Automated Testing of Hybrid Simulink/Stateflow Controllers: Industrial Case Studies

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

ABSTRACT

We present the results of applying our approach for testing Simulink controllers to one public and one proprietary model, both industrial. Our approach combines explorative and exploitative search algorithms to visualize the controller behavior over its input space and to identify test scenarios in the controller input space that violate or are likely to violate the controller requirements. The engineers' feedback shows that our approach is easy to use in practice and gives them confidence about the behavior of their models.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**;

KEYWORDS

Automotive software systems, Matlab/Simulink, testing

ACM Reference format:
Reza Matinnejad, Shiva Nejati, Lionel C. Briand. 2017. Automated Testing of Hybrid Simulink/Stateflow Controllers: Industrial Case Studies. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 6 pages.
https://doi.org/10.1145/3106237.3117770

1 INTRODUCTION

The Simulink/Stateflow (SL/SF) environment is widely used for model-based design and development of control software systems in the Cyber Physical Systems (CPSs) domain [10]. Testing SL/SF models is important in practice and complicated by a number of factors that distinguish the testing of such models from the mainstream software testing applied to code: First, the inputs and outputs of SL/SF models are *signals*, i.e., variables capturing evolution of values over time. Second, SL/SF models have continuous-time behaviors with various signal shapes since they are expected to capture and continuously interact with the physical world. Finally, SL/SF models typically have a large number of inputs and configuration parameters with float data types.

Engineers require tools for automated testing of Simulink models. These tools would be useful in practice if they can effectively deal with the large input spaces of controllers. Specifically, they should be able (1) to visualize the behaviors of controllers over their input spaces, and (2) to find individual test scenarios that violate or are close to violating controller requirements. In our earlier work, we presented automated testing techniques and tools with mechanisms for input space *exploration* and *exploitation* [4, 6]. In this paper we apply our approach to two common types of controllers (i.e., open-loop and closed-loop controllers). In the exploration step, depending on the number of input variables, Heatmap diagrams or regression trees are used to visualize the results of input space exploration. Specifically, Heatmaps are used to visualize two-dimensional input spaces, and regression trees are used to visualize input spaces with more than two dimensions. Both Heatmaps and regression trees help engineers identify critical input space regions of the model under test. In the exploitation step, search algorithms [3] are used to find worst-case test scenarios of the model in critical regions of Heatmap diagrams and critical partitions of regression trees.

Our experience of applying our approach to one public and one proprietary model, both developed in industrial contexts, led to the following observations:

- Engineers want to identify worst-case scenarios of Simulink controllers to gain confidence about what risks can be expected in practice.
- Engineers are interested in exploring controller input spaces to identify conditions under which the risk of violating requirements is higher.
- The overhead of applying our approach is small since it is directly applicable to Simulink models without requiring any translation or pre-processing of the models under test.

2 MOTIVATING EXAMPLE

In this section, we describe our open-source case study – a realistic simulation model containing open-loop and closed-loop controllers. It is an experimental Electro-Mechanical Braking (EMB) system consisting of a physical (plant) model together with the software controller consisting of a discrete state machine (open-loop) and a continuous PID (closed-loop) controller. EMB is published by Bosch research lab as a representative benchmark model for cyber physical systems from the automotive domain [9] and is available at [8]. It is implemented in Matlab/Simulink [10]. A block-level view of the EMB Simulink model is shown in Figure 1. Below, we briefly discuss each block:

Physical model (EMB). The EMB system consists of a brake disk, a brake caliper and an electric DC-motor that moves the brake caliper towards the brake disk. The brake disk is connected to

We gratefully acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277).

ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3117770

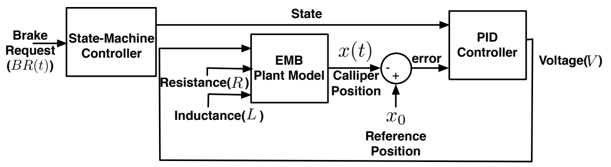

Figure 1: Simulink model for EMB.

