Performance-Driven Metamorphic Testing of Cyber-Physical Systems

Jon Ayerdi, Pablo Valle, Sergio Segura, *Member, IEEE*, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel

***Abstract*—*Cyber-physical systems* (CPSs) are a new generation of systems, which integrate software with physical processes. The increasing complexity of these systems, combined with the uncertainty in their interactions with the physical world, makes the definition of effective test oracles especially challenging, facing the well-known *test oracle problem*. Metamorphic testing has shown great potential to alleviate the test oracle problem by exploiting the relations among the inputs and outputs of different executions of the system, so-called *metamorphic relations* (MRs). In this article, we propose an MR pattern called PV for the identification of performance-driven MRs, and we show its applicability in two CPSs from different domains, which are automated navigation systems and elevator control systems. For the evaluation, we assessed the effectiveness of this approach for detecting failures in an open-source simulation-based autonomous navigation system, as well as in an industrial case study from the elevation domain. We derive concrete MRs based on the PV pattern for both case studies, and we evaluate their effectiveness with seeded faults. Results show that the approach is effective at detecting over 88% of the seeded faults, while keeping the ratio of FPs at 4% or lower.**

***Index Terms*—Autonomous systems, cyber-physical systems (CPSs), metamorphic relation (MR), MR pattern (MRP), metamorphic testing (MT), oracle problem.**

I. Introduction

CYBER-PHYSICAL systems (CPSs) are complex systems, which integrate computational and physical processes, and are often composed by multiple interconnected components [8], [31]. The applications of these systems extend to many domains, such as aerospace, automotive, healthcare, manufacturing, and consumer appliances [29]. Most of these applications require the system to be resilient to failures while operating in uncertain environments [63] (e.g., unmanned vehicles) and many of them also have strict safety requirements [8] (e.g., medical implants).

Considering the safety and robustness requirements for many of these systems, verification is one of the major concerns when it comes to their development [8]. However, given their high complexity and the inherent uncertainty of their interactions with the physical environment, automatically determining the expected output of these systems is not feasible in many cases [28]. For instance, self-driving cars suffer from this problem due to the sheer complexity of determining whether their behavior—typically driven by artificial intelligent (AI) algorithms—is correct or not. Also, these types of systems are extremely hard to test due to the uncertainty of the possible situations that can occur, such as extreme weather conditions or unexpected obstacles. This difficulty in predicting the correct output for a given input and then comparing it with the observed output is known as the *test oracle problem*, and it is recognized as one of the fundamental problems of software testing [10], [59].

There are some alternatives to specifying test oracles for an automated verification process. For instance, pseudo-oracles consist in independently developing multiple versions of the SUT and comparing the outputs in order to find discrepancies [20]. This approach, however, has a very high cost for complex systems, which might make it impractical in many cases. On the other hand, regression testing consists in comparing different versions of the SUT in order to detect breaking changes [62]. While this approach is applicable to most systems, there are many types of failures that cannot be detected with it, for instance, failures that are revealed under new conditions in which the SUT had never been deployed before. A common solution to compensate for the shortcomings of automated test oracles is to employ human oracles (i.e., manual testing), which is costly and error-prone.

Metamorphic testing (MT) adopts an alternative approach to traditional testing in order to alleviate the oracle problem; instead of verifying the correctness of each individual execution of the program under test, MT exploits known input and output relations that should hold among multiple executions, so-called metamorphic relations (MRs) [15]. For example, the following is an MR for the domain of self-driving cars [55]; "the car should behave similarly when traversing the same route under different (nonextreme) weather conditions". MT has been used in many domains, such as machine learning applications, web services,

. This work was supported in part by European Union's Hori-zon 2020 Research and Innovation Programme under Grant 871319, in part by European Commission (FEDER) and Junta de Andalucia through project APOLO under Grant US-1264651 and through project EKIPMENT-PLUS under Grant P18-FR-2895, and in part by Spanish Government (FEDER/Ministerio de Ciencia e Innovación Agencia Estatal de Investigación) through project HORATIO under Grant RTI2018-101204-B-C21. The work of J. Ayerdi, A. Arrieta, and G. Sagardui was supported by the Department of Education, Universities and Research of the Basque Country, and they were part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22). Associate Editor: T. H. Tse. *(Corresponding author: Jon Ayerdi.)*

Jon Ayerdi, Pablo Valle, Aitor Arrieta, and Goiuria Sagardui are with Mondragon Unibertsitatea, 20500 Arrasate, Mondragon, Spain (e-mail: jayerdi@mondragon.edu; hazibek02@mondragon.edu; aarrieta@mondragon.edu; gsagardui@mondragon.edu).

Sergio Segura is with Universidad de Sevilla, 41012 Sevilla, Spain (e-mail: sergiosegura@us.es).

Maite Arratibel is with Orona, 20120 Hernani, Spain (e-mail: marratibel@orona-group.com).

computer graphics, and compilers [17], [46]. This technique has also been successfully applied in the domain of CPSs, such as for testing wireless sensor networks [14], autonomous drones [33], self-driving cars [55], [66], or elevator installations [5].

MRs can often be defined at an abstract level, representing not a single relation, but a set of MRs. Inspired by this idea, the concept of MR patterns has been exploited by different authors [45], [47], [65]. Zhou et al. [65] defined an *MR pattern (MRP)* as an abstraction that characterizes a set of (possibly infinitely many) MRs. MRPs have proved to be very helpful on guiding testers on the search for MRs with a certain structure, making the identification of the relations significantly easier than when starting from scratch. For instance, the following is an MRP for self-driven cars: "the car should behave similarly when performing harmless alterations to the driving scenario". Instances of these pattern could include MRs as the one presented above, where the same route is traversed under different weather conditions, but also others as traversing the same route with different obstacles outside of the driving area [66]. Wu et al. [60] generalized this idea further proposing the *noise* MRP, which states that a reliable system should be able to perform its functions when a low level of interference (noise) is present.

Since its introduction in 1998, most research on MT has focused on functional testing [17], [46]. However, in recent years, some authors have outlined the potential of defining MRs not in terms of the expected impact in functionality, but in terms of the expected impact in nonfunctional properties, such as execution time, memory consumption, or energy usage [13], [49], [51].

In this article, we present an MRP called *performance variation (PV)* for the identification of failures in CPSs. Specifically, the pattern encourages testers to identify changes in the input of the CPSs that should have a predictable impact in its observed performance. For example, adding obstacles in the route of a self-driving car will typically result in more battery consumption. Violations of these MRs can uncover both functional (e.g., nonoptimal route calculation) and nonfunctional bugs (e.g., defective hardware component). To show the applicability of the pattern, we used it to identify MRs in two different types of CPSs: 1) autonomous navigation systems; and 2) elevator control systems. For the evaluation, we assessed the fault detection capability of the identified MRs in an open-source autonomous navigation system and industrial elevation system. Results show that the MRs—derived from the PV pattern—are effective in identifying over 88% of the faults, while keeping the ratio of false positives (FPs) at 4% or lower.

This article extends a previous paper by Ayerdi et al. [5] on the use of quality of service attributes and MT for detecting bugs in an industrial elevation system. Specifically, this work is based on the observation that the proposed MRs can be generalized as a pattern, being applicable to identify failures in potentially any software system and CPSs in particular. Hence, the main contributions of our work with respect to our previous paper lies in the introduction of a novel MRP (i.e., PV) and extensive empirical results, including a new case study, showing the potential of the MRs derived from the PV pattern for uncovering failures in CPSs.

In summary, after presenting the background on CPS (see Section II-A) and MT (see Section II-B), this article presents the following contributions.

1) A novel MRP—*PV*—exploiting the predictable impact in performance of input changes for the detection of failures in CPSs and an overview of potential applications (see Section III).
2) An empirical evaluation studying the effectiveness of MRs derived from the PV pattern to uncover faults in an industrial elevation case study, extending our previous work [5] (see Section IV-B).
3) An empirical evaluation studying the effectiveness of MRs derived from the PV pattern to uncover faults in an open-source autonomous vehicle modeled in MATLAB/Simulink [37] (see Section IV-C).
4) A publicly available replication package containing the source code for the autonomous driving system experiment was discussed in [7]. The results from the industrial elevation system cannot be published due to confidentiality concerns.

We discuss threats to validity and related work in Sections V and VI, respectively. Finally, Section VII concludes this article.

II. Background

In this section, we introduce the basics on CPSs and MT.

A. Cyber-Physical Systems

CPSs are a combination of computation and physical processes that interact with each other in complex ways. These systems are heterogeneous and contain different abstractions for physical and computational elements and their interactions [29]. An example of a CPS is a brake control system for a car, which requires the tight integration of physical calculations (to model the state of the vehicle and predict the effects of the actions from the controller) and computations (the control logic). The car as a whole can also be considered a CPS, which comprises many interconnected subsystems, such as the brake controller, and the obstacle detection systems.

