MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models

Reza Matinnejad, Shiva Nejati, Lionel C. Briand
SnT Centre, University of Luxembourg,Luxembourg
{reza.matinnejad,shiva,nejati,lionel.briand}@uni.lu

Thomas Bruckmann
Delphi Automotive Systems,Luxembourg
thomas.bruckmann@delphi.com

ABSTRACT

Continuous controllers have been widely used in automotive domain to monitor and control physical components. These controllers are subject to three rounds of testing: Model-in-the-Loop (MiL), Software-in-the-Loop and Hardware-in-the-Loop. In our earlier work, we used meta-heuristic search to automate MiL testing of fixed configurations of continuous controllers. In this paper, we extend our work to support MiL testing of all feasible configurations of continuous controllers. Specifically, we use a combination of dimensionality reduction and surrogate modeling techniques to scale our earlier MiL testing approach to large, multi-dimensional input spaces formed by configuration parameters. We evaluated our approach by applying it to a complex, industrial continuous controller. Our experiment shows that our approach identifies test cases indicating requirements violations. Further, we demonstrate that dimensionally reduction helps generate surrogate models with higher prediction accuracy. Finally, we show that combining our search algorithm with surrogate modelling improves its efficiency for two out of three requirements.

Categories and Subject Descriptors [Software Engineering]: Software/Program Verification

Keywords: Search-based testing; continuous controllers; automotive software; dimensionality reduction; supervised learning.

1. INTRODUCTION

Embedded software systems are pervasive in the electronics system industry, e.g., automotive. Many embedded software systems are (partly) generated from *mathematical models* that describe how devices are monitored, controlled, or regulated [39, 18, 28, 15]. A well-known category of such models is *closed loop continuous controllers* [25], which are widely used in industrial control systems, and are designed using differential equations over continuous time. These controller models are typically specified in mathematical modeling environments, most notably Simulink [33], that support automatic transformation of models to source code. The generated code is then integrated with other necessary software components and deployed on embedded devices.

ASE'14, September 15-19, 2014, Vasteras, Sweden.

http://dx.doi.org/10.1145/2642937.2642978.

To identify early design errors of continuous controllers, engineers create a model of the environment, capturing the behavior of the device that interacts with a controller, and perform testing and simulations of the controller and the environment models. This stage of testing is known as *Model-in-the-Loop (MiL)* [39] testing and is performed in various embedded system sectors such as the automotive domain. The subsequent stages of controller development are referred to as *Software-in-the-Loop (SiL)* [39], where the integrated software system is developed and tested, and *Hardware-in-the-Loop (HiL)* [39], where the software deployed on the embedded device is being tested using a real-time simulator. Compared to SiL and HiL, the development and testing at MiL level are considerably faster as the engineers can quickly modify the controller model and immediately test the system. In addition, MiL testing is much less expensive than SiL or HiL testing.

In this paper, we focus on MiL testing of continuous controllers specified as mathematical models in Simulink. It is important to note that more than half of the controllers used in industry are continuous controllers [1]. Continuous controllers are used to capture controllers of low-level devices [18], e.g., controlling the velocity of a DC motor or the position of a flap. Modeling such controllers using more conventional software engineering notations such as state machines results in trivial discrete models where most important details are abstracted away, and hence, cannot be tested. In control theory, continuous controllers are specified using differential equations known as *proportional-integral-derivative (PID)* [25]. These equations include time-continuous variables as well as parameters that are not time-dependent and are fixed for every controller configuration. These parameters are referred to as *calibration* or *configuration* parameters, and optimize the behavior of a particular controller configuration for specific hardware.

Many existing approaches to testing embedded software systems focus on analyzing discrete or mixed discrete-continuous systems [15, 30, 14]. These techniques, however, are not amenable to analyzing controllers of low-level devices with a trivial discrete behavior. Some recent work has concentrated on using search-based algorithms to develop automated and systematic testing techniques for embedded software systems [16, 11, 20, 21]. Among these, our earlier work [20, 21] particularly focuses on testing *fixed* configurations of continuous controllers where only time-dependent variables of controllers are included in the test input data.

In this work, we present an approach for testing continuous controllers while accounting for their feasible configurations. Specifically, we develop a search-based technique to generate test cases, i.e., worst case scenarios, attempting to violate controller requirements. Our search strategy traverses individual points in the input search space of the controller, and is guided by an objective function defined based on the output of the controller simulation for

each point in the input space. The input space of the controller in our previous work [21] was made of time-dependent variables only. In this paper, we extend the search space to include configuration parameters as well as the time-dependent variables, allowing us to test controllers for different hardware configurations at the MiL level. However, the expanded search space becomes so large that our previous approach can no longer scale to identify worst case scenarios. That is, to handle multiple controller configurations, we cannot merely expand the input search space used in our earlier work. Instead, we have to build strategies to scale the search to large multi-dimensional spaces, and to reduce the cost of computing objective functions for individual points in the search space.

In this paper, we extend our previous work [20, 21] to support MiL testing of all feasible configurations of continuous controllers. Specifically, we use a combination of *dimensionality reduction* [6] and *surrogate modeling* techniques based on supervised learning [17, 26, 5, 10] to scale our search to large multi-dimensional spaces. Given an objective function, we first use dimensionality reduction techniques to identify the input variables that do not have a significant impact on the output of the objective functions, i.e., varying the values of those variables does not cause a significant change in the objective function output. We then apply an explorative random search [4] and focus the explorative search only on significant variables. Using the exploration results, we select some partitions of the input space that are more likely to include worst case input scenarios. We then apply a single-state search [19] to the selected partitions to identify worst case scenarios in each partition. Our objective functions require us to simulate Simulink models and are computationally expensive. Therefore, for each objective function and for each partition, we use the exploration results to build a surrogate model [17] based on supervised learning techniques [38]. The surrogate model is faster to compute than the objective function, and is able to predict its output within some confidence interval. Our single-state search uses the surrogate model to predict the output of the objective function when the decision as to which point the search should move to can be made based on the surrogate model.

We evaluated our approach by applying it to a complex, industrial controller, consisting of 443 Simulink blocks from the automotive domain. Our experiment showed that applying dimensionality reduction prior to exploration helps generate more accurate and predictive surrogate models for two out of three requirements. In addition, combining single-state search with surrogate modeling remarkably improves our approach for the same two requirements. Specifically, for one requirement, the search combined with surrogate modeling is eight times faster than the search without surrogate modeling, and for the other requirement, the search with surrogate modelling computes higher output values that could not be computed by the search without surrogate modeling. Finally, our approach identified critical violations of the controller requirements that had been found neither by our earlier work [21] nor by manual testing based on domain expertise.

2. BACKGROUND AND MOTIVATION

In this section, we discuss MiL testing of continuous controllers, and motivate our work based on the needs of the automotive domain. Figure 1(a) shows an overview of a controller and a plant (environment) model in a feedback loop. The system input and output are respectively shown as *desired* and *actual* in Figure 1(a). The variable *desired* represents the location we want a robot to move to, the speed we require an engine to reach, or the position we need a valve to arrive at. The variable *actual* represents the actual state/speed/position of the plant. The *actual* value is expected to

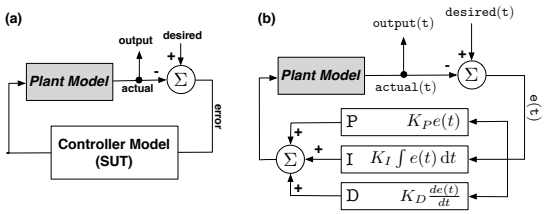

Figure 1: Continuous controllers: (a) A controller model and its environment (plant) model, and (b) a controller PID formulation [1].

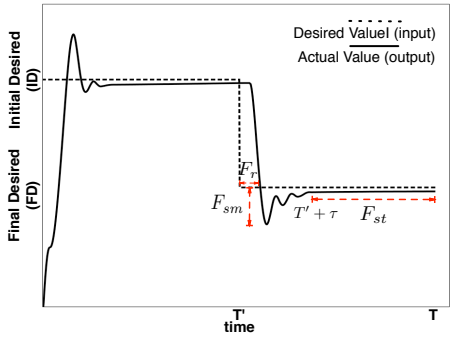

Figure 2: Continuous controller input and output. Objective functions F_{st}, F_{sm} and F_r are illustrated on the output signal.

reach the *desired* value over a certain time limit, making the *error*, i.e., the difference between the *actual* and *desired* values, eventually zero or practically negligible. The task of the controller is to eliminate the error by manipulating the plant to obtain the desired effect on the actual status of the plant. The variables *desired* and *actual* are time-dependent and are specified using signals over time.

