview of our approach is shown in Figure 1

II. RELATED WORK

The validation and verification of autonomous systems has been an incredibly active area of research in the past few years, particularly the use of active learning for test case generation. Surrogate optimization of complex systems has similarly been an increasingly popular tool for design of experiments. Here we discuss recent work in the fields of verification of autonomous systems and surrogate model generation and how it relates to the work in this paper.

Test scenario generation has been an active area of research in the software testing domain for some time [7]. The current focus entails generating tests for requirements verification, ensuring that the software does not throw faults, and ensuring that the hardware meets its reliability specifications. One testing method is to simply stimulate the system with an enormous amount of inputs for fault detection [8] [1], often utilizing optimization-based combination testing [9] to minimize the size of the test suite while maximizing coverage. Additionally, sampling-based methods have been used to discover different performance modes for a car control system [10]. All of these testing methods provide runtime assurances for the robustness of the software, but they do not provide information about the how the autonomous system will perform when executing a mission. Put another way, current software testing validates that the system can

accommodate receiving bad inputs or sending bad outputs; it does not necessarily validate the actual decisions made by the underlying algorithms.

Generating and evaluating test scenarios which stress the autonomous system under test in simulation has been explored with success in the past [11]. For example, the work in [12] used this strategy to find types of multi-UAV encounters that stressed the autonomous system's conflict resolution algorithms. Evolutionary generation techniques are frequently used for this purpose [13] with objective functions specifically designed for the domain being tested. The concept of discovering simulation cases based on their difficulty has also been explored for the navigation of ground vehicles in a 3D environment [14]. These results are encouraging and demonstrate that test generation techniques can be applied to a variety of domains. The open challenge that remains is determining which search strategy is the most efficient at delivering the relevant test cases with the fewest required simulations.

Validation of autonomous systems for critical missions has been increasing in importance as robots have become more prevalent in ordnance disposal, search-and-rescue, and other applications that have very low tolerance for failure [15]. Of particular interest are methods of providing performance guarantees based on model-checking and formal methods [2] [16]. These methods require that a model which fully describes the autonomous system performance can be generated and exhaustively tested for exceptions which break the specifications. Models that have been used in the past include finite state machines [1] and process algebras [16]. The drawback of these techniques is that the resulting model must fully describe the autonomous system 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.

III. PROBLEM FORMULATION

What differentiates our process from previous works is the concept of performance boundaries. As described earlier, performance boundaries are regions of the testing space where the performance of the system under test (SUT) is uncertain, i.e. small alterations to the scenario configuration can cause transitions in the SUT behaviors which result in large performance changes. In this section we more formally define the search and boundary identification problem as well as the terms used throughout the paper.

A. System under test

Our target SUT is the decision making software for an autonomous vehicle executing a mission in a simulated environment. It takes a scenario configuration as an input and returns a set of score metrics of its mission performance as output. This score is based on externally observable attributes that would be associated with mission requirements such as time elapsed, path taken, and completion of objectives. While the output of the system is a set of continuous values, these can be mapped to discrete behaviors or performance modes of the autonomous system. The test cases we consider to be the most informative are those which occur on 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 where performance is constant and known, i.e. cases where the system will almost surely succeed or conversely cases where the system will almost surely fail. Much more information about the system is gained by testing in regions where critical decisions must be made by the autonomous system 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.

B. 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 autonomous system 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 samples who are each others 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 $X_j \in B_{a,b}$.

C. Problem Statement

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:

$$\Gamma(\mathcal{F}, \mathcal{X}^n, \mathcal{Y}^m, N) = \mathcal{L}^N. \tag{1}$$

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.

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

2) Boundary Identification Problem: We formally define the boundary identification algorithm as a function

$$\mathcal{C}(\mathcal{L}) = \mathcal{B} \tag{2}$$

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

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

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.

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.

D. Overview of Approach

The approach presented in this paper is broken into two primary phases: search and identification. During the search phase we utilize an adaptive sampling or active learning approach to select new test cases that are run by the autonomous system simulation. This process utilizes Gaussian Process Regression [17] to model the autonomous system's performance and preferentially select regions that might indicate performance boundaries. The high dimensionality of the state space for an autonomous system 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.