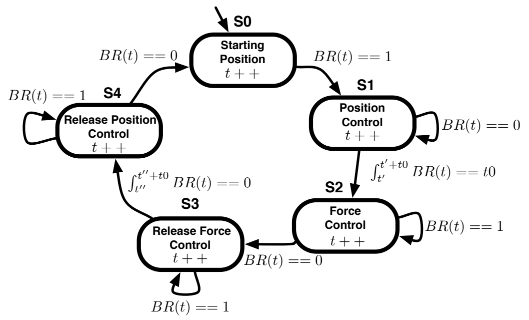

Figure 2: The EMB state machine controller.

the wheel. The contact between the caliper and the disk results in vehicle deceleration. Even after the contact between the disk and caliper has been established, the DC motor can be used to exert additional braking force, allowing for fine-grained control of the vehicle deceleration. In Figure 1, variable $x(t)$ represents the position of the caliper over time. The contact between disk and caliper occurs at the position x_0.

The EMB system is captured by the EMB Simulink plant model in Figure 1. This model consists of mathematical equations representing the relationships between the voltage applied to the motor by the controller, the motor current, the angular velocity, the angle of the motor shaft and the velocity and position of the brake caliper. The detailed equations are available in [8]. The voltage V applied to the motor by the PID controller as well as parameters R and L representing motor resistance and motor inductance, respectively. The parameters R and L are used to create different DC motor configurations corresponding to different DC motor hardware.

The EMB software controller is hybrid (mixed discrete-continuous) and consists of a state machine and a continuous controller (PID controller):

Discrete controller (state machine). The state machine controller in Figure 2 has five states: (1) The initial state (s_0) where the caliper is in its leftmost position. (2) The position control state (s_1) where a brake request is activated (i.e., $BR = 1$) so the controller moves the caliper from left to right. (3) The force control state (s_2) where the brake request activation persists for more than t_0. Then, an additional force is applied to push the caliper to the right harder. (4) The release force control state (s_3) where the brake request is deactivated (i.e., $BR = 0$), and the caliper is slowly moved back to the left. (5) The release position control state (s_4) where the brake request remains deactivated for more than t_0, and the caliper is released back to the initial position. Note that t'/t'' in Figure 2 is when the brake request is activated/deactivated in state s_1/s_3.

Continuous Controller (PID). In control theory, continuous controllers are specified using differential equations known as *proportional-integral-derivative (PID)* [7]. The EMB PID controller is used in states s_2 and s_3. The function of the EMB PID controller is to smoothly and properly control the physical movement of the caliper when the extra force is applied (state s_2) and released (state s_3). In particular, it moves the caliper position ($x(t)$) from the reference position (x_0) towards the disk and vice versa.

Configuration Parameters. Simulink/Stateflow models typically contain 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. For example, t_0 in Figure 2 is a configuration parameter that determines at what point in time extra force should be applied or released in states s_1 and s_3 after activation or deactivation of the brake request (*BR*), respectively. The value of t_0 depends, among others, on x_0, the size of the caliper and the amount of force that can be tolerated when pushing the caliper against the brake disk. For these parameters, engineers usually have some ranges as well as nominal values based on the hardware specifications. We consider the test input space to include both time-dependent input variables (i.e., input signals) and time-independent configuration parameters.

3 APPROACH

In this section we introduce the objective functions we use in our work to find the worst-case test scenarios of the Simulink models. We then provide an overview of our automated test generation approach, which consists of exploration and exploitation steps.

3.1 Specification

Controller input and output: Each input/output variable of a Simulink model is a signal, i.e., a function of time. When the model is simulated, its input/output signals are discretized and represented as vectors whose elements are indexed by time. Assuming that the simulation time is T, the simulation interval $[0..T]$ is divided into small equal time steps denoted by Δt. For example for EMB, we set $T = 1s$ and $\Delta t = 1ms$. We define a signal sg as a function $sg : \{0, \Delta t, 2 \cdot \Delta t, \ldots, k \cdot \Delta t\} \rightarrow \mathcal{R}_{sg}$, where Δt is the simulation time step, k is the number of observed simulation steps, and $\mathcal{R}_{sg}$ is the signal range. In our example, we have $k = 2000$.

Closed-loop controller. The function of closed-loop controllers is to control a system, often called the *plant*, such that its output follows a reference control signal, called the desired output. Closed-loop controllers are typically used when the control behavior of the plant is not precisely known and any disturbance can significantly impact the result of the control process.