Continuous controllers are designed via mathematical models known as *proportional-integral-derivative (PID)* equations [25]. Figure 1(b) shows the generic (most basic) formulation of a PID equation. Let $e(t)$ be the difference between $desired(t)$ and $actual(t)$. A PID equation is a summation of three terms: (1) a proportional term $K_P e(t)$, (2) an integral term $K_I \int_0^t e(t)dt$, and (3) a derivative term $K_D \frac{de(t)}{dt}$. The coefficients K_P, K_I and K_D are configuration parameters and are fixed for every instance of a controller. These parameters depend on the physical properties of the plant (hardware) that the controller eventually interacts with. The number of configuration parameters in real-world PID controllers is often more than three. This is because these PID controllers have more than three terms, or each term may have a more complex formulation consisting of more coefficients. For example, our industrial system includes six configuration parameters.

PID formulations are typically specified in Simulink, allowing engineers to simulate controllers, analyze their output, and eventually generate code from the controller design. Figure 2 shows simulations representing the input and output variables of a controller. The input, i.e., variable *desired*, is shown by a dashed line and is given as a step signal. Specifically, this input signal first sets the controller at an *initial desired (ID)* value until time T', and then requires the controller to move to a *final desired (FD)* value by time T. The output, i.e., variable *actual*, shown by a solid line, starts at zero, and gradually moves to reach and stabilize at the *initial desired* (ID), and then it moves towards the *final desired* (FD) and stabilizes there.

To test a controller, engineers simulate the controller using different input step signals by varying ID and FD. For each simulation, they generate the output signal, i.e., the *actual* signal in Figure 2,

and check if the output conforms to the following three main requirements that we identified in our previous work [21]:

Stability: The controller shall guarantee that the output will reach and stabilize at the input after a time limit.

Smoothness: The actual value shall not change abruptly when it is close to the input.

Responsiveness: The controller shall respond within a time limit.

We define three objective functions F_{st}, F_{sm} and F_r over the output signal to estimate quantitative values for stability, smoothness and responsiveness requirements, respectively. We provided formal definitions of these functions in our earlier work [20, 21]. Briefly, to evaluate stability, engineers check whether, after time τ, the difference between input and output converges to zero, and in addition, the output remains stable afterwards. Function F_{st} measures the maximum difference between input and output over the time periods shown by thin dashed arrows in Figure 2. To evaluate smoothness, engineers check whether the undershoot or overshoot of the output signal is not too large. As shown in Figure 2, F_{sm} measures the maximum undershoot and overshoot of the output signal. The response time is the time it takes for the controller output to reach or to become close to its input. As shown in Figure 2, F_r measures the response time intervals of the output.

To compute F_{st}, F_{sm} and F_r, we provide an input to the controller Simulink model, and use the generated output signal to compute these functions. The input to the controller includes values for the controller configuration parameters, and values for ID and FD, which characterize the input step signal. Therefore, functions F_{st}, F_{sm} and F_r depend on the configuration parameters, and ID and FD variables. Having computed these three functions over an output signal, engineers can then decide, based on their domain knowledge and thresholds provided in the requirements, whether the controller under analysis satisfies each of the above requirements or not. In general, the higher the objective function value, the more likely it is that the controller violates the requirement corresponding to that objective function.

Currently, in most companies, MiL testing of controllers is limited to running the controller for a small number of input signals that are often selected based on the engineers' domain knowledge and experience. Existing MiL testing often fails to find erroneous scenarios that the engineers are not aware of a priori. Identifying such scenarios later during SiL/HiL is much more difficult and expensive than during MiL testing.

In our earlier work, we proposed a search-based approach for testing an individual controller configuration [20, 21]. That is, we fixed the values for configuration parameters based on those used during HiL testing, and developed search algorithms maximizing F_{st}, F_{sm} and F_r within an input search space with two dimensions: ID and FD. For example, consider a controller with float variables ID and FD ranging from 0 to 1. The input search space of this controller with dimensions ID and FD is shown in Figure 3(a). Our approach has two steps: exploration and search. For each objective function, our approach first computes that function for several points randomly selected from the input space (exploration step). We divide the input space into a number of regions, e.g., 100 equal regions in Figure 3(a), and shade each region based on the objective function output. The resulting diagram is called a *heatmap* diagram. For example, Figure 3(a) is a heatmap diagram generated based on the stability objective function values for 1000 points. Specifically, in a heatmap diagram, the points in darker regions yield a higher, average objective function output than the points in lighter regions. Hence, darker regions are more likely to include input values violating the controller requirements. In the search

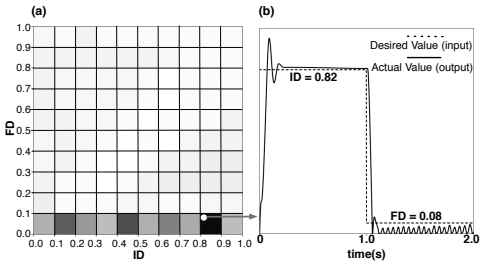

Figure 3: An example representing our MiL testing approach for a single controller configuration [21]: (a) A heatmap diagram generated by our approach for the stability requirement, and (b) the identified worst case scenario violating the stability requirement.

Controller Objective Functions	F_{st}, F_{sm}, F_r
Input Space (d=8)	$R_{ID} \times R_{FD} \times R_{Cal1} \times R_{Cal2} \times R_{Cal3} \times R_{Cal4} \times R_{Cal5} \times R_{Cal6}$
	$[0..1] \times [0..1] \times [3.5..4] \times [2..4] \times [0..0.13] \times [0.3..0.7] \times [0..0.05] \times [1..1.2]$

Figure 4: Controller objective functions and the ranges of the input variables and configuration parameters for our industrial controller.

step of our approach, we apply a single-state search algorithm to dark regions to find points that maximize our objective functions, and hence, are more likely to violate the requirements.

For example, the space shading in Figure 3(a) is produced based on the output of the stability objective function F_{st} applied to a faulty controller. The controller satisfies the stability requirement for the input signals generated by the points in the clear-shaded regions. However, applying our single-state search to the dark region of (ID = $[0.8..0.9]$ and FD=$[0..0.1]$) results in finding the simulation in Figure 3(b) which clearly violates the stability requirement. Note that since the controller behavior for most of the input space (more than 95% of the input space) conforms to the stability requirement, it is very unlikely that one can discover the faulty behavior by manually selecting and running a few simulations.

In this paper, we extend our MiL testing approach to include not only ID and FD variables, but also the controller configuration parameters. For example, the industrial controller used as a case study in this paper has six configuration parameters referred to as Cal1 to Cal6, respectively. The type of variables ID, FD, and Cal1 to Cal6 is float. We denote the range of variables ID and FD by R_{ID} and R_{FD} respectively, and the range of each Cali by R_{Cali}. Figure 4 provides value ranges for each of the configuration variables and ID and FD variables in our industrial controller. To compute highest values of F_{st}, F_{sm} and F_r for any controller configuration, our search algorithm has to handle the search space size of $|R_{ID} \times R_{FD} \times R_{Cal1} \times \ldots \times R_{Cal6}|$. Due to sheer size of the search space, we cannot effectively solve our problem by simply applying existing search algorithms. Instead, in this paper, we combine *dimensionality reduction* and *surrogate modeling* techniques (based on supervised learning) to perform search in large and multi-dimensional input spaces and to reduce the cost of computing our objective functions for individual points in the search space.

3. MIL TESTING USING SEARCH

Figure 5 shows an overview of our search-based approach to MiL testing of continuous controllers. Similar to our previous work [21], our approach is composed of an *exploration* and a *search* step. The input to our approach includes a set of objective functions, and a

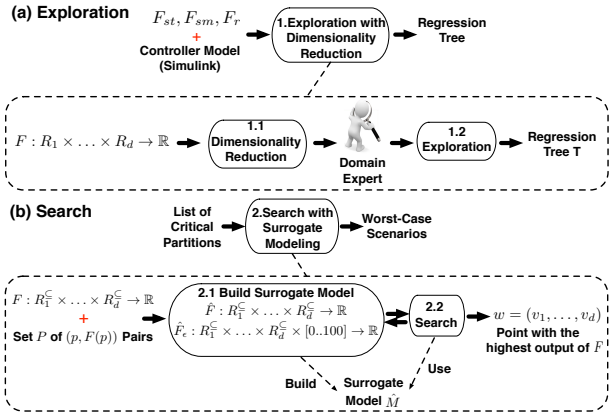

Figure 5: An overview of our automated approach to MiL testing of different configurations of continuous controllers: (a) Exploration step, (b) Search step.

controller Simulink model required to compute the objective functions. Specifically, in our work, objective functions are F_{st}, F_{sm} and F_r as described in Section 2. The input spaces of these functions are the same and equal to the cross product of the ranges for ID and FD variables and the configuration variables. For example, the input space (objective function domains) in our case study is $R_{ID} \times R_{FD} \times R_{Cal1} \times \ldots \times R_{Cal6}$ (See Figure 4). The range of the objective functions is the set of real numbers $\mathbb{R}$.