Compared to software applications, testing CPSs presents additional challenges. On the one hand, CPSs tend to be highly complex heterogeneous systems, which contain both continuous and discrete components [8]. A model-based design is the most common paradigm for CPS development, and it is typically performed in modeling and simulation environments, such as MATLAB/Simulink [36] or OpenModelica [42], since they allow the tight integration of physical elements (e.g., motor mechanics simulation) with discrete logic that might be translated to software (e.g., controller design via state machines) [25], [53]. Testing of a CPS is usually performed on these modeling environments first (so-called model-in-the-loop testing), and later when the actual software is generated; tests can also be run with the real software and simulated hardware (software-in-the-loop testing). On the other hand, CPSs operate in uncertain environments where unexpected scenarios may happen [63]. Although model and simulation-based testing can be used to verify some of the behaviors of the system, it is not possible

to verify the behavior of the system under real conditions until testing is performed on the real hardware (hardware-in-the-loop testing). This kind of testing is even more costly than using simulations, but recent research suggests that the majority of the bugs can be reproduced and identified in simulation, reducing the total cost of the verification process [56].

B. Metamorphic Testing

MT [15], [48] aims to detect bugs by looking at the relations among the inputs and outputs of two or more executions of the program under test, so called *MRs*. For example, consider the program spellcheck(T) that searches for spelling errors in an English text file T. Checking if the output of the program is correct for nontrivial input text file would be difficult; this is an instance of the oracle problem. Suppose that we create a new text file T' by adding an independent text fragment S at the end of T: $T' = T + \{S\}$. Intuitively, the spelling errors found in T' should include those errors found in T. This can be expressed as the following MR: spellcheck(T) $\subseteq$ spellcheck(T'), where $T' = T + \{S\}$. In this relation, (T) is the *source test case* and (T')—created by extending the input text file, which is the *follow-up test case*. This MR can be instantiated into one or more *metamorphic tests* by using specific input values and checking whether the relation holds. If the relation is violated, the metamorphic test is said to have failed, indicating that the program under test contains a bug. Successful applications of MT have been reported in multiple domains, including web services and applications, machine learning, compilers, cybersecurity, and bioinformatics, among others [17], [46]. Industrial applications of MT have been reported at Google [21] and Facebook [2].

MRs can often be defined at a very abstract level, representing not a single relation, but a set of relations. When this happens, relations are referred to as MRPs [45], [47], [65]. Zhou et al. [65] defined an *MRP* as an abstraction that characterizes a set of (possibly infinitely many) MRs. MRPs have proved to be very helpful on guiding testers on the identification of MRs. As an example, Zhou et al. [65] proposed a *symmetry* MRP, based on the observation that most systems can be observed from different viewpoints from which the system appear the same. For example, an AI-enabled object recognition system should recognize the same objects in a video, regardless of whether it is played forwards or backwards. Segura et al. [50] proposed several MRPs for query-based systems, such as adding new conjunctive conditions (i.e., filters) for a search and expecting the results to be a subset of the original search.

Patterns are often defined as incomplete MRs where only the relation among the inputs or the outputs is specified. These are referred to as *MR input patterns (MRIPs)* [65] and *MR output patterns (MROPs)* [47], respectively. For example, Zhou et al. [65] proposed the "change direction" MRIP, representing those MRs where the follow-up test cases are created by changing the *direction* of the inputs, either physical or logical, or explicit or implicit. For example, the abovementioned MR, where an AI-enabled object recognition systems are executed twice running the input video forward and backward, is an instance of this pattern. Analogously, Segura et al. [47] proposed, among others, the "subset" MROP, which represents those MRs where the follow-up output should be a subset of the source output. Patterns can be defined hierarchically with some patterns being instances of more general ones. In this article, we propose an MRP and several MRIPs derived from it.

Most of the works on MT have focused on the detection of functional faults [17], [46]. Recently, Segura et al. [49], [51] proposed the concept of *performance MT*, where MRs are defined in terms of how the performance of the program under test (e.g., execution time) is expected to change when making certain changes in the programs' inputs. For example, intuitively, the execution time observed when searching for spelling errors in a text should increase, or at least remain the same, if the size of the text increases. This can be expressed as the following (performance) MR: T(spellcheck(T) $\leq T$(spellcheck($T + \{S\}$)), where S is a random nonempty text string. Research on performance MT is thriving with new applications emerging in domains, such as code generators [13] and data analytic platforms [26].

III. PV Pattern

In this section, we propose a novel MRP, defined as follows.

A. Performance Variation

This pattern represents those MRs that involve a change in the source input that has a predictable effect on the performance of the test case execution.

The intuitive idea behind this pattern is that it is typically straightforward to think in a change in the system's inputs such that it should have an expected impact in its performance. For example, if one or more obstacles are placed in the way of an autonomous vehicle, the time and the energy required to reach its destination should be higher than when performing the same route without obstacles, assuming similar external conditions (e.g., weather, traffic, etc.). If they are not, we could be certain that the system is faulty. Note that PVs could reveal not only nonfunctional bugs, but also functional ones. For instance, in the previous example, a violation of the MR could be caused by an energy leak (nonfunctional) or a bug in the navigation system (functional).

A key characteristic of the performance variation (PV) pattern is that it is extremely generic, being potentially applicable to the identification of MRs in most systems. However, there are certain characteristics of CPSs that make them especially suitable as a target domain. First, many performance metrics are directly related to requirements on CPSs, such as execution time on real-time systems, which makes monitoring this type of property during testing crucial. Second, even if the performance metrics are not directly part of the requirements, these types of systems are often resource-constrained in many aspects, such as tight processing capabilities, low memory, and limited power sources (e.g., batteries), which makes performance bugs much more likely to escalate into severe failures. Finally, the interactions that CPSs have with the physical environment make some aspects of the state of these systems uncertain, and performance metrics may be one of the few ways to detect and diagnose invalid or undesirable physical states (e.g., ground vehicle traction loss).

Performance measurements are inherently nondeterministic; they can vary among executions due to numerous factors, such as the system workload or the hardware settings. This means that it is usually not possible to perform a direct comparison between the performance metrics (e.g., execution time) observed in two or more executions of the SUTs. This is also the case with heuristic programs, where the system may return different responses for the same inputs, leading to PVs among executions. To address this issue, several approaches have been proposed, such as using tolerance thresholds [40] or comparing statistical distributions obtained from running the program multiple times [23]. In what follows, when we refer to a performance measurement being *lower* ($\lesssim$), *higher* ($\gtrsim$), or *similar* ($\simeq$) than another, we assume that some of the previous methods might be used. In the following section, we explain how the PV pattern can be used to identify MRs in two different types of CPSs. It is common that a certain change in the inputs can have an expected impact in different performance metrics, such as execution time, memory consumption, and energy usage. To reflect this, we present several MRIPs derived from the more general pattern PV. Each MRIP represents groups of MRs sharing the same input relation. Then, for each MRIP, we mention at least one MR that can be instantiated using a specific performance metric.

It is worth noting that the MRs presented in the following sections are intentionally simple for illustrative purposes. Later, in the evaluation section, we show that how MRs can get more complex in practice.

B. Elevator Control System

Passenger elevator control systems must respond to vertical transportation requests by coordinating one or more elevators so that all the requests are fulfilled as efficiently as possible. The efficiency of these systems can be measured by one or more objectives, including total execution time for a set of requests, average waiting time (AWT) for the passengers, or energy consumption, among others.

In order to describe MRs, we define the operation serve(E, P, C) for elevator installations, where E is a list of integers indicating the floors where the elevators are positioned initially, P are the various elevator parameters (motor start delay, acceleration, maximum speed, etc.), and C is a set of passenger calls $c \in C$, each of which will be encoded as (c_t, c_s, c_d), representing an arrival time (c_t), a source floor (c_s), and a destination floor (c_d). Fig. 1(a) shows an example scenario of a six-story building with two elevators in floors 4 and 5, which we encode as $E = \{4, 5\}$. For simplicity, we assume that the elevator parameters P apply to all the elevators equally, i.e., all the elevators are identical. We will also omit the parameters in P that are not relevant, so the syntax $\{\text{speed} = 1, \ldots\}$ indicates that the value of the speed parameter is 1, whereas the rest of the parameters are irrelevant/unchanged.

In what follows, we describe some MRIPs derived from the PV pattern and some sample MRs derived from them.