IV. SEARCH STRATEGY

A. 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 (LH) or Sobol sequences, which attempt to optimize uniform coverage and density and are precomputed based upon the size of the state space. In this paper we utilize a generalized method for adaptive sampling which allows for changing the underlying meta-models and information metric. This is more formally defined in Algorithm 1. The adaptive algorithm uses the normalized unit states $\bar{X}$ and scores $\bar{Y}$ for the information metrics.

B. Boundary Information Metrics

There are multiple query strategies that can be used for adaptive sampling including entropy, model improvement, uncertainty, and density. Our objective is to drive the search towards performance boundaries; thus, we have designed our metrics to look for areas with high gradients that have not been sampled yet. This is similar to the exploration-exploitation approach of the LOLA-Voronoi algorithm [18] 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 number of samples and input dimensionality. For n points in $\mathbb{R}^d$ it takes $O(nlogn + n^{\lceil d/2 \rceil})$, making it infeasible for higher dimensional problems. Therefore techniques which provide better scaling with number of samples and input parameters.

We introduce two new meta-model metrics for the purpose of discovering performance boundaries: one which uses 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 quality of a proposed query as an output.

For each query the GPR meta-model returns the mean value μ, the first-order gradient of the mean $\bigtriangledown\mu$, and the

estimated co-variance σ. This 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: $\mathcal{M}_{GPR}(X) = (|\bigtriangledown \mu(X)|)^g \cdot (\sigma(X))^v$ 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) meta-model estimates the size and gradiant of a query using its nearest neighbors. For a given query state it returns the scores and distances of its k nearest neighbors. It then computes the variance σ_K of the scores and the mean distance d_K of its neighbors. The NNDV evaluator is then computed as follows: $\mathcal{M}_{NNDV}(X) = (\sigma_K(X))^g \cdot (d_K(X))^v$ where g and v are the same tuning parameters used in the GPR meta-model evaluator.

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 normalized sample set $[\bar{X}, \bar{Y}]$
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 $\mathcal{L}$

V. BOUNDARY IDENTIFICATION

A. Identifying Performance Modes

One of the issues of black box testing is that we cannot look inside the decision engine to determine which behavior the autonomous system is executing. Instead, we must use externally observable states 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 autonomous system 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 order to apply our techniques to systems which provide continuous unlabeled outputs we utilize Mean-Shift [19] clustering 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 [20] in in space to identify distinct regions of interest. We selected these two clustering algorithms as our classification method because they do not require a priori knowledge of the shapes of the boundaries between classes or the number of classes.

B. Cluster Stitching

To obtain the boundaries from these clusters, we perform a pair-wise comparison between every cluster with a differing performance mode. We utilize a k-nearest neighbor detection algorithm to determine the closest neighbor in the adjacent cluster for each sample. Any samples that are within ϵ_B distance of their nearest neighbor in the opposite cluster are added to the final boundary set. The pair distances in the boundary set are then used to determine how close the samples are to the performance boundary. This approach is defined further in Algorithm 2.

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

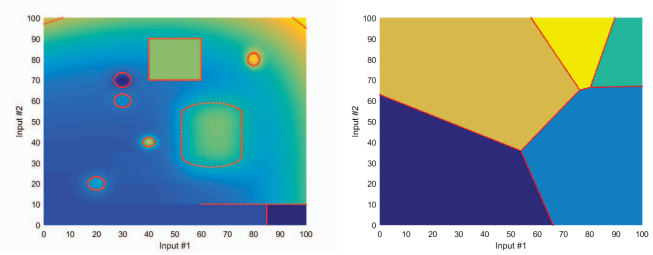
Input: A set N of labeled samples $\mathcal{L}$ containing the input states X^N and output scores Y^N
Output: A set of identified performance modes, a collection of boundaries $\mathcal{B}$, and distance estimate vector D
Let λ_P be the threshold distance for the flat kernel mean shift function, ϵ_C 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
$\hat{\mathbf{C}}_\mathbf{Y} = DBSCAN(\bar{X_{P_L}}, \epsilon_C, n_{min})$
Append the subclusters to the complete set of clusters
$\hat{\mathbf{C}} \frown [\hat{\mathbf{C}}_\mathbf{Y}]$
end for
for all $\hat{\mathbf{C}}_{\mathbf{Y}i}$ and $\hat{\mathbf{C}}_{\mathbf{Y}j} \in \hat{\mathbf{C}} | Y_i \neq Y_j$ **do**
$D_{ij} = knnsearch(\bar{X_{P_i}}, \bar{X_{P_j}})$
$\mathcal{B}_{ij} = [X_{P_i}, X_{P_j}, Y_i, Y_j] \forall \bar{X_{P_i}}, X_{P_j} | D_{ij} < D_\epsilon$
end for
return $\mathcal{B}$