In the exploration step (Figure 5(a)), we apply a random (unguided) search to the entire input space of the objective functions, and then based on the results we build a *regression tree* [38] partitioning the input space such that the variance of the objective function values within each partition is minimized. Before performing exploration, for each objective function, we use a dimensionality reduction strategy to identify dimensions that have the most impact on that objective function. This allow us to focus the exploration step only on dimensions with most impact on the objective functions, and hence, increase the scalability of our approach. The regression tree built based on the exploration results enables us to divide the space into partitions such that, for each partition, the value of the objective function is predictable within a certain confidence interval. In addition, regression trees allow the engineers to visualize partitions from a multidimensional space. We then use regression trees to identify higher risk partitions, i.e., those partitions that contain input values that are likely to violate controller requirements. Regression trees replace the heatmap diagrams (e.g., Figure 3(a)) we used in our earlier work [20], which can no longer be used for a space that has more than three dimensions. From the regression trees, we select the partitions with the highest mean for the objective function, which are considered to be higher risk as they are more likely to contain critical errors.

In the search step (Figure 5(b)), we focus our search on the selected partitions and employ single-state search algorithms to identify, within those partitions, the worst case scenarios to test the controller. In this step, we build surrogate models to minimize the need for running simulations of Simulink controller models so as to make the search more scalable. Based on our previous experience, the main cause of computation time of our search is such simulations. Recall that to compute objective functions, we have to simulate the input controller model. In this work, for each objective function and for each input space partition, we create a surrogate model that predicts the objective function values within that partition. We use the exploration results related to each partition to build surrogate models. A surrogate model built for an objective function is able to predict the output of that function within a confidence interval. Using this model, our single-state search can determine whether the decision as to which point the search has to move to can be made without resorting to running simulations or not. In Sections 3.1 (Exploration) and 3.2 (Search), we describe the first and second steps of our approach, respectively.

3.1 Exploration

Figure 5(a) shows the exploration step of our approach. The input of this step is an objective function $F : R_1 \times \ldots \times R_d \rightarrow \mathbb{R}$. Function F can be any of the objective functions in Figure 4. The goal of this step is to efficiently select a set of points in the space of $R_1 \times \ldots \times R_d$ and compute the output of F for each point in this set. The output of this step is used to identify critical parts of the $R_1 \times \ldots \times R_d$ space (i.e., those partitions for which F produces the most critical (highest) values), and further, to create a surrogate model for individual critical partitions that can estimate as precisely as possible the output of F for any arbitrary point in that partition.

Given an objective function F, we first use dimensionality reduction techniques to identify search input dimensions that have the least impact on the output of F. A dimension i ($1 \leq i \leq d$) has a low impact on the output of F if varying the input vector $(v_1, \ldots, v_i, \ldots, v_d)$ of F by varying v_i within the range R_i does not yield a significant change in the output of F.

In addition, this step uses *adaptive random search* [19] to explore the input space of F by focusing on the dimensions with most impact on F. Here, we briefly discuss adaptive random search applied to the entire input space without considering dimensionality reduction. In Section 3.1.2, we show how this algorithm is modified to focus on significant search dimensions only. Adaptive random search is an extension of the naive random search that attempts to maximize the euclidean distance between the selected points. Adaptive random search explores the space by iteratively selecting points in areas of the space where fewer points have already been selected. Let $R_1 \times \ldots \times R_d$ be the input space, and let P_i be the set of points selected by adaptive random search at iteration i. At iteration $i+1$, adaptive random search randomly generates a set P of candidate points in the input space. The search computes distances between each candidate point p and points already selected in P_i. Formally, for each point $p = (v_1, \ldots, v_d)$ in P, the search computes a function $dist(p, P_i)$ as follows:

$$dist(p, P_i) = \text{MIN}_{(v'_1,\ldots,v'_d) \in P_i} \sqrt{\sum_{j=1}^{d} (v_j - v'_j)^2}$$

The search algorithm then adds to P_i a point p in P such that $dist(p, P_i)$ is the largest. The algorithm terminates after generating a specific number of points. Adaptive random search is similar to *quasi-random number generators* that are available in some languages, e.g., MATLAB [32]. Similar to our adaptive random search algorithm, these number generators attempt to generate points that are evenly distributed across the entire space. Below, we describe how we use dimensionality reduction and adaptive random search to efficiently select a set of points in the input space of F that can be utilized in the search step (Figure 5(b)) for building an effective surrogate model.

3.1.1 Dimensionality Reduction

Using dimensionality reduction, we identify the dimensions of the domain of F that have the most impact on the output of F. To do so, we rely on *sensitivity analysis* which is the study of how the variations in the outputs of a function are related to the variations in its inputs [6]. An application of sensitivity analysis is identifying input variables with the most and the least significant impact on a

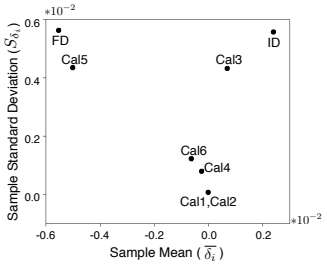

Figure 6: The elementary effect analysis results for the stability objective function (F_{st}) with an eight-dimension input space.

given function. Among different sensitivity analysis techniques, we use the *elementary effects* method [24]. This method is intuitive, and compared to other techniques such as variance-based methods [6], requires fewer number of function evaluations, and hence, is well-suited for functions that are expensive to compute.

The elementary effects method works as follows: Using adaptive random search, we generate r points in the input space of $R_1 \times \ldots \times R_d$. For each dimension i ($1 \leq i \leq d$), and for each point j, we vary v_i in the input vector $(v_1, \ldots, v_d)$ of F by a parameter Δ and measure the resulting variation δ_{ij} in the output of F. We then compute the sample mean $\overline{\delta_i}$, and the sample standard deviation S_{δ_i} for each input space dimension i to assess the impact of that dimension on F. Figure 6 shows an example output of the elementary effects method for the stability objective function F_{st} with an eight-dimension input space. Provided with this diagram, engineers choose the dimensions with significant impact on F_{st}. For example, they may decide that Cal1, Cal2, Cal4, and Cal6 (which have sample means and standard deviations close to zero) are not significant.

3.1.2 Exploration in the Reduced Dimensional Space

To explore the input space, we use the adaptive random search algorithm as described at the beginning of Section 3.1. The difference is that we modify the *dist* function to ensure that the search maximizes diversity along the dimensions with significant effect on F. Otherwise, note that exploration with and without dimensionality reduction take about the same time and operate in the same space. Let D_r be the set of dimensions with significant impact on F, e.g., D_r in Figure 6 is $\{\mathsf{ID}, \mathsf{FD}, \mathsf{Cal3}, \mathsf{Cal5}\}$. For each candidate point p, we compute $dist_{D_r}(p, P_i)$ as follows:

$$dist_{D_r}(p, P_i) = \text{MIN}_{(v'_1,\ldots,v'_d)\in P_i} \sqrt{\sum_{j\in D_r}(v_j - v'_j)^2}$$

In contrast to the *dist* function presented at the beginning of Section 3.1, in $dist_{D_r}$, we consider only the values related to the dimensions in D_r. That is, we focus on maximizing the diversity of the selected points along the dimensions in D_r, and the values of the variables along the dimensions in $\{1, \ldots, d\} \setminus D_r$ can either be fixed or set arbitrarily. This allows us to intensively explore parts of the space that results in the most variations in the output of F.

Having selected N points in the input space of F and having computed F for each point, we build a *regression tree* based on these points, e.g., see Figure 7. The regression tree represents a stepwise partition of the input space aimed at getting increasingly homogeneous partitions with respect to F. Such a representation is a convenient and intuitive way to visualize the impact of input space dimensions on F [38].