1) MRIP1: Additional Calls: This pattern represents MRs where the follow-up test cases are constructed by adding one or more passenger calls to the source input. When this happens, the performance of the system is expected to be worse or at least the same, since the elevator(s) must perform extra tasks. For example, if we add an extra call to the test case, the total distance (TD) traversed by the elevators should increase or remain the same, since the elevators need to attend to one additional passenger. This can be expressed as the following MR:

$$\text{TD}(\text{serve}(E, P, C_s)) \lesssim \text{TD}(\text{serve}(E, P, C_f)) \quad (1)$$

where $C_f = C_s \cup c$.

For instance, suppose a source test case consisting of the initial elevator positions from Fig. 1(a), $E = \{4, 5\}$, and the set of passenger calls $C_s = \{(1, 2, 3)\}$, representing a single call at $t = 1$ from floors 2 to 3. Suppose that a follow-up test case is created by adding a new call at $t = 2$ from floors 5 to 3: $C_f = \{(1, 2, 3), (2, 5, 3)\}$. Then, the TD should increase or remain the same as

$$\begin{aligned}&\text{TD}\left(\text{serve}(\{4, 5\}, \{\cdots\}, \{(1, 2, 3)\})\right)\\ &\quad\lesssim \text{TD}\left(\text{serve}(\{4, 5\}, \{\cdots\}, \{(1, 2, 3), (2, 5, 3)\})\right). \quad (2)\end{aligned}$$

Analogous MRs can be derived using other performance metrics, such as the passenger waiting time or the number of elevators' movements (see Section IV-B).

2) MRIP2: Additional Elevators: This pattern groups the relations where the follow-up test case is generated by adding new elevators to the source input scenario. When this happens, the overall performance of the system from the user perspective should be better, since the elevator control system has more resources available to attend the passenger calls. For example, the following MR is an instance of this pattern, where adding one or more elevator is expected to decrease the AWTs of passengers

$$\text{AWT}(\text{serve}(E_s, P, C)) \gtrsim \text{AWT}(\text{serve}(E_f, P, C)) \quad (3)$$

where $E_f \supset E_s$.

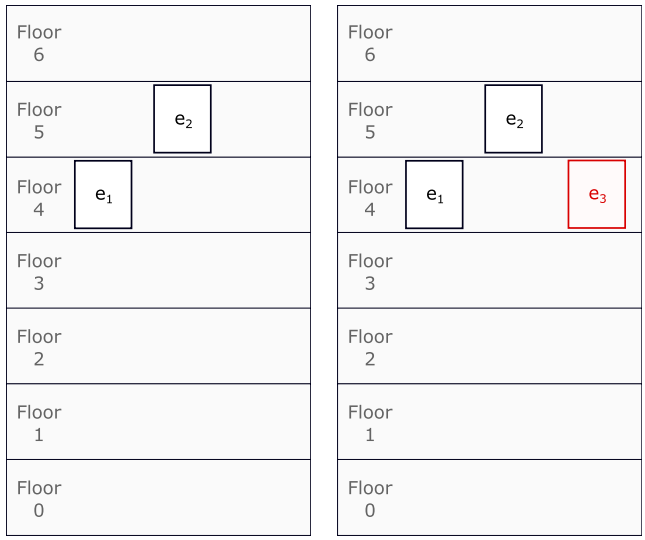

Fig. 1. Elevator control system scenarios. (a) Original elevators state. (b) MRIP2: Additional elevators.

Fig. 1(b) shows a sample instance of this MR. An additional elevator e_3 is enabled at *floor 4* in the follow-up test case, resulting in $E = \{4, 5, 4\}$. Consider the passenger calls $C = \{(1, 2, 3)\}$. In this case, the AWT should decrease, or in a worst case remain the same, when adding the new elevator

$$\begin{aligned}&\text{AWT}(\text{serve}(\{4, 5\}, \{\cdots\}, \{(1, 2, 3)\}))\\ &\gtrsim \text{AWT}(\text{serve}(\{4, 5, 4\}, \{\cdots\}, \{(1, 2, 3)\})).\end{aligned} \quad (4)$$

3) MRIP3: Faster Elevators: This pattern represents MRs where the configuration of the elevators P_f is changed so that the elevators from the follow-up test case are faster than those from the source test case. This can be implemented in various ways, such as increasing the nominal speed and acceleration, or reducing the motor start-up delay. As an example, the AWT of the follow-up test case is expected to improve due to the elevators being able to attend calls faster, resulting in the following MR:

$$\text{AWT}(\text{serve}(E, P_s, C)) \gtrsim \text{AWT}(\text{serve}(E, P_f, C)) \quad (5)$$

where the parameters in P_f allow the elevators to attend calls faster than those in P_s. Similar MRs could be defined considering other performance metrics. For instance, increasing the speed of elevators may result in a higher energy consumption.

For example, suppose the initial elevator positions from Fig. 1(a), the set of passenger calls $C_s = \{(1, 2, 3)\}$, and the parameters $P_s = \{\text{speed} = 1, \ldots\}$. Consider a follow-up test case is created by doubling the nominal speed of the elevators ($P_f = \{\text{speed} = 2, \ldots\}$). In this scenario, we should expect the AWT of the follow-up test case to be lower, or at worst, similar to the one observed in the source test case

$$\begin{aligned}&\text{AWT}(\text{serve}(\{4, 5\}, \{\text{speed} = 1, \ldots\}, \{(1, 2, 3)\}))\\ &\gtrsim \text{AWT}(\text{serve}(\{4, 5\}, \{\text{speed} = 2, \ldots\}, \{(1, 2, 3)\})).\end{aligned} \quad (6)$$

C. Autonomous Navigation System

Autonomous navigation systems can automatically plan and execute the route of a vehicle without human intervention. These vehicles (henceforth, referred to as *autonomous vehicles*) may include, for example, driverless cars, drones, submarines, and robotic vacuum cleaners. In practice, autonomous vehicles should be able to determine their own position in its frame of reference, identify and avoid obstacles, and calculate the optimal path to traverse a set of target points, among other tasks.

In what follows, we present some MRIPs and MRs for autonomous vehicles derived from the PV pattern. For the definition of the relations, we define the operation move(P, p_A, p_B, S, O), where P is a set of guidance points to follow, p_A is the origin point (the vehicle's initial position), p_B is the destination point, S is the vehicle's nominal speed, and O is the set of obstacles in the environment (which the vehicle should avoid). We will assume that the vehicle's path can be modeled as a sequence of guidance points corresponding to locations in the world where the vehicle is moving, and that the vehicle is capable of following these guidance points while avoiding the obstacles that may be encountered. Fig. 2(a) shows an example scenario where the vehicle (in green) must traverse several guidance points (in blue and purple) to reach its destination (in red). Throughout this work, we will use an autonomous car to illustrate scenarios for the proposed MRs, since this is the type of autonomous vehicle, we use for the empirical evaluation, but most of the MRs described in this section should be applicable to other types of vehicles (e.g., drones, boats, etc.) as long as their functionality can be mapped to the move operation we described.

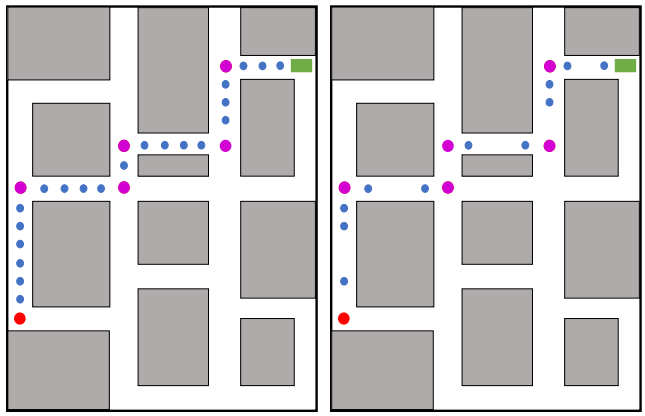

Fig. 2. Autonomous navigation system scenarios. (a) Original guidance points. (b) MRIP4: Fewer guidance points.

1) MRIP1: Faster Vehicles: This pattern represents MRs where the vehicle's nominal speed is increased in the follow-up test case. The expected performance should be the same or better in terms of travel time, since the vehicle can traverse its route faster as long as it can accelerate to its nominal speed. Thus, the time to destination (TTD) for a given route is expected to decrease or remain the same, resulting in the following MR:

$$\begin{aligned}&\text{TTD}(\text{move}(P, p_A, p_B, S_s, O))\\ &\gtrsim \text{TTD}(\text{move}(P, p_A, p_B, S_f, O))\end{aligned} \quad (7)$$

where the nominal speed from the follow-up test case S_f must be greater than the source nominal speed S_s.