VI. RESULTS

A. Test Systems

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

1) Synthetic Test Functions: 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) Custom2D (b) Plates2D

Fig. 2: Top-down view of synthetic 2D functions with outlines of true boundaries: (a) Custom2d (b) Plates2d

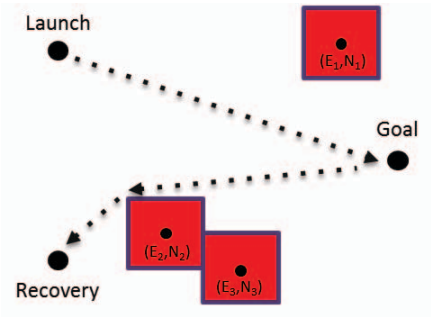


Fig. 3: Illustration of the autonomous UUV scenario described in Section VI-A.2 .

- Custom 2D - Two input dimensions with one continuous unlabeled output. It contains peaks, valleys, plateaus and cliffs as features of interest and is the function illustrated in Figure 1.
- Plates 2D - Two input dimensions with one discrete output. There are 5 score categories.
- Plates 3D - Three input dimensions with one discrete output. There are 5 score categories.

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

2) UUV Scenario: In addition to test functions, the search and identification algorithms were also evaluated on an autonomous vehicle simulation. For the autonomous vehicle we chose a simulated UUV operating under a multi-objective navigation scenario. The objective of the mission was to travel from a starting point to a goal waypoint and then return to a separate recovery point (all of which are fixed) before running out of battery. The goal waypoint was placed on the opposite side of a 2km operational area from the start and recovery points. Additionally, inside the operational area were three square obstacles that are 400m on a side that could vary in the East and North directions. Thus, the centers for the three obstacles form our six input state dimensions, $X = [E_1, N_1, E_2, N_2, E_3, N_3]$. This scenario is illustrated in Figure 3.

The score space for the UUV scenario consisted of two binary performance labels: mission success and safety success. The scenario was labeled as a mission success if the UUV achieved its goal of making it to the waypoint. The scenario was labeled as a safety success if the UUV returned to its recovery point with sufficient battery. Thus, four performance modes exist in this framework, consisting of the different combinations of the mission success and safety success criteria.

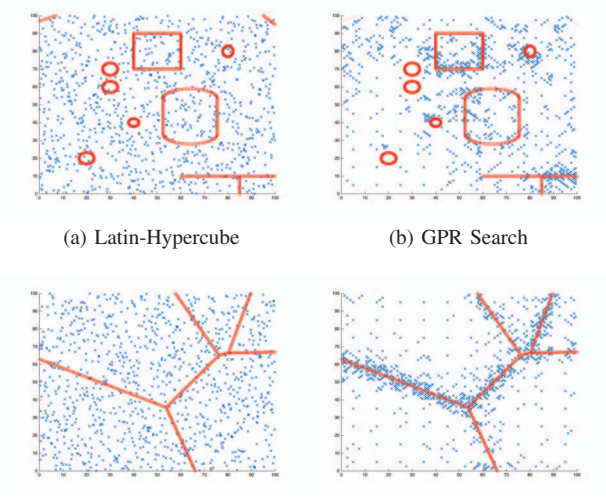
B. Search Performance on Synthetic Functions

We evaluated the performance of the search algorithms presented in Section IV based on their ability to identify features in test functions and sample near the performance boundaries. For comparison, we chose a Sobol sequence design as our baseline space-filling 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 [21] 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. Precision is defined as the percentage of samples from the entire set that within a distance of 0.01 units of a performance boundary. Coverage is the percentage of a performance boundary which has a sample within a distance of 0.01 units. Convergence is number of samples required to reach 90% coverage of all known boundaries. Runtime is simply the number of seconds required to collect the prescribed number of samples. The results of these tests are summarized in Table I.