In our earlier work, we proposed an approach to testing closed-loop controllers with respect to three *generic* requirements of such controllers, namely *stability*, *smoothness*, and *responsiveness* [4, 6]. Our approach, however, is applicable not just to generic properties but also to controllers function-specific requirements. For example, the following requirement is specified for EMB [9]:

- As soon as braking is requested, the contact between caliper and disk should occur within t_0 ms. The contact occurs when the distance between caliper ($x(t)$) and disk (x_0) is less than ϵ.

The above requirement (denoted by ϕ_1) can be formalized in terms of a temporal logic formula as follows [9]:

$\phi_1 = \Diamond_{[0,t_0]}\Box((x \leq x_0 + \epsilon) \wedge (x \geq x_0 - \epsilon))$

To apply our approach, we need to convert ϕ_1 into a quantitative (objective) function ideally in such a way that the notion of logical satisfaction is preserved and can be expressed as a predicate over the resulting quantitative function. We follow the translation of the temporal logic formulas into quantitative functions provided by [1] for this purpose. Below, we have shown the quantitative function obtained from ϕ_1 following the translation of [1]:

$$F(\phi_1) = Min\{Max\{Max\{|x(t') - (x_0 + \epsilon)|, |x(t') - (x_0 - \epsilon)|\}\}_{t \leq t' \leq T}\}_{0 \leq t \leq t_0}$$

where T is the simulation time. According to Abbas et.al. [1], we have the following: $F(\phi_1) > 2 \times \epsilon \Leftrightarrow \phi_1$ is violated

That is, EMB violates the property ϕ_1 if and only if there is some simulation output of EMB for which the function $F(\phi_1)$ yields a value larger than $2 \times \epsilon$.

Open-loop controllers: Open-loop controllers are another class of embedded controllers that control the plant in the absence of environment feedback. Open-loop controllers are typically used when the possible disturbances do not largely impact the control behavior or when it is too costly to implement the feedback mechanism. Figure 2 shows an example of an open-loop controller where the state of the caliper is computed based on the current state, time and the brake request and not based on the feedback from the plant. For open-loop controllers, there is no plant model and feedback is not available. For such controllers, we usually do not have requirements such as ϕ_1 since these requirements rely on the feedback from the plant. In our work, to test open-loop controllers, we look for anti-patterns in the controller outputs, i.e., patterns indicating a potential problem in the output signal.

In our earlier work, we identified two anti-patterns, namely instability and discontinuity [5]. In this paper, we introduce an additional pattern referred to as *growth to infinity*. Figures 3(a) to (c) show our three anti-patterns. These anti-patterns were identified by our review of the control engineering literature and the help of automotive engineers from our industry partners.

Instability. The first anti-pattern is instability where the controller output signal shows quick and frequent oscillations. Presence of instability anti-pattern in Simulink model outputs may have undesirable impact on physical processes or objects that are controlled by or interact with a Simulink model. An example of this behavior is shown in Figure 3(a). Given an output signal sg_o, we define the function $instability(sg_o)$ as the sum of the differences of signal values for consecutive simulation steps:

$$instability(sg_o) = \sum_{i=1}^{k} |sg_o(i \cdot \Delta t) - sg_o((i-1) \cdot \Delta t)|$$

Specifically, function $instability(sg_o)$ provides a quantitative approximation of the degree of instability of sg_o. The higher the value of the *instability* function for a signal sg_o, it is more likely that sg_o exhibits an instability failure.

Discontinuity. The second anti-pattern is discontinuity where the controller output signal shows a very short duration pulse in the controller output at point A in Figure 3(b). The control output of a Simulink model is a continuous function with some discrete jumps at state transitions. As a result, control signals, at each simulation step, are expected to be either left-continuous or right-continuous, or both. We define a heuristic objective function to identify signals that are neither left-continuous nor right-continuous at some simulation step. Since in our work simulation time steps (Δt) are not infinitesimal, we cannot compute derivatives for signals, and instead, we rely on discrete change rates that approximate derivatives when time differences of observable