Figure 7 shows an example of the regression tree generated from the exploration results for F_{st} ($N = 1000$). Each node in the tree corresponds to a space partition and is labeled by the number of the points in that partition as well as the mean and standard deviation of the values of F_{st} for those points. For example,

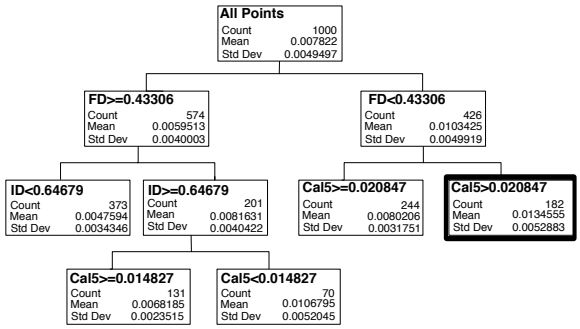

Figure 7: An example of a regression tree generated for F_{st}.

the highlighted node in Figure 7 corresponds to a partition where $min_{FD} \leq \mathsf{FD} < 0.43306$ and $0.020847 < \mathsf{Cal5} \leq max_{Cal5}$, and it includes 182 points selected during exploration. The mean and standard deviation of F_{st} for these points are 0.0134555 and 0.0052883, respectively.

We select a partition with the highest mean value for the objective function from the regression tree. Note that such a partition has to be a leaf node because any non-leaf node has exactly one child node whose mean value is higher than the mean value of its parent. The partition with the highest mean is more likely to include errors and critical scenarios. We denote input space partitions by $\mathcal{R}^{\subseteq}$, and define it as $\mathcal{R}^{\subseteq} = R_1^{\subseteq} \times \ldots \times R_d^{\subseteq}$ such that $R_i^{\subseteq} \subseteq R_i$ for $1 \leq i \leq d$. For example, in Figure 7, we select the highlighted partition that has the highest mean value, and denote it by

$\mathcal{R}^{\subseteq} = R_{ID}^{\subseteq} \times R_{FD}^{\subseteq} \times R_{Cal1}^{\subseteq} \times R_{Cal2}^{\subseteq} \times R_{Cal3}^{\subseteq} \times R_{Cal4}^{\subseteq} \times R_{Cal5}^{\subseteq} \times R_{Cal6}^{\subseteq}$ such that $R_{FD}^{\subseteq} = [min_{FD}..0.43306)$, $R_{Cal5}^{\subseteq} = (0.020847..max_{Cal5}]$, and $R_v^{\subseteq} = R_v = [min_v..max_v]$ for every other variable v. After selecting a partition $\mathcal{R}^{\subseteq}$, this partition together with the points generated during exploration inside this partition are passed to step 2.

3.2 Search

Figure 5(b) shows the search step of our approach. The search step takes as input a function F, a partition $\mathcal{R}^{\subseteq}$, and a set P of $(p, F(p))$ pairs computed during the exploration step. Specifically, the space $\mathcal{R}^{\subseteq}$ is a critical input space partition identified in the previous step, and P includes pairs of $(p, F(p))$ where p is a point in $\mathcal{R}^{\subseteq}$ that was selected by the adaptive random search in step **1.2**, and $F(p)$ is the output of F applied to p. We refer to P as an *observation set*. In step **2.1** in Figure 5(b), we first build a surrogate model $\hat{M}$ using *supervised learning* techniques [38]. The surrogate model $\hat{M}$ consists of two functions: A predictive function $\hat{F} : \mathcal{R}^{\subseteq} \rightarrow \mathbb{R}$ that estimates the output of F for any point in $\mathcal{R}^{\subseteq}$, and an error function $\hat{F}_\epsilon : \mathcal{R}^{\subseteq} \times [0..100] \rightarrow \mathbb{R}$. For each point $p \in \mathcal{R}^{\subseteq}$ and a given confidence level cl, $\hat{F}_\epsilon(p, cl)$ is the prediction error indicating that, with a confidence level of cl, the actual value of $F(p)$ is within $\hat{F}(p) \pm \hat{F}_\epsilon(p, cl)$.

In step **2.2**, we search the points in $\mathcal{R}^{\subseteq}$ to find a point that maximizes F. Since computing F is expensive, we expedite the search by checking whether we can conclusively decide the next point that the search should move to using $\hat{F}$. Specifically, the output of $\hat{F}$ is conclusive, if the prediction error $\hat{F}_\epsilon$ is less than the difference between the output of $\hat{F}$ and the existing highest value found by the search. Otherwise, we have to compute the actual output of F by simulating the Simulink controller related to F.

3.2.1 Surrogate Modeling

We use supervised learning techniques to build a surrogate model of the function F. Given a point p, supervised learning predicts

$F(p)$ using a set P of observations with known output values [38]. We divide the observation set P into a *training* set and a *test* set. The training set is used to infer a predictive function $\hat{F}$. This is done by estimating the parameters of $\hat{F}$ such that $\hat{F}$ fits the training data as well as possible, i.e., for the observations in the training set, the differences between the output of F and that of $\hat{F}$ are minimized. The test set is then used to evaluate the accuracy of the predictions produced by $\hat{F}$ when applied to observations outside the training set.

Supervised learning techniques are categorized into *regression* and *classification* techniques where the goal is to predict real-valued and categorical outputs, respectively. We use regression techniques because F is a real-valued function. Specifically, we use the following regression techniques:

Linear Regression (LR). Linear regression assumes that F is linear. Given an observation point $p = (v_1, \ldots, v_d)$, linear regression infers $\hat{F}$ as a linear function

$$\hat{F}(v_1, \ldots, v_d) = \beta_0 + \beta_1 v_1 + \beta_2 v_2 + \ldots + \beta_d v_d$$

Exponential Regression (ER). Exponential regression assumes that F is non-linear but monotonic, and infers $\hat{F}$ in the following form:

$$\hat{F}(v_1, \ldots, v_d) = \beta_0 v_1^{\beta_1} v_2^{\beta_2} \ldots v_d^{\beta_d}$$

Polynomial Regression (PR). Polynomial regression assumes that F is neither linear nor monotonic. The inferred function $\hat{F}$ takes the form of an nth-degree polynomial:

$$\hat{F}(v_1, \ldots, v_d) = \beta_0 + \sum_{i=1}^{d} \beta_{i1} v_i + \beta_{i2} v_i^2 + \ldots + \beta_{in} v_i^n$$

In this paper, we consider PR($n = 2, 3$) because based on our experiments the predictability of our surrogate models decreases for $n > 3$. In the above three regression methods, parameters β_i ($0 \leq i \leq d$) and β_{ij} ($1 \leq i \leq d$ and $1 \leq j \leq n$) have to be estimated using the training data. The goal is to estimate these parameters such that the sum squared error of the predicted outputs for the training points is minimized. That is, $\sum_{i=1}^{m} (F(p_i) - \hat{F}(p_i))^2$, where m is the number of observations in the training set, is minimized. In addition, we use *stepwise regression* to build $\hat{F}$ in the above three regression methods [38]. Instead of including all variables v_1 to v_d at once, stepwise regression aims at selecting a minimal subset of variables that are statistically significant at explaining the variation in F. The variables may be selected in a forward or backward way. In the forward selection, variables are iteratively selected and added as long as the sum squared error over the training data decreases, i.e., the predictive power of $\hat{F}$ improves. At each iteration, a statistical test (*F-test*) is used to determine which variable best improves the predictive power of $\hat{F}$ [7]. Dually in the backward elimination, variables are iteratively removed as long as the predictive power of $\hat{F}$ over training data does not decrease. Most implementations of stepwise regression, e.g., *stepwiselm* in MATLAB [34], combine the backward and forward methods by iteratively switching between them. That is, they add variables using forward selection for some iterations, and then switch to backward elimination after a while to remove unnecessary variables. We note that there are a number of other supervised learning methods such as *support vector regression* [31, 8] and neural networks [38], that we do not discuss in this paper due to lack of space, and leave them for future work.

In addition to function $\hat{F}$, all the above regression methods provide a function $\hat{F}_\epsilon : \mathcal{R}^{\subseteq} \times [0..100] \rightarrow \mathbb{R}$. Function $\hat{F}_\epsilon(p, cl)$ estimates the prediction error for a point p based on a given confidence level cl, which is a percentage value between 0 and 100, usually above 80%. For example the input $cl = 95$ implies that the actual value of $F(p)$ lies in the interval of $\hat{F}(p) \pm \hat{F}_\epsilon(p, cl)$ with a confidence level of 95%.

3.2.2 Single-State Search Using Surrogate Model

As discussed in our earlier work [21] to compute highest values of our objective functions, among the existing meta-heuristic search techniques, we opt for *single-state* algorithms as opposed to *population-based* ones. This is because population-based search algorithms compute fitness functions for a set of points (a population) at each iteration [19]. Hence, they are less likely to scale when objective functions are computationally expensive.