The search methods introduced in this paper outperformed both the space-filling approaches as well as the popular LOLA-Voronoi adaptive search in all of our chosen metrics. This is particularly true in cases where the boundaries are sharply defined, as in our Plates2d test function. As shown in Figure 4, 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 cases of the Latin

Test System	LH Design	Sobol Design	LOLA Voronoi	GPR Search	NNDV Search
Custom2d	Based on 1000 Samples				
Precision	6.52%	6.4%	9.53%	11.6%	19.2%
Coverage	29.6%	31.76%	49.0%	48.43%	59.2 %
Convergence	1401	801	1101	701	701
Runtime(sec)	0.177	0.791	27.2	2.96	0.645
Plates2d	Based on 1000 Samples				
Precision	5.10%	6.4%	6.58%	11.6%	19.2%
Coverage	30.9%	31.7%	39.4%	48.4%	59.2 %
Convergence	1151	1051	951	501	601
Runtime(sec)	0.177	0.791	31.9	2.96	0.64
Plates3d	Based on 3000 Samples				
Precision	3.33%	3.46%	4.22%	7.43%	12.17%
Coverage	1.28%	1.31%	1.526%	2.64%	4.65 %
Convergence	36501	26201	N/A	12001	7501
Runtime(sec)	0.081	0.233	246.0	32.7	2.12

TABLE I: Comparison of Search Methods



(a) Latin-Hypercube (b) GPR Search

(c) Latin-Hypercube (d) GPR Search

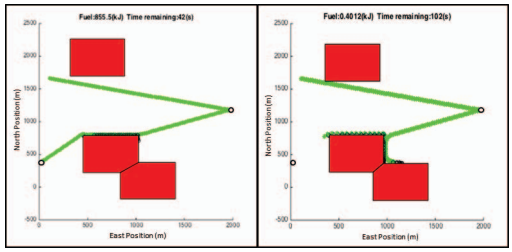
Fig. 4: Scatter plot of a GPR search vs. Latin hypercube sampling of the Custom2d (top) and Plates2d test function (bottom). Samples taken are in blue and the true locations of the boundaries are in red.

hypercube method. The results are even more pronounced for the NNDV search algorithm, with the added benefit of shorter runtime as well. One thing that becomes immediately apparent in the performance comparison between the Plates2d and Plates3d functions is the added dimension greatly increases the number of cases necessary to obtain coverage of the boundaries.

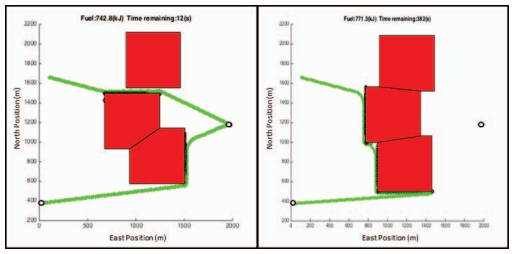
For the given number of samples, LOLA-Voronoi search did not distinguish itself significantly from a space-filling design. This is likely due to the fact that it was originally designed to minimize global model-fitting error. The techniques proposed in this paper have the different objective of finding boundary regions, resulting in sample sets that do not waste samples in low-gradient regions. Despite the superficial similarities in approach, the problem of identifying boundary regions in an unknown landscape is one that traditional adaptive sampling techniques are not suited.

C. Evaluation of UUV Simulation

While the test functions are useful for evaluating the performance of the search and identification algorithms against a priori truth boundaries, they cannot serve as a substitute for the performance space of an autonomous system under test. This section applies the GPR-based search algorithm and cluster stitching boundary identification algorithm to the UUV simulation, where the actual performance boundaries are unknown. A second data set was run using the Latin hypercube sampling method and boundaries identified using the same boundary identification algorithm.



(a) Boundary 1: Success (b) Boundary 1: Safety Fail



(c) Boundary 2: Success (d) Boundary 2: Mission Fail

Fig. 5: Examples of two performance boundaries identified for the UUV simulation. Boundary 1 (a)&(b) illustrates a boundary between safety success and safety failure, while Boundary 2 illustrates a boundary between mission success and mission failure.