For example, consider a scenario for a self-driving car where the nominal speed measured in km/h. The route contains the waypoints $P = \{w1, w2, w3\}$, where the starting point is $p_A = w1$, the goal is $p_B = w3$, and there are no obstacles ($O = \{\}$). If the nominal speed from the source test case is $S_s = 60$, and then the execute a follow-up test case with a higher nominal speed $S_f = 80$, the TTD should decrease as

$$\begin{aligned}&\text{TTD}(\text{move}(\{w1, w2, w3\}, w1, w3, 60, \{\}))\\ &\gtrsim \text{TTD}(\text{move}(\{w1, w2, w3\}, w1, w3, 80, \{\})).\end{aligned} \quad (8)$$

An analogous MRIP could be defined by decreasing the nominal speed rather than increasing it.

2) MRIP2: Additional Obstacles: This pattern represents the MRs where follow-up test cases are created by adding obstacles to the environment where the vehicle operates. In this case, the expected performance in terms of time or energy consumption should be worse, since the vehicle must overcome this new obstruction in its path by taking otherwise unnecessary actions. Obstacles may include static or dynamic objects (e.g., other

vehicles) as well as adverse environmental conditions (e.g., storms). The following is a specific MR derived from this pattern using TTD as the evaluated performance metric:

$$\begin{aligned}&\text{TTD}(\text{move}(P, p_A, p_B, S, O_s))\\ &\lesssim \text{TTD}(\text{move}(P, p_A, p_B, S, O_f)))\end{aligned} \tag{9}$$

where $O_f \supset O_s$, i.e., one or more additional obstacles have been placed in the vehicle's route.

Depending on the type of SUT and the obstacle types, the applications of this MR may vary. For solid obstacles where taking a longer route is necessary, we can expect an increase in both TTD and energy usage, whereas, for example, a condition, such as headwind, may only cause an increase in the energy usage if the navigation system is configured to compensate for it by increasing its throttle, e.g., ArduPlane[1], with airspeed throttle adjustment.

For the autonomous car example, consider a scenario where a static object (e.g., a cone in the middle of the road) is introduced as an obstacle, resulting in the car having to steer to avoid it. Since the cone is an additional restriction for the car, the alternative trajectory should always be less optimal than the original one performance-wise. As in the previous example, we have $P = \{w1, w2, w3\}$, $p_A = w1$, $p_B = w3$, and $S = 60$, and no obstacles in the source test case ($O = \{\}$). The follow-up test case is then generated by adding a cone to the obstacles, resulting in the following MR:

$$\begin{aligned}&\text{TTD}(\text{move}(\{w1, w2, w3\}, w1, w3, 60, \{\}))\\ &\lesssim \text{TTD}(\text{move}(\{w1, w2, w3\}, w1, w3, 60, \{\text{cone}\})).\end{aligned} \tag{10}$$

3) MRIP3: Reversed Path: This pattern groups the MRs where the path P is reversed in the follow-up test case. Intuitively, this should result in source and follow-up test executions having similar performance measures. For example, the following is an MR derived from this pattern expressing that the energy consumption φ should be similar when traversing the path forward and backward as

$$\varphi(\text{move}(P, p_A, p_B, S, O_s)) \simeq \varphi(\text{move}(P,' p_B, p_A, S, O_f))) \tag{11}$$

where P' is obtained by reversing the waypoints in P.

This is a very intuitive relation often used to illustrate MT [16], [46], although here we provide a novel perspective by using performance metrics as a proxy to reveal failures. This relation can also be considered an instance of the symmetry MRP proposed by Zhou et al. [65].

As an example, the energy consumed in the same scenario used to demonstrate the previous MRIPs should remain approximately the same if the path is reversed as

$$\begin{aligned}&\varphi(\text{move}(\{w1, w2, w3\}, w1, w3, 60, \{\}))\\ &\simeq \varphi(\text{move}(\{w3, w2, w1\}, w3, w1, 60, \{\})).\end{aligned} \tag{12}$$

4) MRIP4: Fewer Guidance Points: This pattern represents MRs where some of the guidance waypoints from the path of the vehicle are removed in the follow-up test case. In this case, the car should be able to traverse the path faster, since there are fewer guidance points to traverse and so the traversed distance will be shorter.

$$\begin{aligned}&\text{TTD}(\text{move}(P_s, p_A, p_B, S, O))\\ &\gtrsim \text{TTD}(\text{move}(P_f, p_A, p_B, S, O)))\end{aligned} \tag{13}$$

where $P_f \subset P_s$, i.e., some of the waypoints have been removed from the vehicle's path.

For our autonomous car example, if our path is $P_s = \{w1, w2, w3\}$, the following relation should hold:

$$\begin{aligned}&\text{TTD}(\text{move}(\{w1, w2, w3\}, w1, w3, 60, \{\}))\\ &\simeq \text{TTD}(\text{move}(\{w1, w3\}, w1, w3, 60, \{\})).\end{aligned} \tag{14}$$

IV. Evaluation

In this section, we report two experiments on the effectiveness of performance-driven MT of CPSs. Specifically, we aim to answer the following research questions (RQs).

1) *RQ1: Do the Generated MRs Trigger FPs? What Causes Them?* Due to the nondeterministic nature of performance measurements, FPs are likely to emerge. We aim to investigate to what extent FPs appear in practice.
2) *RQ2: Is Performance-Driven MT Effective in Revealing Failures in CPSs?* We aim to study the ability of performance-driven MT, and in particular, MRs derived from the PV pattern, to detect bugs in different types of CPSs. Automated regression test oracles will be used as baselines.
3) *RQ3: Do Particular MRIPs or Performance Metrics Perform Significantly Better Than Others?* We plan to compare the performance of different MRIPs and performance metrics. Also, we want to study whether the results from some of the input relations and metrics subsume, or rather complement, those obtained by other relations and metrics.

To answer these RQs, we employed two different case studies, whose main features are given in Table I.

A. Evaluation Metrics

In this section, we describe the key definitions and metrics used for the presentation of the experimental results with both case studies.

An MR can be instantiated into one or more *metamorphic tests* by running the source and follow-up test cases with specific input values and checking whether the relation holds. If the relation is violated, the metamorphic test is said to have failed, indicating a *test failure*. However, in nondeterministic programs—as the ones used in our case studies—the MR may be exceptionally violated by mere chance generating a *FP* [23], [49].

We use the following three different metrics to determine the effectiveness of our approach.

1) In first place, we use the *FPs*, which refers to the percentage of test failures on the original system executions. FPs may result in unnecessary debugging efforts, so the lower the number of FPs, the better.

[1]Online available at http://ardupilot.org/plane/docs/airspeed.html

TABLE I
MAIN CHARACTERISTICS OF THE EXPERIMENTAL CASE STUDIES

SUT	Language	Test cases (sources + follow-ups)	Execution time	Mutants	MRs	Metrics
Elevator	C	1340 (140 + 1200)	∼60 h	89	9	3
Autonomous car	Simulink	1300 (100 + 1200)	∼8 h	20	12	2

2) Second, we report the *mutation score (MS)*, which refers to the percentage of mutants killed by the MRs. Specifically, we consider a mutant as "detected" or "killed" when one or more of the metamorphic tests failed on the mutant, but not with the original system. The higher the MS, the better, since more seeded faults are detected.
3) Finally, we measured the *failure detection ratio (FDR)*, which is the percentage of metamorphic tests on mutants that resulted in a test failure. A higher percentage is better, since more potentially faulty behaviors are identified.

B. Experiment 1: Elevator Control System

In this experiment, we tested an industrial elevator dispatcher system developed by Orona [41], which inspired the example presented in Section III-B. A previous version of this experiment was presented in [5]. In what follows, we describe the SUT, performance metrics, MRs, experimental setup, and the results of the experiment.

1) System Under Test: An elevator is a complex CPS, where software and hardware interact with the goal of transporting passengers safely and by considering certain quality of service (QoS) measures. Among the components of the elevator installation, the traffic master is in charge of managing the passenger flow. This element is composed of different software modules, including the dispatching algorithm that decides which elevator should attend each call. The dispatching algorithm has a high impact on the QoS measures of the elevator installation. Different elevator dispatchers can be used to optimize different objectives depending on the installation requirements and traffic profiles. For this experiment, we used the most commonly used elevator dispatching algorithm from Orona [41], a leading elevator company in Europe, as the SUT. This dispatcher employs a deterministic rule-based algorithm, which optimizes for the best AWT for the passengers. The dispatcher's source code is written in C, so that it can be easily compiled into different targets.

Note that unlike other types of optimizers, such as source code compilers, a deterministic elevator dispatcher cannot output the optimal solution for any given scenario. This is because performing the optimal elevation dispatches requires the algorithm to know about the passengers that will arrive in the (near) future, since their effect on the QoS metrics will be affected by the actions of the dispatcher before they actually arrive. Since this information will not be available under real circumstances, the dispatcher algorithm will need to *predict* the expected passenger behavior and act accordingly, which may or may not be the best decision for a given scenario. In practice, the dispatcher algorithm will mostly optimize for the best QoS under the expected most common passenger behaviors, with some reasonable tradeoffs to avoid worst-case scenarios in less-expected cases.