We propose a new Hill Climbing (HC) single-state search algorithm [19] extended to use surrogate models. Our algorithm speeds up the search by avoiding simulations when it is possible to decide the next move for search, based on the surrogate model predictions. The algorithm is shown in Figure 8. It takes as input the function F, the set P of observations in the partition $R_1^{\subseteq} \times \ldots \times R_d^{\subseteq}$, the surrogate model $\hat{M} = (\hat{F}, \hat{F}_\epsilon)$, and a confidence level cl . The output of the algorithm is a point w with the highest output of F found in T_s seconds.

The algorithm first identifies the observation $(p, F(p)) \in P$ such that $F(p)$ is the largest in P, and sets the variable $highest$ to $F(p)$ (lines 1, 2). At the beginning of each iteration of this algorithm, $highest$ is the highest output of F computed so far. The algorithm then iteratively generates a new point $newp$ by tweaking the current point p (line 4). The tweak operator is similar to the one used in our earlier work [21]. That is, we tweak a point $p = (v_1, \ldots, v_d)$ by shifting each v_i with a value x randomly selected from a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 0.1 \times |R_i^{\subseteq}|$. In our previous work, we showed that this tweak operator yields effective results for two dimension input space functions [21]

For each new point $newp$, the algorithm computes surrogate model functions $\hat{F}(newp)$ and $\hat{F}_\epsilon(newp, cl)$ (lines 5, 6). At line 7, the algorithm determines whether it needs to compute the actual value of $F(newp)$, or it can decide the next move only using $\hat{F}(newp)$. Figure 9 depicts the conditions under which we have to compute $F(newp)$. Specifically, if $highest$ is less than or equal to $\hat{F}(newp) - \hat{F}_\epsilon(newp, cl)$, or more than or equal to $\hat{F}(newp) + \hat{F}_\epsilon(newp, cl)$, with a confidence level of cl, $highest$ is less or greater than $F(newp)$, respectively. In the case of $highest$ greater than $\hat{F}(newp) + \hat{F}_\epsilon(newp, cl)$, the search does not move to $newp$, and hence, no need to compute $F(newp)$. In the case of $highest$ less than $\hat{F}(newp) - \hat{F}_\epsilon(newp, cl)$, the search may move to $newp$ depending on the value of $F(newp)$. Thus, we compute $F(newp)$. If $highest$ is between $\hat{F}(newp) - \hat{F}_\epsilon(newp, cl)$ and $\hat{F}(newp) + \hat{F}_\epsilon(newp, cl)$, we cannot confidently compare $highest$ with the actual value of $F(newp)$ using $\hat{F}(newp)$, and hence, have to compute $F(newp)$.

Line 7 in Figure 8 summarizes the condition for determining whether $F(newp)$ has to be computed or not. If yes, the algorithm computes $F(newp)$, and refines the surrogate model $\hat{M}$ using the new observation $(newp, F(newp))$ (lines 8–10). Otherwise, at line 11, the algorithm decides the next point that the search should move to. When it decides to move (lines 12, 13), it updates the current point p with $newp$, and $highest$ with the highest value of F computed so far and stored in y. In addition, it keeps a copy of $newp$. Finally, once the loop at line 3 terminates, the algorithm reports the point w with the highest output of F found in T_s seconds.

Algorithm. SINGLESTATESEARCH

Input: $F : R_1^{\subseteq} \times \ldots \times R_d^{\subseteq} \rightarrow \mathbb{R}$.
The set P of $(p, F(p))$ pairs.
The surrogate model $\hat{M} : (\hat{F}, \hat{F}_\epsilon)$.
A confidence level cl.
Output: Point $w = (v_1, \ldots, v_d)$ with the highest output of F.

1. Let $(p, f) \in P$ s.t. for all $(p', f') \in P$, we have $f \geq f'$
2. $highest = f$
3. **for** T_s seconds **do:**
4. $newp = Tweak(p)$
5. $y = \hat{F}(newp)$
6. $\epsilon = \hat{F}_\epsilon(newp, cl)$
7. **if** $highest < (y + \epsilon)$ **:** /*If $highest$ is less than $\hat{F}(newp) + \hat{F}_\epsilon(newp, cl)$, we
8. $y = F(newp)$ simulate and compute $F(newp)$, as shown in Figure 9.
9. $P = P \cup (newp, y)$ Otherwise, we bypass simulation.*/
10. $(\hat{F}, \hat{F}_\epsilon) = BuildSurrogateModel(P)$
11. **if** $y > highest$ **:**
12. $highest = y$
13. $p = w = newp$
14. **return** w

Figure 8: Single-state Hill Climbing (HC) search algorithm with surrogate modeling.

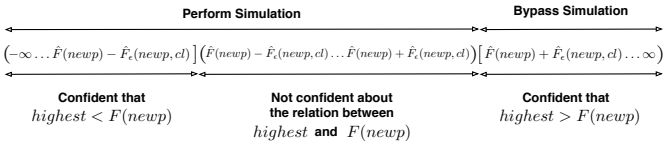

Figure 9: Depicting the conditions used by the algorithm in Figure 8 to perform or to bypass simulations.

Note that the higher the value of cl, the algorithm in Figure 8 is more likely to compute the actual value of F by running simulations. For $cl = 100$, the interval of $\hat{F}(newp) \pm \hat{F}_\epsilon(newp, cl)$ is equal to $(-\infty, +\infty)$, and hence, the algorithm behaves like a conventional Hill Climbing algorithm and runs a simulation at each iteration. For $cl = 0$, we have $\hat{F}_\epsilon(newp, cl) = 0$, and hence, the algorithm runs fewer simulation, i.e., only when $highest < \hat{F}(newp)$ (see Figure 9).

4. EXPERIMENT SETUP

In this section, we present the research questions, some information about our industrial subject, the metrics used to evaluate surrogate models, and information about our experiment design.

4.1 Research Questions

RQ1 How do the different surrogate modeling techniques perform compared to one another?

RQ2 Does dimensionality reduction improve prediction accuracy of the best surrogate modeling technique identified in **RQ1**?

RQ3 How do our single-state search algorithms, with and without surrogate modeling, perform compared to each other?

RQ4 Does our approach help identify testing results that are useful in practice?

In **RQ1**, for each objective function, we identify, among the four regression methods discussed in Section 3.2.1, the method that yields a surrogate model with highest prediction accuracy. For each objective function, we then use this best surrogate model for the single-state search. In **RQ2**, we determine whether focusing exploration on significant dimensions of each fitness function improves the prediction accuracy of the surrogate models. Recall that exploration with and without dimensionality reduction take about the same time and operate in the same space. However, the exploration results with dimensionality reduction are more diversely distributed along the dimensions with significant impact on the objective functions. The question is whether this gives rise to more accurate and predictive surrogate models? In **RQ3**, we compare the performance and results of our single state search algorithms with and without surrogate modeling, in order to determine whether our new approach scales better in large search spaces. Finally, in **RQ4**, we compare our best results, i.e., test cases with highest objective function values, with those obtained in our previous work [21] as well as the existing test cases used in practice.

4.2 Industrial Subject

Supercharger is an air compressor blowing into a turbo compressor to increase the air pressure supplied to the engine. The air pressure is controlled by a mechanical bypass flap: When the flap is completely open (resp. closed), the air pressure is minimum (resp. maximum). Our industrial subject is the **S***upercharger* **B***ypass Flap* **P***osition* **C***ontroller (SBPC)* which determines the position of the bypass flap to achieve a desired air pressure. SBPC is one of the most complex controllers among those dedicated to engine management. In SBPC, the *desired* and *actual* variables (see Figure 1) represent the desired and actual positions of the flap, respectively. The flap position is bounded within $[0..1]$ (0 for open and 1.0 for closed), i.e., R_{ID}=R_{FD}=$[0..1]$ in our experiments. The SBPC controller and plant models are both implemented in Simulink and include 443 blocks in total. SBPC has six configuration parameters Cal1 to Cal6 impacting the PID controller terms, and hence, the controlling behavior of SBPC. Figure 4 shows the ranges for the SBPC configuration parameters.

4.3 Surrogate Modeling Evaluation Metrics

To assess the goodness of fit and predictive power of the generated surrogate models, we use two well-known evaluation metrics for supervised learning methods: (1) *coefficient of determination* or R^2 [38] and (2) *Mean of Relative Prediction Error* or MRPE [27]. Specifically, R^2 measures the proportion of the total variance of F explained by $\hat{F}$ for the observations in the training set. In other words, R^2 is a measure of goodness of fit on the training set. The value of R^2 is always less than 1. The higher the value of R^2, the higher the goodness of fit of $\hat{F}$ predictions.