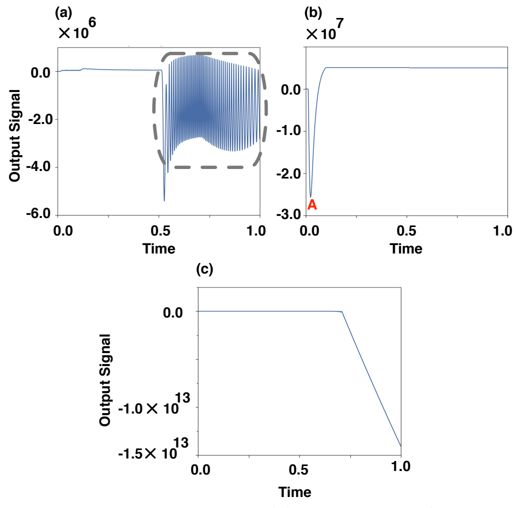

Figure 3: Three anti-patterns: (a) instability, (b) discontinuity and (c) growth to infinity.

changes cannot be arbitrarily small. Given an output signal sg_o, let $lc_i = \frac{|sg_o(i \cdot \Delta t) - sg_o((i-dt) \cdot \Delta t)|}{\Delta t}$ be the left change rate at step i, and let $rc_i = \frac{|sg_o((i+dt) \cdot \Delta t) - sg_o(i \cdot \Delta t)|}{\Delta t}$ be the right change rate at step i. We define the function $discontinuity(sg_o)$ as the maximum of the minimum of the left and the right change rates at each simulation step over all the observed simulation steps:

$$discontinuity(sg_o) = \max_{dt=1}^{3}(\max_{i=dt}^{k-dt}(\min(lc_i, rc_i))))$$

Specifically, we first choose a value for dt indicating the maximum expected time duration of a spike. Then for a fixed dt, for every step i such that $dt \leq i \leq k - dt$, we take the minimum of the left change rate and the right change rate at step i. Since we expect the signal to be either left-continuous or right-continuous, at least one of the right or left change rates should be a small value. We then compute the maximum of all the minimum right or left change rates for all the simulation steps to find a simulation step with the highest discontinuity from both left and right sides. Finally, we obtain the maximum value across the time intervals up to length dt. For our work, we pick dt to be between 1 and 3.

Growth to infinity. The third anti-pattern is growth to infinity where the controller output signal grows to an infinite value. An example of this behavior can be seen in Figure 3(c). Given an output signal sg_o, we define the function $infinity(sg_o)$ as the maximum of the signal values among all the simulation steps:

$$infinity(sg_o) = \max_{i=1}^{k} |sg_o(i \cdot \Delta t)|$$

Specifically, function $infinity(sg_o)$ provides a quantitative approximation of how likely the output signal sg_o is to grow to *infinity*. The higher the value of the infinity function for a signal sg_o, it is more likely that sg_o is growing to infinity.

3.2 Exploration and Exploitation

No matter which type of controller we are testing, we are able to obtain quantitative objective functions either based on the system specific or generic requirements (e.g., $F(\phi_1)$) or based on the three anti-patterns discussed earlier. These functions are defined such that optimizing them indicates violation of the system behavior or may indicate presence of error. Our approach to test controllers based on a given objective function has two consecutive steps: exploration and exploitation.

In the exploration step, we apply a random (unguided) search to the entire input space of the objective functions. Whether testing closed-loop or open-loop controllers, to visualize the results of the exploration step in terms of the given objective functions we use *Heatmap diagrams* [2] and *Regression trees* [11]. Both Heatmaps and Regression trees visualize the degree of compliance with a requirement or a pattern and identify critical partitions of the input space. Heatmap diagrams are graphical 2-D or 3-D representations of data where a matrix of values are represented by colors. They can be used when we have an input space consisting of two or three dimensions. Figure 4(a) shows an example of a Heatmap diagram generated for EMB model and based on property ϕ_1. Regression trees partition the input space such that the variance of the objective function values within each partition is minimized. Unlike Heatmap diagrams, regression trees can be used with input spaces consisting of any number of dimensions. Figure 4(b) shows an example of a regression tree generated for property ϕ_1 which corresponds to the Heatmap diagram in Figure 4(a). Each node in the tree corresponds to a space partition in the Heatmap and is labeled by the number of the points in that partition as well as the mean and standard deviation of the values of $F(\phi_1)$ for those points. For example, the highlighted node in Figure 4(b) corresponds to a partition where $0.5454 \leq R < 0.5725$ and includes 37 simulated points. The mean and standard deviation of $F(\phi_1)$ for these points are 0.0066 and 0.0004528, respectively.