Orona has a large suite of elevators dispatching algorithms, which need constant maintenance to address new functional requirements, new QoS measures, legislative changes, bug fixing, hardware obsolescence or system degradation, adaptation to building requirements, etc. When changes are made, Orona has a well-established verification and validation process of the dispatching algorithm before deploying the new release in real installations. In a first stage, tests are executed within a software-in-the-loop level. The software of the dispatching algorithm is an executable that communicates with a domain-specific simulator named Elevate [32]. Elevate simulates all the physical components of the elevator and provides a set of QoS measure results when the simulation has finished. The following stage is the hardware-in-the-loop phase. Here, the software of the dispatching algorithm is integrated with the rest of the software and hardware infrastructure, encompassing, among others, real-time operating systems, communication protocols, and the real target, in which the software is executed. In this stage, the tests are executed in real-time, and their goal is to validate the functional correctness of the release within the real infrastructure. Last, the software is deployed into the real system at operation. The elevator maintainer performs a set of manual tests to ensure that the software has been successfully deployed and that it works correctly. As the test level becomes more realistic, the test execution cost increases significantly, so it is important to detect bugs as early as possible during the verification and validation process. Unfortunately, the testing process largely relies on human oracles (i.e., the test engineer's judgement) to decide the final verdict for each test, which hinders full testing automation.

In Orona, a test for the dispatching algorithm is constituted by the passengers list and the building installation information. The *passengers list* represents a list of passengers that arrive to a landing floor, call an elevator, and request a destination. For each passenger, the following input values must be provided.

a) The arrival time.
b) The arrival floor.
c) The destination floor.
d) The weight of the passenger.
e) Capacity factor, i.e., the mass threshold at which the passengers will consider the elevator to be full.
f) The loading time.
g) The unloading time.
h) The expected passenger behavior when not all elevators serve all floors, e.g., waiting for the right elevator versus switching elevators until reaching the destination.

For our experiment, we set different values for the inputs a)–d) and use the default values for the remaining ones. Regarding the *building installation information*, it refers to an XML file with all the information of the building and elevators installation at which the SUT is being executed. For instance, it encompasses

the number of floors of a building, number of elevators, floors served by each of the elevators, maximum weight each elevator can lift, etc.

Note that this information is passed to the simulation environment, and the dispatcher algorithm only receives the information that it would get in a real installation. For example, the dispatcher does not receive the passengers list beforehand, it will only be notified of the passengers as they arrive, and their destination is only known after they get into an elevator and press a button.

2) Performance Metrics: For this experiment, we used the following performance metrics.

a) *Average Waiting Time:* The average time from the moment a landing call is issued until an elevator stops to attend the call measured in seconds. This is among the most important metrics for providing a good user experience [9], and it is the metric that the dispatcher we use for the experiments is designed to optimize.

b) *Total Distance:* The sum of the distances traversed by all the elevators of the building, measured in floors. We consider this metric because an unexpected value may reveal behaviors, such as consistently not assigning elevators that are close to the landing calls or unnecessarily dispatching multiple elevators to a single call.

c) *Total Movements (TM):* The count of all the movements (i.e., engine start-ups) of all the elevators of the building. We considered that this metric may reveal inefficient or bugged behaviors in a similar way to *TD*.

3) Metamorphic Relations: In the following, we describe the MRs used in the experiment. These relations were derived from the MRIPs presented in Section III-B, which in turn are instances of the more general PV pattern proposed in our work. These MRs were defined based on our knowledge of the dispatcher—acquired during our long-term collaboration in technology transfer with Orona—and specific inquiries made to the engineers involved in the development and maintenance of the dispatcher.

The following MRs are defined assuming the dispatcher always provides an optimal assignment. However, as previously explained, the dispatcher under test provides approximate solutions. In practice, this means that FPs could arise. To mitigate this, as explained in Section III, we define approximate relations (i.e., $\simeq, \gtrsim$, and $\lesssim$) instead of strict ones (i.e., $=, \geq$, and $\leq$). In practice, these are implemented using tolerance thresholds, meaning that only violations exceeding a certain value will be consider as failures. The threshold values used in our experiments are detailed later in the experimental setup.

In what follows, we revisit the MRIPs defined in Section III-B, describing the MRs derived from them in the context of our case study. For the sake of simplicity, we use the same notation introduced in Section III-B, where $\text{serve}(E, P, C)$ denotes an execution of the dispatcher, E is a set of floors indicating the positions of the elevators, P are the elevator parameters, and C is the list of passenger calls.

a) *MRIP1:* Additional Calls: We propose several MRs where the follow-up test input is created by appending an additional passenger call to the source test case. Formally, the input relation can be defined as $C_f = C_s \cup c'$, where c' is the additional passenger call. In this scenario, the TD traversed by the elevator should increase [see Section III-B and (1)]. In practice, however, we found that it is possible to define a tighter—and therefore more likely to reveal failures [46]—relation by making a rough estimation of the worst case TD required to be traversed, measured as the sum of the largest possible distance to the source floor and the distance between the source and the destination floors. This can be expressed as the following MR:

$$\text{TD}(\text{serve}(E, P, C_f)) \lesssim \text{TD}(\text{serve}(E, P, C_s)) + \text{TD}_w(c') \quad \texttt{MR1}_{\text{TD}} \tag{15}$$

where $\text{TD}_w(c')$ is the worst case TD that an elevator will have to traverse for serving c' and calculated as $\text{TD}_w(c) = \max(c_s - 1, \text{FLOORS} - c_s) + |c_s - c_d|$, where $\max(c_s - 1, \text{FLOORS} - c_s)$ is the longest possible distance that may need to be traversed to reach the source floor c_s and $|c_s - c_d|$ is the distance from the source floor to the destination floor of the passenger.
A similar relation is defined based on the expected impact on the AWT, namely

$$\text{AWT}(\text{serve}(E, P, C_f)) \lesssim \text{AWT}(\text{serve}(E, P, C_s)) + \text{WT}_w(c') \quad \texttt{MR1}_{\text{AWT}} \tag{16}$$

where $\text{WT}_w(c')$ is the estimated worst case waiting time for c', calculated as $T(\max(c_s - 1, \text{FLOORS} - c_s))$, with $\max(c_s - 1, \text{FLOORS} - c_s))$ is the longest possible waiting distance described previously, and $T(\text{distance})$ is a formula that calculates the time in seconds that it takes an elevator to traverse the given distance considering its speed, acceleration, and jerk (which are the parameters that can be obtained from the building installation XML).
Finally, when adding a call to the passenger list, the number of total movements of the elevators should increase or remain the same. This is expressed as the following MR:

$$\text{TM}(\text{serve}(E, P, C_f)) \gtrsim \text{TM}(\text{serve}(E, P, C_s)) \quad \texttt{MR1}_{\text{TM}}. \tag{17}$$

b) *MRIP2: Additional Elevators:* We define MRs where the follow-up test input is created by enabling one or more additional elevators to the source test. Formally, we can define this as $E_f = E_s \cup E'$, where E' is a set of one or more new elevators. For elevator dispatching algorithms that aim at obtaining the best passenger waiting times, this should improve the AWT, or at least remain the same, yielding the following MR:

$$\text{AWT}(\text{serve}(E_f, P, C)) \lesssim \text{AWT}(\text{serve}(E_s, P, C)) \quad \texttt{MR2}_{\text{AWT}}. \tag{18}$$

Conversely, the TD traversed is likely to increase if more elevators are moving in parallel. This is reflected in the following MR:

$$\text{TD}(\text{serve}(E_f, P, C)) \lesssim \text{TD}(\text{serve}(E_s, P, C)) \times (1 + |E_f| - |E_s|) \quad \texttt{MR2}_{\text{TD}} \tag{19}$$

where $|E_f| - |E_s|$ is the upper bound of the traversed distance based on the number of additional elevators. For instance, if we add two more elevators, the TD could be increased by up to 200% in the worst case.
Similarly, the total number of movements (TM) is expected to increase if new elevators are added, namely

$$\text{TM}(\text{serve}(E_f, P, C)) \lesssim \text{TM}(\text{serve}(E_s, P, C)) \times \left(1 + \frac{|E_f| - |E_s|}{2}\right) \quad \texttt{MR2}_{\text{TM}} \tag{20}$$

where $\frac{|E_f|-|E_s|}{2}$ is the upper bound of the traversed distance based on the number of additional elevators. Here, TM is only expected to increase by up to 50% more for each additional elevator, because out of the two movements that the elevators might have to perform for each call (one for attending to the calling floor, and another one for travelling to the destination floor), only the first one may increase due to attending calls in parallel.