The *Mean of Relative Prediction Error (MRPE)* [27] measures the predictive power of $\hat{F}$ using the observations from the test set. Let k be the number of observations in the test set. We compute the MRPE as follows:

$$\frac{1}{k} \times \sum_{i=1}^{k} \left| \frac{F(p_i) - \hat{F}(p_i)}{F(p_i)} \right|$$

That is, MRPE measures the average of the relative prediction errors of $\hat{F}$ for the observations in the test set. The lower the value of MRPE, the higher the predictive power of $\hat{F}$. Note that a surrogate model $\hat{F}$ with high R^2 may not yield low MRPE values due to *overfitting*, i.e., when $\hat{F}$ is excessively tailored to the training observations, but does not generalize well to the test observations. Therefore, to compare different surrogate models, we have to take into account both R^2 and MRPE metrics.

In theory, both R^2 and MRPE can be computed on either the training set or the test set. When applied to the training set, they measure the goodness of fit, and when applied to the test set, they assess the prediction accuracy. However, it is more common to use R^2 to measure goodness of fit, to get an idea of how much variance remains unexplained in the data used to fit the model, and MRPE to address prediction accuracy as it is more readily interpretable to assess the applicability of a prediction model. Therefore, we chose to compute R^2 on the *training* set and MRPE on the *test* set.

4.4 Experiment Design and Analysis Strategy

We implemented the elementary effect analysis, adaptive random search, surrogate modeling techniques including LR, ER, and PR($n = 2, 3$), and our single-state search algorithm in MATLAB. Both adaptive random search and single-state search are required to call simulations of the SBPC Simulink model. The latter, in addition, may resort to surrogate modelling by calling and refining surrogate models. We ran each Simulink simulation for 2 sec ($\mathbf{T} = 2s$ in Figure 2) to give SBPC enough time to stabilize. We ran all the experiments on Amazon micro instance machines which is equal to two Amazon EC2 Compute Units. Each unit has a CPU capacity of a 1.0-1.2 GHz 2007 Xeon processor. Each 2-sec Simulink simulation of SBPC takes about 31 sec on the Amazon machine. While calling the surrogate models is negligible (less than 5ms) and rebuilding them takes about 2 sec on average.

To investigate **RQ1**-**RQ4**, we designed and performed the following experiments. Below, we use abbreviations DR and SM for dimensionality reduction and surrogate models, respectively.

EXP-I. To answer **RQ1** and **RQ2**, we computed the output of exploration once with and once without DR. To compute exploration results with DR, we applied the elementary effect analysis to our three objective functions with parameters $r = 20$ and $\Delta = 0.556$. Recall from Section 3.1 that r is the number of points, and Δ is the size of the modification applied to each dimension of each point. These values are selected based on the guidelines in [6]. To account for randomness we repeated the elementary effect analysis 10 times. The results across different repetitions were consistent, and out of the eight input space dimensions, four were significant for F_{st}, and three were significant for F_r and F_{sm}. We then applied our adaptive random search to each of these functions by focusing the exploration on their significant dimensions only. We let $N = 1000$, and executed our adaptive random search to generate 1000 points for F_{st}, and 1000 points for F_r and F_{sm}. Note that since F_r and F_{sm} have the same significant dimensions, we generated the same points for both functions, but kept two output values for each point.

To compute exploration results without DR, we let $N = 2000$, and generated 2000 points across the eight dimensions of the input space, but computed F_{st}, F_r, and F_{sm} separately for each point. Note that the total number of points generated during exploration with and without DR was the same and equal to 2000.

For each objective function, we built two regression trees: one from the exploration results with DR, and one from the exploration results without DR. Each regression tree node corresponds to an input space partition. Suppose that σ and μ respectively denote the standard deviation and mean values related to each partition. We expand each regression tree until $\frac{\sigma}{\mu}$ for every leaf node falls below 0.1. Expanding the trees further often results in leaf nodes containing very few observations and corresponding to very small space partitions. By expanding the tree, the variance of the objective function values (σ) in leaf node partitions decrease. In each tree, among all the leaf nodes with $\frac{\sigma}{\mu} < 0.1$, we select the one that has the highest μ for further search of worst case scenarios.

For the partitions with the highest μ, based on their observation points, we created four surrogate models (SMs) using LR, ER, and PR ($n = 2, 3$) techniques. To effectively apply surrogate modeling techniques, we want to have at least 200 points in the selected partitions. If lower, we generate additional points before building a SM. We then use these 200 points as training data to build SMs. For the test sets, we generate an additional 50 points using a naive random selection technique. That is, the test points are chosen the same way irrespective of the use of DR. This allows us to use the same test sets to facilitate the comparison of the SMs generated with and without DR.

In total, to obtain the exploration results with and without DR, we performed 24 experiments (3 objective functions, 4 surrogate modeling techniques, with/without DR). To account for randomness, we repeated each of the 24 experiments 10 times.

EXP-II. To answer **RQ3** and **RQ4**, we performed single state search, for a given input space partition, once with and once without using SMs. To compute the search results with SM, we first built a SM using the best technique identified in **RQ1**. Then, for each objective function, we applied our SM-based single state search algorithm in Figure 8 to a given input space partition. We ran this algorihtm for three confidence levels (cl): 80, 90, and 95. For each objective function and each confidence level, we ran the search for 3000 sec.

To compute the search results without SM, for each objective function, we applied a Hill Climbing (HC) algorithm similar to that used in our previous work [21] to a given input space partition.We let the search run for 3000s, i.e., the total search budget time for the single state search with and without SM was the same and equal to 3000s. We refer to our SM-based single state search algorithm in Figure 8 as HC-SM, and to our single-state search algorithm without SM as HC-NoSM.

In total, to obtain the search results with and without SM, we performed 9 experiments with SM (3 objective functions, 3 confidence levels), and 3 experiment corresponding to the 3 objective functions without SM. To account for randomness, we repeated each of the 12 experiments 30 times.

5. EXPERIMENT RESULTS

This section provides responses, based on our experiments, for research questions **RQ1** to **RQ4** described in Section 4.1.

RQ1. To answer **RQ1**, we use the SMs generated by the **EXP-I** experiments (Section 4.4). Since **EXP-I** includes 24 experiments, and each experiment was repeated 10 times, we obtain 24 different groups of SMs where each group consists of 10 different SMs. For all the SMs, we compute R^2 and MRPE values. Figure 10(a) shows the average R^2 and average MRPE values for 12 SM groups generated for F_{sm}, F_{st}, and F_r based on the exploration results with DR. As shown in Figure 10(a), the SMs built using PR ($n = 3$) have the best goodness of fit (highest R^2) and best predictive accuracy (lowest MRPE). The results for the other 12 SM groups generated from the exploration results without DR are consistent with those shown in Figure 10(a). Specifically, our results confirm that, compared to LR, ER and PR($n = 2$), PR ($n = 3$) generates the most accurate SMs for our objective functions.

In addition, the results in Figure 10(a) show that while the surrogate models generated by PR ($n = 3$) for F_{sm} and F_r have high goodness of fit and predictive power, this is not the case for F_{st}. Additionally, though due to space constraints this cannot be shown here, we observed that applying PR with $n > 3$ results in less accurate SMs, i.e., lower R^2 and higher MRPE.

RQ2. To answer **RQ2**, we focus on the best SMs in the **EXP-I** experiments, i.e., the SMs generated by PR($n = 3$). Figure 10(b) shows the MRPE distributions related to the best SMs generated from the exploration results both with and without DR, and for each of F_{sm}, F_r and F_{st}. To statistically compare the MRPE values, we performed the non-parametric pairwise Wilcoxon Pairs Signed Ranks test [7], and calculated the effect size using Cohen's d [9]. The level of significance (α) was set to 0.05, and, following standard practice, d was labeled small for $0.2 \leq d < 0.5$, medium for $0.5 \leq d < 0.8$, and high for $d \geq 0.8$ [9]. Testing differences in MRPE distributions shows that for F_{sm} and F_r, the SMs built

(a) Mean of R^2/MRPE values for different surrogate modeling techniques

	LR	ER	PR(n=2)	PR(n=3)
	R^2/**MRPE**	R^2/**MRPE**	R^2/**MRPE**	R^2/**MRPE**
F_{sm}	0.66/**0.0526**	0.44/**0.0791**	0.95/**0.0203**	0.98/**0.0129**
F_r	0.78/**0.0295**	0.49/**1.2281**	0.85/**0.0247**	0.85/**0.0245**
F_{st}	0.26/**0.2043**	0.22/**1.2519**	0.46/**0.1755**	0.54/**0.1671**

(b) Distribution of MRPE for three objective functions with and without DR

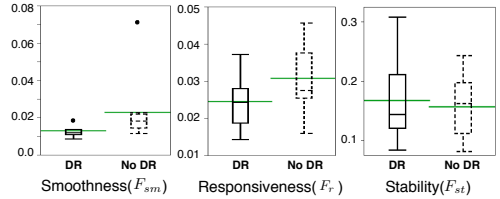

Figure 10: Experiment results for RQ1 and RQ2: (a) Comparing different surrogate modelling techniques, and (b) comparing exploration results with and without dimensionality reduction (DR).

with DR have significantly better predictive power than those built *without* DR. In addition, the effect size is "high" for both F_{sm} and F_r. For F_{st}, however, there is no statistically significant difference between the MRPE values of the SMs generated with and without DR. Specifically, focusing exploration on a reduced-dimension space significantly improves (with a high effect size) the predictive power of the SMs for F_{sm} and F_r, but does not have a significant impact on the predictive power of the SMs related to F_{st}. This may be due to the SMs for F_{st} being less accurate than those for F_{sm} and F_r. Since based on our results, DR never decreases the predictive power of the resulting SMs, our results suggest to focus exploration on the significant dimensions identified by DR.