In the exploitation step, we use meta-heuristic search to explore the critical partitions of the input space selected after the exploration and generate test cases maximizing the likelihood of presence of failures in controller outputs (i.e., test cases that produce outputs that break or are close to breaking closed-loop controller requirements or produce open-loop controller outputs containing anti-patterns). This can be reached by minimizing or maximizing the objective functions defined in Section 3.1. Depending on the output of the exploration step, the user selects some of the critical regions of the Heatmap or critical partitions of the regression tree to further search for increasingly worse test scenarios. For example Figures 3(a) to (c) show three examples of the worst-case test scenarios the search could find for the EMB controller with respect to instability, discontinuity and growth to infinity.

4 RESULTS

In this paper, we applied our test generation approach to two Simulink/Stateflow models. The first case study is the EMB model described in Section 2 [8]. The second case study is an industrial Simulink model from an automotive partner company and cannot be made public. We refer to the latter as the M model.

As explained in Section 3.2, during the exploration step of our approach we create a Heatmap diagram or a regression tree to visualize the controller behavior with respect to the objective functions, e.g., $F(\phi_1)$. Then, during the exploitation step, we run a single-state search algorithm to find the worst-case test scenarios of the controller with respect to a given property, e.g., ϕ_1, by minimizing or maximizing the corresponding objective function.

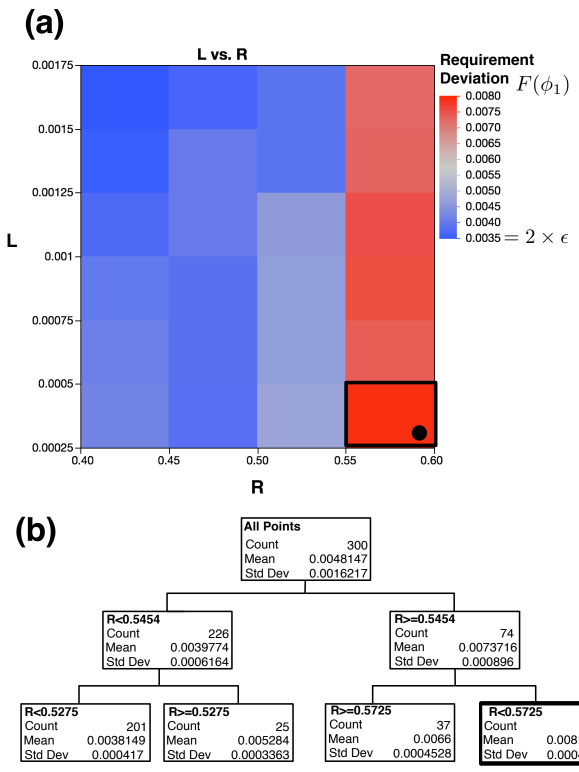

Figure 4: The results of applying our approach to test the EMB model against property ϕ_1: (a) The Heatmap diagram and (b) the regression tree

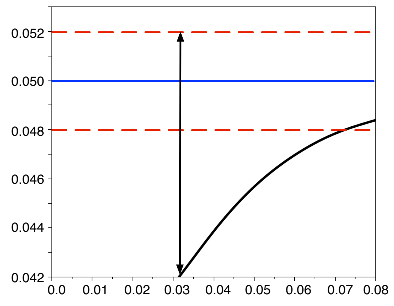

Figure 5: The worst-case scenario of the EMB model computed by our approach approach and violating property ϕ_1.

In our work, to test the EMB model's behavior, we ran two experiments. In the first experiment, we verify the correctness of the EMB model with respect to property ϕ_1 by applying our approach for closed-loop controllers. To complement the first experiment, in the second experiment, we ignore the feedback received from the plant and apply our approach for open-loop controllers to the EMB model to find anti-patterns in the output signals of the models.