c) *MRIP3: Faster Elevators:* We define several MRs where the follow-up test case is created by increasing the speed of the elevators in the source test case. Speed is increased in all the elevators of the scenario equally. For the elevators dispatching algorithm under test, this should generally improve the AWT, since faster elevators should be able to attend the calls more rapidly. This can be expressed as the following MR:

$$\text{AWT}(\text{serve}(E, P_f, C)) \lesssim \text{AWT}(\text{serve}(E, P_s, C)) \quad \texttt{MR3}_{\text{AWT}}. \tag{21}$$

On the other hand, TD and TM are expected to increase. This is because when the elevators move slower, the passenger calls on the same floor will accumulate, and the elevators will end up carrying more passengers, and therefore traversing less distance and making fewer movements. Based on this, we define the following MR using the TD traversed:

$$\text{TD}(\text{serve}(E, P_f, C)) \gtrsim \text{TD}(\text{serve}(E, P_s, C)) \quad \texttt{MR3}_{\text{TD}}. \tag{22}$$

And, an analogous one using the total number of movements

$$\text{TM}(\text{serve}(E, P_f, C)) \gtrsim \text{TM}(\text{serve}(E, P_s, C)) \quad \texttt{MR3}_{\text{TM}}. \tag{23}$$

4) Experimental Setup: The test cases are based on a template project from a real building with ten floors and up to six elevators. For the MRs, our inputs are as follows.

a) The set of available elevators (at least two, and up to six), including their positions (floors 1–10).
b) The set of relevant elevator parameters to change their speed (explained later on).
c) The passengers list, where the arrival time, source floor, destination floor, and passenger weights are variable and the rest of the parameters are set to default values.

The rest of the building parameters, elevator specs, etc., are taken from the building template and will be identical for all tests.

Source test cases were randomly generated based on the template project of the building. Each test case has a duration of roughly 3 min (simulation time) on average. For each generated test case, we selected a random number of elevators (between 2 and 6), a random initial floor for each elevator, and a random passenger list generated by uniformly distributing the calls across a fixed time period. The source and destination floors for each call were also uniformly selected from the ten landing positions of the building. In total, we generated 140 random source test cases and 1200 follow-up test cases. In total, there are 1200 pairs of source and follow-up test cases, which are 420 for MRIP1, 360 for MRIP2, and 420 for MRIP3.

The follow-up test cases for the MRs derived from MRIP1 (additional calls) were generated by appending a single additional call to the end of the passengers list, i.e., the new call is always the last one. This is to ensure that the additional call has no unexpected effects on the execution of the rest of the test case. On the other hand, the follow-ups for MRIP2 (additional elevators) was implemented by randomly selecting a number of elevators $|E_f|$ for the follow-up test case, given the constraint $|E_s| < |E_f| \leq 6$ (the limit of six elevators is specific to the elevator installation template we use in our experiments). The additional elevators are given random initial positions. As for the implementation of MRIP3 (faster elevators), C_f is generated by modifying the elevator parameters from C_s, i.e., speed, acceleration, jerk, door open time, door close time, motor start delay, and leveling delay. Specifically, we select a multiplier m, which is a constant integer number ranging between 2 and 4, and we multiply the speed, acceleration, and jerk parameters by that constant, while the rest of the mentioned parameters are divided by it.

To measure the effectiveness of the MRs at detecting bugs, we seeded artificial faults into the SUT using mutation testing. This approach has been found to be a valid substitute for testing with real faults [27]. Specifically, we created 89 faulty variants (mutants) of the elevator dispatcher by seeding faults using traditional mutation operators, including arithmetic, logical, and relational operator mutations [1]. Faults were manually seeded in a uniform manner throughout the sections of the source code that are relevant in the simulation environment. This process was performed by one of the authors, who is a domain expert and has extensive experience with this system. The behavior of the generated mutants was also checked in order to assert that none of them were semantically equivalent. Both the 140 source test cases and the 1200 metamorphic tests (pairs of source and follow-up test cases) were executed against the original dispatcher and the 89

TABLE II
MAIN CHARACTERISTICS OF THE USED TEST CASES FOR THE CONSIDERED BASELINE, WHICH IS THE CURRENT APPROACH USED IN ORONA

Test case	# of up calls	# of down calls	# of detected mutants	Simulation time (h:min)
real1	2756	1711	18	8:30
real2	3086	2366	18	9:10
real3	3438	3117	18	11:45
real4	3508	3050	21	13:35
theoretical1	3994	3377	20	12:55
theoretical2	3950	3379	18	12:55
theoretical3	3983	3379	26	12:55
theoretical4	3989	3402	18	12:55
theoretical5	3989	3387	18	12:55
theoretical6	3964	3384	19	12:55
theoretical7	3977	3386	21	12:55
theoretical8	3919	3433	21	12:55
theoretical9	3976	3354	18	12:55
theoretical10	3945	3407	20	12:55

mutants resulting in a total of $(140 + 1200) \times 90 = 120600$ test executions.

As previously mentioned, to implement approximate relations in practice, we defined tolerance thresholds for some of the MRs. After some preliminary tests, we defined a threshold of 30% for the relations $\mathrm{MR3_{TD}}$ and $\mathrm{MR3_{TM}}$. In $\mathrm{MR3_{TD}}$, for example, the exact assertion we used is $\mathrm{TD}(\mathrm{serve}(E, P_f, C)) \geq \mathrm{TD}(\mathrm{serve}(E, P_s, C)) \times 0.7$. The rest of the MRs were implemented by evaluating them strictly, without any threshold or other tolerance mechanisms, since they did not yield any FPs during preliminary testing. For instance, $\mathrm{MR3_{AWT}}$ was evaluated as $\mathrm{AWT}(\mathrm{serve}(E, P_f, C)) \leq \mathrm{AWT}(\mathrm{serve}(E, P_s, C))$.

5) Baseline: We considered the current practice for testing elevator dispatching algorithm versions at Orona, as explained in [3]. Current approaches use a regression test oracle. Such oracles use a previous version of the SUT and compare the AWT performance metric over time between both the SUT and its previous version. Specifically, the regression oracle has different thresholds for each of the following three possible failing conditions.

1) Major AWT degradation over a single 5 min period.
2) Accumulated AWT degradation over multiple 5 min periods.
3) Degradation of the total AWT.

In addition, the current automated process at Orona for executing such tests is by means of employing 14 full-day test cases, which are both theoretical (i.e., synthetic test cases) and real (i.e., test cases obtained from the building installation). We used the very same test suite used at Orona for testing the dispatcher under test, composed of ten theoretical test cases and four real test cases. Table II gives the key characteristics of the test cases in the baseline.

Out of the 89 mutants created, eight of them ended in an infinite simulation due to one or more call left unattended. This could easily be detected by an implicit timeout, and therefore were marked as killed. The total simulation time of these test cases was 10 330 min (approximately seven days), but notice that each test case should be executed twice (one with the SUT and the other with the reference implementation). Therefore, the total execution time is 20 660 min (approximately 14 days).

TABLE III
EVALUATION RESULTS ON THE ELEVATOR DISPATCHER

MRIP	MR	MS (%)			FDR (%)	FP (%)
MRIP1	$\mathrm{MR1_{AWT}}$	29.21	85.39	88.76	0.19	0.00
	$\mathrm{MR1_{TD}}$	65.17			0.88	0.24
	$\mathrm{MR1_{TM}}$	75.28			0.98	0.24
MRIP2	$\mathrm{MR2_{AWT}}$	42.70	42.70		2.27	0.00
	$\mathrm{MR2_{TD}}$	13.48			0.08	0.00
	$\mathrm{MR2_{TM}}$	5.62			0.03	0.00
MRIP3	$\mathrm{MR3_{AWT}}$	31.46	44.94		0.55	0.24
	$\mathrm{MR3_{TD}}$	33.71			0.24	0.00
	$\mathrm{MR3_{TM}}$	6.74			0.06	0.00

6) Experimental Results: The evaluation of the proposed MRs resulted in a single FP for each of these three MRs: $\mathrm{MR1_{TD}}$, $\mathrm{MR1_{TM}}$, and $\mathrm{MR3_{AWT}}$. The analysis from the corresponding test cases revealed some suspicious behaviors from the elevator dispatcher for both of the cases from MR1. After consulting with domain experts, one of those cases was due to a mismatch between the way Orona's controllers and Elevate send information to the dispatcher, which can cause abnormal results in some simulation scenarios. This discrepancy has not been fixed because it requires to either maintain two separate versions of the dispatcher or modifying Orona's controllers, and both options were deemed too expensive compared with tolerating some infrequent deviations in the simulations. Another FP revealed a case where an elevator skipped a passenger call in a scenario where stopping for the passenger would have been the obvious choice. This was a corner case already known by Orona developers, who preferred to leave the system as is to provide a better performance on average. Reporting a failing condition can be considered acceptable in both of these cases, since there are abnormal conditions involved. As for the FP for $\mathrm{MR3_{AWT}}$, the change in speed just happened to cause the scenarios to diverge in a way that happened to favor the slower elevators, which is statistically unlikely but possible, and no obvious abnormal behavior from the dispatcher was observed in either of the test executions.