RQ3. To answer **RQ3**, we use the **EXP-II** experiments (Section 4.4). Recall that in **EXP-II**, each run of each algorithm was executed for 3000 sec. In each run, we recorded the value of the variable *highest*, i.e., the highest found output of the objective function (see Figure 8), at every 100 sec time interval. Figure 11 compares the value distributions of *highest* obtained from HC-SM with cl = 80, 90, and 95, and from HC-NoSM at some selected (representative) time points for each of the objective functions F_{sm}, F_{st}, and F_r. Specifically, Figure 11(a) shows the value distributions of *highest* for F_{sm} at 800s, 1500s, 2500s, and 3000s. Figure 11(b) shows the value distributions of *highest* for F_r at 200s, 300s, and 3000s, and Figure 11(c) shows the value distributions of *highest* for F_{st} at 3000s. Time points, for each fitness function, were selected to make the overall trends visible for HC-SM and HC-NoSM.

Figures 12(a) and (b) respectively represent the differences between the mean values found for F_{sm} and F_r by each of the HC-SM algorithms and by HC-NoSM over 3000s of time. For F_{sm}, as shown in both Figures 11(a) and 12(a), within 3000s, all the HC-SM algorithms are able to find higher values compared to those found by HC-NoSM. Among the HC-SM algorithms, cl = 80 finds highest values for F_{sm} at 2500s with a mean of 23% and a median of 23.4%. HC-SM with cl = 90 and cl = 95 reach slightly lower values for F_{sm} at around 3000s. HC-NoSM, however, is not able to go higher than 22.3% (both mean and median) in the 3000s allotted time. That is, in 3000s and across 30 different runs, half of the values computed by HC-SM indicate an over/undershoot of 23.4% or higher, while the highest smoothness violation found by HC-NoSM is about 22.3%.

The one percent improvement of HC-SM over HC-NoSM is important in practice. This is because, depending on the hardware configuration, engineers specify a maximum over/undershoot (see Figure 2) that can be tolerated for the smoothness requirement. Ex-

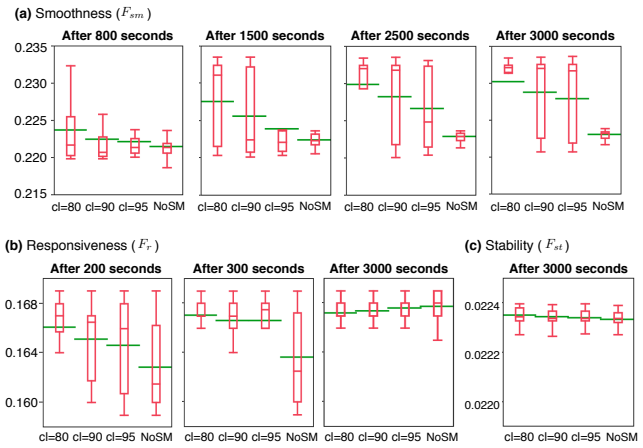

Figure 11: Boxplots for single-state search output values with and without surrogate modeling at some selected time points and applied to: (a) Smoothness (F_{sm}), (b) Responsiveness (F_r), and (c) Liveness (F_{st}).

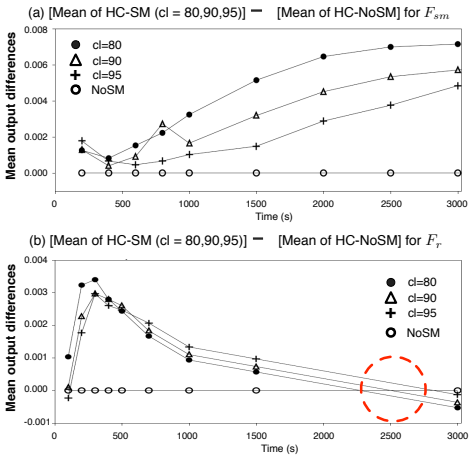

Figure 12: Differences of mean output values of search with surrogate modeling (HC-SM with cl = 80, 90, 95) and search without surrogate modelling (HC-NoSM).

ceeding this value, even slightly, is not in general acceptable. In the particular case of our case study (SBPC), slight deviation for the smoothness causes the flap to hit other hardware parts, generating noise and damaging hardware over time. We note that, even after running HC-NoSM for 5000s, its average and median output remained at around 22.3%. In general, the overall increase in the output of HC-NoSM over 5000s was very small.

For F_r, as shown in both Figures 11(b) and 12(b), the HC-SM algorithms find their highest values within the first 300s of time with cl = 80 being the fastest again. Specifically, at 300s, on average, HC-SM identifies a worst response time of 167s, while the average output of HC-NoSM indicates a response time of 163s. In contrast to the results of F_{sm}, for F_r, HC-NoSM is able to match the HC-SM algorithms in around 2500s of time (see the area shown by a dashed circle in Figure 12(b)). That is, the HC-SM algorithms are about 8 times faster than HC-NoSM. Finally, for F_{st} (Figure 11(c)), we did not observe any noticeable difference between the values found by the HC-SM algorithms and those found by HC-NoSM within 3000s of time. That is, within this time, all the algorithms behaved the same. This is due to the SM for F_{st}, which is clearly less accurate than those for F_{sm} and F_r. Hence,

	MiL-Testing different configurations	MiL-Testing fixed configurations	Manual MiL-Testing
Stability	2.2% deviation	-	-
Smoothness	24% over/undershoot	20% over/undershoot	5% over/undershoot
Responsiveness	170 ms response time	80 ms response time	50 ms response time

Figure 13: Comparing our MiL testing results with the results of MiL testing fixed controller configurations [21], and the results of manual MiL testing.

for F_{st}, the HC-SM algorithms almost run Simulink simulations at every search iteration, producing the same results as HC-NoSM.

We conclude that for accurate SMs with high predictive power (i.e., those built for F_{sm} and F_r), HC-SM outperforms HC-NoSM. Specifically, for F_{sm}, HC-SM computes higher output values that could not be computed by HC-NoSM, and for F_r, HC-SM is about eight times faster than HC-NoSM in finding the same output. This is because, compared to HC-NoSM, HC-SM runs fewer Simulink simulations, relying on the SM output in many search iterations. In addition, HC-SM with cl = 80 is slightly faster than HC-SM with cl = 90 and 95 because it runs fewer simulations.

RQ4. To demonstrate practical usefulness of our approach, we argue that MiL testing for *different* configurations of the SBPC controller finds requirements violations that have neither been identified via MiL testing of *fixed* configurations, nor by manual testing based on domain expertise.

Figure 13 compares our results with the results of our previous work on MiL testing with fixed configuration parameters [21, 20], and the results of manual expertise-based MiL testing. As shown in the figure, by extending our approach to test different configurations within some given ranges, we were able to identify a critical violation of the stability requirement with a deviation of 2.2%. Note that our previous results [21, 20] as well as the results from manual testing never indicated any stability error in SBPC. In addition, for smoothness and responsiveness, our current work found more critical violations: Specifically, the maximum observed over/undershoot was 24% compared to the 20% found by our previous work, and 5% found by manual testing. Finally, we computed a worst response time of $170ms$ compared to $80ms$ found by our previous work and $50ms$ identified via manual testing.

We conclude our results by noting that, due to limitations of manual testing, MiL testing in practice mostly focuses on a single controller configuration which is typically the one specified based on HiL configuration parameters. This obviously falls short when the controller is configured and deployed on a hardware with parameters that differ from those used on HiL. Since existing MiL testing does not consider different configurations, the errors that could have been found at MiL level go unnoticed until the very late development stages. Our work attempts to alleviate this shortcoming by enabling MiL testing for various configurations obtained by varying configuration parameters within their given ranges.