In the first experiment, we fix the $BR(t)$ signal to the one used in [9] and vary the values of R and L within the ranges $[0.4, 0.6]$ and $[0.00025, 0.00175]$, respectively. These ranges are identified by model designers [9]. In general, the ranges for model inputs and configuration parameters should be identified by engineers who design and implement the models. In the first step of our

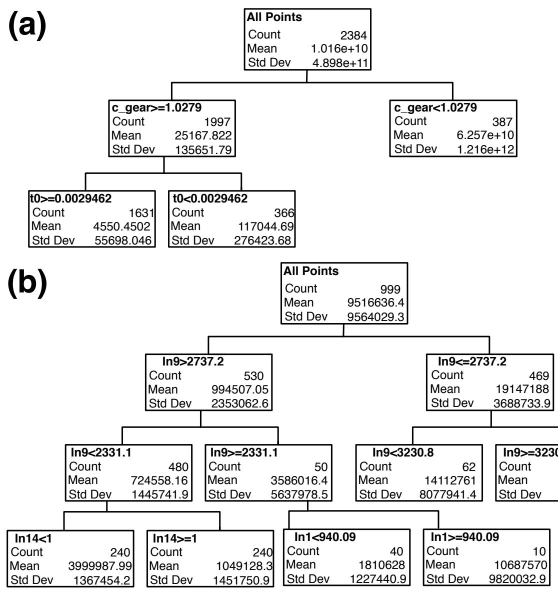

Figure 6: Example regression trees: (a) Growth-to-infinity for EMB model, and (b) Instability for M model.

approach, we use random exploration to generate test scenarios within the input space of the controller. Each point in the input space (identified by the two dimensions R and L in Figure 4(a)) corresponds to one test scenario of the controller. For each test scenario, we compute the value of $F(\phi_1)$ based on the simulation output. To compute the values of $F(\phi_1)$, similar to what is suggested by Strathmann et al. in [9], we set $\epsilon = 0.002$. Figure 4(a) shows the Heatmap diagram generated for ϕ_1. The color of each Heatmap region depicts the average value of the objective function $F(\phi_1)$ for the points in that region. Red regions have higher average values of the objective function, and hence, are more likely to contain test scenarios that violate ϕ_1. Recall from Section 3.1 that ϕ_1 holds for test scenarios with $F(\phi_1)$ less than or equal to $2 \times \epsilon$ and does not hold otherwise. One interesting observation from the Heatmap diagram in Figure 4(a) is that larger R values result in test scenarios that are likely to violate ϕ_1. Furthermore, it can be seen that varying L only slightly impacts compliance with ϕ_1, while varying R has a much more significant impact on whether ϕ_1 holds on the Simulink model or not.

In the second step of our approach, we applied a Hill-Climbing search algorithm (HC) [3] to the Heatmap region with the highest average value of $F(\phi_1)$ to find the worst-case test scenarios of the controller. We chose Hill-Climbing because based on our experience of applying search algorithms to continuous controllers [5], Hill-Climbing performs the best for critical Heatmap regions surrounded by other critical regions, such as the highlighted region in Figure 4(a). We note that the engineers may be interested to see more than one test scenario by searching multiple regions of the Heatmap, but here we show only the worst-case scenario found in the most critical region of the Heatmap. Figure 5 shows the worst-case scenario of the controller that was found with respect to property ϕ_1. This test scenario matches values $R = 0.59807$ and $L = 0.000297$, and is identified by a black circle on the Heatmap in Figure 4(a). In Figure 5, black and blue lines show $x(t)$ and x_0, respectively. The dashed red lines depict the two boundaries (i.e., $x_0 + \epsilon$ and $x_0 - \epsilon$) specified by property ϕ_1. Further, the solid arrow in Figure 5 indicates the value (0.01) of the objective function $F(\phi_1)$ for the test scenario, which violates property ϕ_1 (larger than $2 \times \epsilon$).

Let A and B be the highest objective function values computed during the exploration step and by the HC algorithm, respectively. We compute the relative improvement that the search step could bring about over the results of the exploration step as $\frac{B-A}{A}$. For the EMB case study, B was 0.01 (see Figure 5) and A was 0.0089. As a result, the relative improvement was about 12%, which is significant compared to our previous results [5]. Finally, it took around 75min and 15min to generate the Heatmap diagram in Figure 4(a) and the worst-case test scenario in Figure 5, respectively.