As for their effectiveness, all the MRs combined killed 79 out of 89 mutants, which results in an MS of 88.76%. On the other hand, there were 1593 out of 320 400 metamorphic test failures, which corresponds with an FDR of 0.5%. Recall that there are 420 test pairs for each MR derived from MRIP1 and MRIP3, and 360 test pairs for each MR derived from MRIP2, so considering there are 89 mutants, the number of metamorphic tests on mutants is calculated as: $89 \times (420 \times 3 + 360 \times 3 + 420 \times 3) = 320400$. The original dispatcher was also verified with the proposed MRs and the same test cases, and three of the MRs yielded FPs in a single case each.

Table III details the evaluation results. For each MR, the table gives the MS, FDR, and the percentage of FPs. In addition, the table gives the aggregated MS for each MRIP and in total. Furthermore, Fig. 3 shows the exact number of mutants killed by all the MRs for each MRIP, and the intersection of mutants killed for all the MRIP combinations. On the other hand, Fig. 4 shows the same, but with the MRs grouped by the QoS metric they use. The MS of each MR ranged from 5.62% to 75.28%,

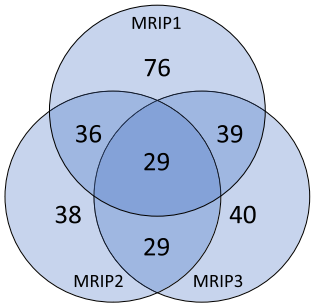

Fig. 3. Mutants killed per MRIP (out of 89).

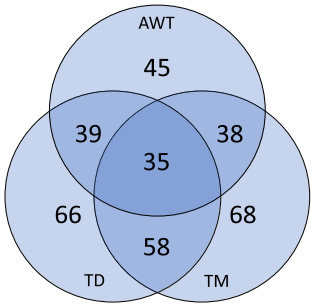

Fig. 4. Mutants killed per QoS metric (out of 89).

whereas the FDR ranged from 0.03% to 2.27%. This suggests a great diversity in the results of the different MRs.

When comparing our approach with the baseline (i.e., regression test oracle, see Section IV-B5), our approach killed 32 more mutants. That is, the current approaches used in Orona are killed 47 out of 89 mutants, resulting in an MS of 52.8% (our MRs killed 79 out of 89 mutants, achieving an MS of 88.76%). In terms of execution time, the sum of the costs from all the source and follow-up test cases used in our experiments is 3678.62 min (approximately two days and a half), whereas the test cases used by the baseline have a total cost of 10 330 min (approximately seven days), and the actual cost is twice as much if the reference implementation needs to be run as well. Overall, the results show that MT is significantly more cost-effective than the baseline. It is worth noting, however, that our MRs did yield three FPs, which may lead to some unnecessary efforts from the domain experts, whereas the regression oracles do not have this issue on a pseudo-deterministic simulation environment.

$\mathrm{MR1_{TM}}$ obtained the highest MS (i.e., 75.28%), which indicates that this MR is capable of detecting more (types of) failures than the rest, at least for the seeded faults used in our evaluation. Furthermore, it also achieved a relatively high FDR (i.e., 0.98%), the second highest for all the proposed MRs. $\mathrm{MR1_{TD}}$ obtained very similar but slightly worse results. On the other hand, $\mathrm{MR2_{AWT}}$ obtained the highest FDR (i.e., 2.27%). For MRIP2 (additional elevators), $\mathrm{MR2_{AWT}}$ also appears to subsume the other two metrics, as they do not contribute to the aggregate MS, and their results are overall much worse. As shown in Fig. 3, MRIP1 (additional calls) is overall the most effective pattern in terms of MS by a large margin, since all of the derived MRs have a fairly high score and aggregated, they only miss three of the mutants that can be killed by the other MRIPs. As for Fig. 4, surprisingly, AWT appears to be the least effective metric, even though this is supposed to be the most relevant one to the dispatcher algorithm under test. Nevertheless, AWT still detects six mutants missed by TD and seven mutants missed by TM.

Overall, it seems that the best results can be obtained by using $\mathrm{MR1_{TD}}$ and $\mathrm{MR1_{TM}}$ due to their apparent ability to detect many different types of failures (as indicated by their high MS), combined with $\mathrm{MR2_{AWT}}$ due to its significantly higher FDR (more than three times higher than any other MR). These three MRs combined can detect 78 of the mutants (MS of 87.64%), only one less than when using all nine MRs combined.

C. Experiment 2: Autonomous Driving System

For this experiment, we tested an autonomous vehicle in a simulation environment. This is a particular case of the autonomous navigation system presented in Section III-C. A very preliminary version of this experiment was presented in [58]. In what follows, we describe the SUT, performance metrics, MRs, experimental setup, and the results of the experiment.

1) System Under Test: For this experiment, we tested an autonomous car simulated using MATLAB and Simulink, based on the model published by MathWorks [37]. Both, MATLAB and Simulink are popular environments for the development of CPSs [19], and they are also widely used by the scientific community for research on CPS testing [64], [66].

Within the autonomous car, one of the most important components is the navigation controller, which drives the vehicle through the optimal path from an origin location to the destination by traversing a set of guidance points. This controller is the SUT for this experiment. The navigation controller used in our evaluation uses a reference speed value and a set of guidance points, including the current and destination positions, as its inputs, and adjusts the vehicle throttle and steering in order to move it to the destination. The vehicle will try to move through the guidance points in order until the destination point is reached, where the vehicle will stop. The test execution will end when the vehicle has stopped completely at the destination point.

This Simulink model allows the simulation of different driving scenarios, which can be used as test cases for our SUT, the underlying vehicle controller. Each test case for this system is composed of five different inputs: a) the origin point; b) the destination point; c) a set of guidance points to go through; d) the vehicle's nominal speed; and e) a set of obstacles on the way. In this case, the obstacles are other vehicles that move in a straight line and may cross the path of the car. Whenever another vehicle blocks the route, the implemented obstacle avoidance system makes the car slow down to a stop and waits until the path is clear. Since the scenarios may involve multiple cars, we refer to the car controlled by the SUT as the *ego car* in order to distinguish it from other cars that are used as obstacles.

2) Performance Metrics: For this experiment, we used the following performance metrics, which were selected among those typically used in the domain of autonomous vehicles [24].

a) *Time to Destination:* This is the time required for the vehicle to reach its destination from its initial position, measured in seconds. The vehicle controller is expected to traverse its assigned route as fast as possible, as long as the nominal speed is respected and there is no significant risk of collision against obstacles or deviating from the trajectory (e.g., the vehicle should reduce its speed to a reasonable value before steering with a sharp angle).

b) *Total Trajectory Offset (TTO):* This is the integral of the offset between the vehicle's angle and the reference angle. When the vehicle is not facing the next reference point, the larger the difference between the current and expected angles, and the longer it takes to correct its angle, the higher this metric will be. A high trajectory offset may be caused, for instance, because its speed was too high when taking a turn, which makes it difficult for the vehicle to correct its direction. The vehicle controller is expected to keep the value of this metric reasonably low.

3) Metamorphic Relations: For the definition of the MRs, we used the same notation introduced in Section III-C, where an execution of the autonomous navigation system is denoted by the operation move(P, p_A, p_B, S, O). To reiterate, the inputs of the system are a set of guidance points to follow P, the origin point p_A, the destination point p_B, the nominal speed S, and a set of obstacles O. As in the previous experiment, we present the MRs derived from the MRIPs presented in Section III-C, which in turn are instances of the proposed PV pattern, as follows.