6. RELATED WORK

Several approaches to testing and analysis of Simulink models rely on formal methods [11], e.g., by analyzing hierarchical finite state machines [35], verifying logic control systems [22], and applying formal model checking [37]. These approaches are more amenable to verification of logical and state-based behaviors. In our work, we focused on testing pure continuous controllers which have not been previously captured by any discrete-event or mixed discrete-continuous notation [14, 2]. Further, our approach does not require any additional modelling since we apply our technique directly to Simulink models already developed as part of the controller development process in industry.

Search-based techniques have been applied to Simulink models to generate test input data with the goal of maximizing some coverage criteria [40, 12]. These criteria, however, are inadequate for testing continuous behaviours. Further, coverage criteria satisfaction alone is a poor indication of test suite effectiveness [29]. Our work enables generation of test cases specifically for controllers and based on their high-level requirements.

Continuous controllers have been widely studied in control engineering [25, 36], where the focus has been to optimize controller behaviors for a specific hardware [3]. Such optimization techniques are largely performed at late stages of development on real hardware and cannot replace our testing approach which is particularly useful for early design and development of controllers.

Surrogate modeling has been used to scale up computation in various application domains such as avionics [26], chemical systems [5], and medical domain [10]. Similar to our work, they use surrogate models in the context of evolutionary algorithms, and their goal is to approximate complex mathematical models with faster-to-compute models, i.e., surrogate models, to speed up highly time-consuming and costly simulations and experiments. Our work is the first to apply surrogate models for testing continuous controllers. Furthermore, our Hill Climbing (HC) single-state search algorithm represents a new combination of surrogate modelling and evolutionary algorithms by precisely showing when the search has to run real simulations and when the surrogate model is sufficient. Finally, our algorithm continuously refines surrogate models using the real simulations performed during the search.

In this work, we applied surrogate modeling in conjunction with dimensionality reduction to test controllers. *Dimensionality reduction* in our work differs from *input domain reduction* used in software code testing where the goal is to remove irrelevant variables prior to test case generation via static analysis [13, 23]. By dimensionality reduction, we mean identifying significant dimensions of each fitness function to focus exploration on those dimensions.

7. CONCLUSIONS

Testing and verification of the software embedded into cars is a major challenge in the automotive industry. The problem becomes more pressing when engineers have to consider different hardware configurations on which the software will be eventually deployed. In this paper we proposed an approach to MiL testing of continuous controllers in large configuration spaces, based on meta-heuristic search, with respect to smoothness, responsiveness, and stability requirements. The main challenge is to scale search to large multi-dimensional input spaces made up of possibly many configuration parameters. To scale search, we combined techniques for dimensionality reduction and supervised learning to build surrogate models that accurately predict simulation results without resorting to simulation in many cases. Our evaluation shows that our approach is able to identify critical violations of the controller requirements that had neither been found by our earlier work [20, 21] nor by manual testing. Further, we showed that combining search with surrogate modeling remarkably improves our approach for two out of three requirements. Specifically, for one requirement, the search combined with surrogate modeling is eight times faster than the search without surrogate modeling, and for the other requirement, the search with surrogate modelling computes more critical requirements violations than what could be detected by the search without surrogate modeling.

Acknowledgments

Supported by the Fonds National de la Recherche, Luxembourg (FNR/P10/03 - Verification and Validation Laboratory, and FNR 4878364), and Delphi Automotive Systems, Luxembourg.

8. REFERENCES

[1] Continuous Controllers. https://controls.engin.umich.edu/wiki/index.php/PIDTuningClassical, 2007. [Online; updated 20-May-2013; accessed 21-Mar-2014].

[2] R. Alur. Timed automata. In *CAV*, pages 8–22, 1999.

[3] M. Araki. PID control. *Control systems, robotics and automation*, 2:1–23, 2002.

[4] A. Arcuri and L. Briand. Adaptive random testing: An illusion of effectiveness? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 265–275. ACM, 2011.

[5] J. A. Caballero and I. E. Grossmann. An algorithm for the use of surrogate models in modular flowsheet optimization. *AIChE journal*, 54(10):2633–2650, 2008.

[6] F. Campolongo, J. Cariboni, and A. Saltelli. An effective screening design for sensitivity analysis of large models. *Environmental modelling & software*, 22(10):1509–1518, 2007.

[7] J. A. Capon. *Elementary Statistics for the Social Sciences: Study Guide*. Wadsworth Publishing Company, 1991.

[8] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[9] J. Cohen. *Statistical power analysis for the behavioral sciences (rev)*. Lawrence Erlbaum Associates, Inc, 1977.

[10] D. Douguet. e-LEA3D: a computational-aided drug design web server. *Nucleic acids research*, 38(suppl 2):W615–W621, 2010.

[11] F. Elberzhager, A. Rosbach, and T. Bauer. Analysis and testing of MATLAB/Simulink models: A systematic mapping study. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA'13)*, pages 29–34. ACM, 2013.

[12] K. Ghani, J. A. Clark, and Y. Zhan. Comparing algorithms for search-based test data generation of MATLAB/Simulink models. In *IEEE Congress on Evolutionary Computation, 2009. CEC'09.*, pages 2940–2947. IEEE, 2009.

[13] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener. The impact of input domain reduction on search-based test data generation. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 155–164. ACM, 2007.

[14] T. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.

[15] T. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM*, pages 1–15, 2006.

[16] M. Z. Iqbal, A. Arcuri, and L. Briand. Combining search-based and adaptive random testing strategies for environment model-based testing of real-time embedded systems. In *SBSE*, 2012.

[17] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[18] E. A. Lee and S. A. Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2011.

[19] S. Luke. *Essentials of Metaheuristics*. Lulu, 2009. http://cs.gmu.edu/~sean/book/metaheuristics/.

[20] R. Matinnejad, S. Nejati, L. Briand, T. Bruckmann, and C. Poull. Automated model-in-the-loop testing of continuous controllers using search. In *Search Based Software Engineering*, pages 141–157. Springer, 2013.

[21] R. Matinnejad, S. Nejati, L. Briand, T. Bruckmann, and C. Poull. Search-based automated testing of continuous controllers: Framework, tool support, and case studies. 35 pages. Accepted for publication at IST Journal, 2014.

[22] M. Mazzolini, A. Brusaferri, and E. Carpanzano. Model-checking based verification approach for advanced industrial automation solutions. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

[23] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener. Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation. *Software Engineering, IEEE Transactions on*, 38(2):453–477, 2012.

[24] M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[25] N. S. Nise. *Control Systems Engineering*. John-Wiely Sons, 4th edition, 2004.

[26] Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*, 41(4):687–696, 2003.

[27] H. Park and L. Stefanski. Relative-error prediction. *Statistics & probability letters*, 40(3):227–236, 1998.

[28] A. Pretschner, M. Broy, I. Krüger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *FOSE*, pages 55–71, 2007.

[29] M. Staats, G. Gay, M. Whalen, and M. Heimdahl. On the danger of coverage directed test case generation. In *Fundamental Approaches to Software Engineering*, pages 409–424. Springer, 2012.

[30] T. Stauner. Properties of hybrid systems-a computer science perspective. *Formal Methods in System Design*, 24(3):223–259, 2004.

[31] I. Steinwart and A. Christmann. *Support vector machines*. Springer, 2008.

[32] The MathWorks Inc. MATLAB quasi random numbers. http://www.mathworks.nl/help/stats/generating-quasi-random-numbers.html, 2003. [Online; accessed 17-Mar-2014].

[33] The MathWorks Inc. Simulink. http://www.mathworks.nl/products/simulink, 2003. [Online; accessed 25-Nov-2013].

[34] The MathWorks Inc. Stepwise Linear Regression Model. http://www.mathworks.nl/help/stats/stepwiselm.html, 2003. [Online; accessed 30-Mar-2014].

[35] I. Toyn and A. Galloway. Formal validation of hierarchical state machines against expectations. In *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, pages 181–190. IEEE, 2007.

[36] T. Wescott. PID without a PhD. *Embedded Systems Programming*, 13(11):1–7, 2000.

[37] M. Whalen, D. Cofer, S. Miller, B. H. Krogh, and W. Storm. Integration of formal analysis into a model-based software development process. In *Formal Methods for Industrial Critical Systems*, pages 68–84. Springer, 2008.

[38] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. Elsevier, 2011.

[39] J. Zander, I. Schieferdecker, and P. J. Mosterman. *Model-based testing for embedded systems*, volume 13. CRC Press, 2012.

[40] Y. Zhan and J. A. Clark. A search-based framework for automatic testing of MATLAB/Simulink models. *Journal of Systems and Software*, 81(2):262–285, 2008.