In the second experiment, we vary the values of R and L within the same ranges as the first experiment. In addition, we vary the values of three configuration parameters of the EMB model, namely t_0, c_gear and d_rot. Parameter t_0 controls how long we stay in states s_1 and s_3 in Figure 2 and was explained in Section 2. c_gear and d_rot are two configuration parameters of the EMB plant model. In our work, we vary the values of t_0 within $[0.001, 0.1]$, c_gear within $[1, 32]$ and d_rot within $[0.001, 0.1]$. Since the input space has five dimensions, we generated regression trees for the instability, discontinuity, and growth to infinity properties. Figure 6(a) shows the regression tree generated for the growth to infinity pattern. Among different partitions identified by the regression tree in Figure 6(a), the *infinity* function has the highest mean value when c_gear is less than 1.0279. The two other regression trees are similar to the regression tree in Figure 6(a). Specifically, c_gear and t_0 are the variables that appear in the first and second levels of the regression trees for both instability and discontinuity functions. The results of the exploitation step are shown in Figure 3. Specifically, Figures 3(a) to (c) show the worst-case test scenarios for instability, discontinuity, and growth to infinity, respectively. We note that the relative improvement of the results of the search step over the results of the exploration step were around 5%, 6% and 24% for instability, discontinuity and growth to infinity, respectively. In this experiment it took around 9 hours to generate the regression tree in Figure 4(a) as well as the other two regression trees and 1 hour to compute each worst-case test scenario in Figures 3(a) to (c).

As mentioned earlier, in addition to the EMB model, we applied our test generation approach for open-loop controllers to an industrial case study (M Model) in the automotive domain. The model is used to implement the vehicle automatic transmission controller and has 24 input variables. We generated three regression trees for instability, discontinuity and growth to infinity properties and ran the search to find the worst-case test scenarios within the partitions with the highest mean of the objective function for each property. Figure 6(b) shows the regression tree generated for instability. As shown in the figure, among the 24 inputs of the model, inputs one, nine, and 14 appear in the tree and are therefore more significant factors than other inputs. For the 24 input variables, we assume value ranges given to us by engineers with domain expertise. According to the regression tree, the instability function has the highest mean value in the partition where input nine is greater than 3230.8. The other two regression trees for discontinuity and growth to infinity are quite similar to the regression tree in Figure 6(b) and input variables five, nine and 14 appear in different levels of those trees. Note that the regression tree in Figure 6(b) provides interesting insights for engineers on where and under what conditions the model generates critically unstable behavior. Figures 7(a) and (b) show the worst-case test scenarios we found for discontinuity and growth to infinity, respectively. The worst-case test scenarios in Figures 7(a) and (b) yield values 11000 and 7221400 for the discontinuity and growth-to-infinity objective functions, respectively. The relative improvements of the result of the search step over the exploration step were around 10% and 8% for discontinuity and growth to infinity, respectively. Finally, we note that it took around 8 hours and a half to generate the three regression trees and 1 hour to compute each worst-case test scenario.

Lessons Learned: We can draw four conclusions based on our experiments results and the feedback we received from engineers:

- Most existing tools and techniques on testing and verification of Simulink models focus on identifying individual scenarios revealing some failure behavior. The exploration step of our approach provides engineers with conditions on the input variables under which failures are likely to occur. Specifically, the Heatmaps and regression trees help engineers specify conditions on the input variables specifying input space regions that contain most failure scenarios, and moreover, they show the level of risks associated with each critical input region. For example, as mentioned earlier, one interesting observation from the Heatmap diagram and regression tree in Figure 4 is that larger R values lead to test scenarios that are likely to violate ϕ_1. Furthermore, it can be seen that varying L only slightly impacts ϕ_1, while varying R is a much more significant factor. In a similar way, but this time based on a regression tree, the engineers can see the conditions under which the model has the most unstable behaviors in Figure 6(b). This, we were told, helps engineers find the root causes of failures.