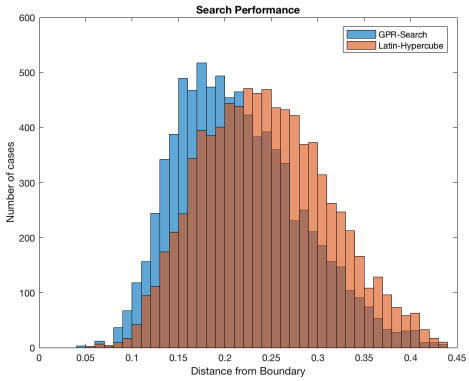


Fig. 6: The above histogram shows the distance to the boundary for all samples collected from the UUV simulation

Test System	Latin-Hypercube	GPR Search
Precision	1.76%	4.13%
Convergence	12500	4800
Mean distance	0.2647	0.2146
Min distance	0.053	0.043
Runtime (minutes)	82.75	95.66

TABLE II: Comparison of Search on UUV Scenario

After 7500 runs were performed using both GPR search and the Latin hypercube search, there were sufficient cases of all 4 potential performance modes to execute the boundary identification step. Mean-Shift identification method was set with a boundary distance threshold of 0.1 units. The Precision metric for this case was also set to 0.05 units. As the true locations of the boundaries were not known, the coverage metric was not applied. The system was determined to have converged when each boundary had at least 10 cases within the boundary distance threshold.

In nearly every regard the GPR search outperformed the space-filling approach as can be seen summarized in table II. The histogram presented in Figure 6 also shows how the distribution of cases trend closer to the boundary. The GPR search sample set contained examples of all 6 possible boundaries within the boundary threshold of 0.1 units while the Latin hypercube set contained examples from only 3 of the possible boundaries within the boundary threshold. In total, the GPR identified an average amount of 250 cases per boundary while the Latin hypercube only identified 20 cases per boundary identified. Convergence tests demonstrated the GPR capable of finding cases from all 6 boundaries within 3000 samples while a Latin hypercube method required approximately 12000 samples. Figure 5 shows examples of two performance boundaries with a pair of test cases representing each side of the boundary. In the first case a slight change in the obstacle causes the UUV to attempt to navigate to the left instead of the right due to it's heuristic for guessing the shorter path and failing to reach the recover point.

VII. CONCLUSIONS AND FUTURE WORK

In this work we introduced a process for intelligently discovering and identifying test cases for an autonomous system near its performance boundaries. These boundaries were defined as locations where a small change in the scenario configuration would cause a large change in the system's performance. By utilizing an adaptive sampling approach, we were able to reduce the number of samples necessary to find the features of interest. We also introduced a method for unsupervised clustering of the resulting samples in order to identify the queries which described the performance boundary. By combining these techniques we were able to generate sets of test cases for a multi-objective UUV mission that exercised different underlying decision processes of the autonomous system.

This research is still ongoing and there are many avenues that have yet to be explored. In particular, we are interested in methods for scaling our system to handle more test cases and a higher dimensionality. One approach is utilizing localized GPR techniques to speed up model generation, as well as investigation into non-stationary covariance functions to handle varying resolution of features. Another avenue of particular interest are sensitivity analyses to determine the impact of certain states on the performance of the system. In addition we are investigating more sophisticated anomaly detection methods as a possible augmentation to our current approach to boundary detection. Finally we are developing more realistic test scenarios for an autonomous UUV to include parameters such as ocean current, additional types of objectives, and a larger variety of obstacles.

REFERENCES

[1] 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. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-662-49224-6-1

[2] J. Choi, "Model checking for decision making behaviour of heterogeneous multi-agent autonomous system," Ph.D. dissertation, Cranfield University, 2012.

[3] 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, 2016.

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

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

[6] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, vol. 52, no. 55-66, p. 11, 2010.

[7] 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.

[8] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, 2008.

[9] 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, 2015.

[10] 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, 2016.

[11] 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.

[12] X. Zou, R. Alexander, and J. McDermid, "Testing method for multi-uav conflict resolution using agent-based simulation and multi-objective search," *Journal of Aerospace Information Systems*, pp. 191–203, 2016.

[13] 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.

[14] 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)*, 2016.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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, no. Jul, pp. 2051–2055, 2010.