- Our approach is further useful when engineers need to gain more confidence regarding the worst-case behavior of their models. Specifically, the output of the exploitation step generates the worst-case test scenarios of the controller under test. Such scenarios are characterized by the values of objective functions, which can then be used to assess how critical and risky such situations are in practice. For example, a value of 0.01 for $F(\phi 1)$, which is the worst-case test scenario in Figure 5, entails that the highest distance of $x(t)$ from the acceptable range is $0.01 - 2 \times \epsilon$.

- The overhead of applying our approach is small. Most of the existing techniques need Simulink models to be manually or automatically translated into an intermediary formalism, e.g., hybrid automaton, before the tool can be applied [9]. Manual translation of models into intermediary formalisms is time-consuming and infeasible for large models. Moreover, as models become more complex, they often contain features and blocks that cannot be translated into an intermediary formalism. Since our approach is black-box, it can be directly applied to any model without requiring any translation or pre-processing.

- Finally, the execution time of our approach is acceptable for real industrial Simulink models used in practice. Specifically, our test generation approach is a *simulation-based* approach where the execution time is determined by the number of model simulations and the time it takes to run each single simulation. The time to run a single simulation depends on the size and complexity of the model. As for the number of simulations, we have two different situations for exploration and exploitation steps. In our experiments, we ran the exploration step between one and nine hours, depending on the number of input space dimensions. We observed that if we run more simulations for the exploration step, more variables are likely to appear in the regression tree. As for the exploitation step, we ran the algorithm for 100 iterations. We picked 100 iterations since, based on our previous experiments [5], after 100 iterations the search often reaches a fitness plateau. In our experiments, running the model for 100 iterations took from several minutes to one hour.

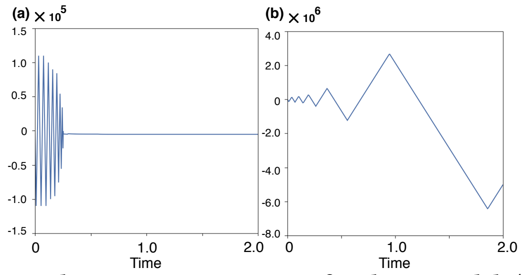

Figure 7: The worst-case scenarios for the M model: (a) discontinuity and (b) growth to infinity.

5 CONCLUSION

We presented the results of applying our approach for testing open-loop and closed-loop controllers to one public and one proprietary Simulink model, respectively developed by Bosch and another undisclosed partner. Our approach enables users to explore and visualize the controller input search space and identify worst-case test scenarios of the controller within the critical operating regions of the controller. Our experience shows that our approach is efficient and easy to use in practice, has a small overhead as it is directly applicable to Simulink models, and helps engineers gain confidence in the behavior of their Simulink models.

REFERENCES

[1] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. 2013. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)* 12, 2s (2013), 95.
[2] G. Grinstein, M. Trutschl, and U. Cvek. 2001. High-Dimensional Visualizations. In *7th Workshop on Data Mining Conference KDD Workshop.* 7–19.
[3] S. Luke. 2013. *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.
[4] R. Matinnejad, S. Nejati, L. Briand, and T. Brcukmann. 2014. MiL testing of highly configurable continuous controllers: scalable search using surrogate models. In *Proceedings of the 29th ACM/IEEE ASE.* ACM, 163–174.
[5] R. Matinnejad, S. Nejati, L. Briand, and T. Bruckmann. 2015. Effective Test Suites for Mixed Discrete-Continuous Stateflow Controllers. In *Proceedings of the 10th ACM SigSoft ESEC/FSE.* 84–95.
[6] R. Matinnejad, S. Nejati, L. Briand, T. Bruckmann, and C. Poull. 2015. Search-based automated testing of continuous controllers: Framework, tool support, and case studies. *IST Journal* 57 (2015), 705–722.
[7] N. S. Nise. 2004. *Control Systems Engineering* (4th ed.). John-Wiely Sons.
[8] T. Strathmann and J. Oehlerking. 2015. Electro-Mechanical Braking (EMB) system. http://cps-vo.org/node/20289. (2015).
[9] T. Strathmann and J. Oehlerking. 2015. Verifying Properties of an Electro-Mechanical Braking System. In *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems.* 49–56.
[10] The MathWorks Inc. 2017. Simulink. http://www.mathworks.nl/products/simulink. (2017).
[11] I. Witten, E. Frank, and M. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques.* Elsevier.