Just as in the previous case study, the MRs described here may be violated by a small margin in practice, due to factors, such as nondeterminism, limited precision of the simulation, or MRs assuming unrealistically ideal behavior from the SUT. Therefore, just as before, we define approximate relations (i.e., $\simeq, \gtrsim,$ and $\lesssim$), and we specify the threshold values used in our experiments later, in the experimental setup.

a) *MRIP1: Faster Vehicles:* We propose several MRs where the follow-up test input is created by increasing the original nominal speed. Formally, $S_f > S_s$. When this happens, the TTD of the follow-up test case should be lower than the source test case or in the worst case similar. This can be expressed as the following MR:

$$\text{TTD}(\text{move}(P, p_A, p_B, S_f, O)) \lesssim \text{TTD}(\text{move}(P, p_A, p_B, S_s, O)) \quad \texttt{MR1}_{\text{TTD}}. \quad (24)$$

On the contrary, the TTO should increase, or in the best case, it should be similar, because the car is more difficult to control as the speed increases. This can be expressed as the following MR:

$$\text{TTO}(\text{move}(P, p_A, p_B, S_f, O)) \gtrsim \text{TTO}(\text{move}(P, p_A, p_B, S_s, O)) \quad \texttt{MR1}_{\text{TTO}}. \quad (25)$$

b) *MRIP2: Additional Obstacles:* We define several MRs where the follow-up test input is constructed with an extra obstacle in the vehicle's path. This can be expressed as $O_f \supset O_s$. The TTD should increase due to the vehicle having to dodge an extra obstacle. This can be expressed as the following MR:

$$\text{TTD}(\text{move}(P, p_A, p_B, S, O_f)) \gtrsim \text{TTD}(\text{move}(P, p_A, p_B, S, O_s)) \quad \texttt{MR2}_{\text{TTD}}. \quad (26)$$

Conversely, the TTO metric should not change, since the SUT we use in this case should just stop and wait without altering its trajectory. Thus, the following MR expresses this relation:

$$\text{TTO}(\text{move}(P, p_A, p_B, S, O_f)) \simeq \text{TTO}(\text{move}(P, p_A, p_B, S, O_s)) \quad \texttt{MR2}_{\text{TTO}}. \quad (27)$$

c) *MRIP3: Reversed Path:* We define several MRs by swapping the origin and destination points. This should not make any difference for the metrics we use, since all the roads are bidirectional in our test scenarios, and therefore the vehicle's trajectory in the follow-up test should be exactly the reverse of the one in the source test case. This can be expressed as the following MRs considering both TTD and TTO:

$$\text{TTD}(\text{move}(P, p_A, p_B, S, O)) \simeq \text{TTD}(\text{move}(P,'p_B, p_A, S, O)) \quad \texttt{MR3}_{\text{TTD}}$$

$$\text{TTO}(\text{move}(P, p_A, p_B, S, O)) \simeq \text{TTO}(\text{move}(P,' p_B, p_A, S, O)). \quad \texttt{MR3}_{\text{TTO}}$$

d) *MRIP4: Fewer Guidance Points:* We have defined some MRs where some points of the guidance path are removed, i.e., $P_f \subset P_s$. For our SUT specifically, we define a *tighter* version of the MRIP where only *nonessential* guidance points are removed from the path. We assume that the path contains nonessential guidance waypoints that only help the vehicle navigate to the next goal more accurately, but have no significant effect on the trajectory of the car, such as the blue waypoints shown in Fig. 2. Note that p_A and p_B should never be removed, and just as in the example shown in Fig. 2(b), the purple waypoints should also not be removed in order to avoid severe alterations in the trajectory. Intuitively, this transformation should now result in similar performance measurements. We can therefore define the following output relations for TTD and TTO:

$$\text{TTD}(\text{move}(P,' p_A, p_B, S, O)) \simeq \text{TTD}(\text{move}(P, p_A, p_B, S, O)) \quad \texttt{MR4}_{\text{TTD}}$$

$$\text{TTO}(\text{move}(P,' p_A, p_B, S, O)) \simeq \text{TTO}(\text{move}(P, p_A, p_B, S, O)) \quad \texttt{MR4}_{\text{TTO}}.$$

4) Experimental Setup: This empirical evaluation is based on short-scenario test cases, which have a duration of 2 min (simulation time) on average. Source test cases were randomly generated from a template project of a city modeled in Simulink, which includes the ego car and two additional cars, which can act as obstacles. This city's map contains 51 guidance points for the navigation system, each of which is a joint between roads. For each generated test case, we selected two random points: a) p_A;

and b) p_B, and a random trajectory and speed for each of the other two simulated cars (obstacles). The rest of the environmental conditions (weather, road friction, etc.) were identical for all the test cases. The navigation controller from the ego car calculates the shortest path between the selected points and follows it. In total, we generated 100 random source test cases and 600 follow-up test cases, resulting in $100 + 600 = 700$ individual test cases. In total, there are 600 pairs of source and follow-up test cases, 300 for MRIP1, 100 for MRIP2, 100 for MRIP3, and 100 for MRIP4.

The follow-ups for MRIP1 (faster vehicles) were generated multiplying the nominal speed value in the source test case by a constant. In this case, we generated three different MRs for each metric, as we used three different constant multipliers for the speed: a) 1.1 for MR1.1; b) 1.2 for MR1.2; and c) 1.3 for MR1.3. On the other hand, the follow-ups for MRIP2 (additional obstacles) were implemented by adding an obstacle within the ego vehicle's path, making sure that it will interfere with its operation, forcing it to stop and wait. For the implementation of MRIP3 (reversed path), the follow-ups were generated by reversing the path to be traversed by the vehicle. Since the pathfinding algorithm is not a part of the system that we are testing and there is a risk that the route calculated by swapping the initial and destination points is different, the path for the original initial and destination points is calculated first, and then the whole path is reversed for the follow-up test case. Finally, the follow-ups for MRIP4 (fewer guidance points) were generated by removing 20% of the guidance points in the path. Given the source path, 20% of the guidance points are selected with a uniform random function and discarded from the follow-up path, but the initial and destination points are never selected for removal, so that the path is always similar in both test cases. This process works for our system because the waypoints are relatively close to each other, so the trajectory remains very similar even if some arbitrary points are removed.

Similarly to our previous case study, mutation testing was used in order to assess the effectiveness of the proposed MRs. Specifically, we created 20 faulty models (mutants) of the autonomous vehicle. Most of the mutants contain a seeded fault on the vehicle control block, as this is the main component. Some other mutants simulate failures in sensors and other components (e.g., bad reference speed input) instead. The faults were seeded manually using traditional mutation operators [11], and equivalent or broken models were checked for and discarded. All the test cases were executed against the original system and the 20 mutants resulting in a total of $(100 + 600) \times 21 = 14700$ test executions. Some of the test executions on the original system did not terminate correctly after a timeout (i.e., the vehicle did not stop at its destination), either due to the vehicle not having enough time to reach its destination, or due to the vehicle stopping too far away from the destination to trigger the stopping condition. The corresponding test cases have been ignored in our evaluation (including the mutant executions of these test cases). In total, four out of 100 source test cases and one of the follow-ups for MRIP2 did not terminate correctly, so there are $100 - 4 = 96$ test pairs for every MR and mutant form MRIP1, MRIP3, and MRIP4, and $100 - 4 - 1 = 95$ test pairs for MRIP2.

After some tests, we defined a threshold of 50% for $\mathrm{MR3_{TTO}}$ and $\mathrm{MR4_{TTO}}$, and a threshold of 15% for every other MR, in order to implement approximate operators (i.e., $\simeq$, $\gtrsim$, and $\lesssim$). Both of the 50% thresholds were used because the MRs can cause unexpected changes to the performance metrics in some cases, particularly in very short test cases (some of our test executions were shorter than 10 s). In the case of $\mathrm{MR3_{TTO}}$, the vehicle always drives through the right lane, which means that if the vehicle drives through the inner lane in the source test case, it will drive through outer lane on the follow-up, so the sharpness of the turns will differ. As for $\mathrm{MR4_{TTO}}$, removing some key waypoints might allow the vehicle to take a shorter path and make smoother turns.

5) Baseline: Since there is no previously existing test oracle for this system, we have implemented a simple threshold-based oracle, which will raise an alarm if any of the performance metrics drop below a certain threshold value. This oracle is a simple version of the one proposed in [24], with the following differences.

a) The oracle operates on global performance metrics rather than per road sector.
b) We compute thresholds for $\frac{\text{TTD}}{\text{distance}}$ and $\frac{\text{TTO}}{\text{distance}}$.
c) We do not allow any FPs when we compute the optimal thresholds.

These characteristics ensure that the baseline is comparable to our MRs. The thresholds cannot be calculated directly for TTD or TTO, since both of these metrics increase as the test execution progresses, making the thresholds dependent on how long the test case is. This is why the thresholds are calculated for these performance metrics *over distance*, where distance is the sum of the distances between the waypoints that the vehicle must traverse. Note that we chose to not allow any FPs in the thresholds calculation, because our MRs are also expected to have zero or very few FPs.

In order to compute the thresholds, we simply take the maximum values of the metrics obtained for a given test suite, which are the minimum values that ensure no FPs, as

$$\max_{t \in T} \left(\frac{\text{TTD}_t}{\text{distance}_t} \right) \quad \texttt{Threshold}_{\text{TTD}}$$

$$\max_{t \in T} \left(\frac{\text{TTO}_t}{\text{distance}_t} \right) \quad \texttt{Threshold}_{\text{TTO}}$$

where T are the test executions of the original system on the 1300 test cases used for our MRs ($100 + 12 \times 100$, source test-cases plus follow-ups for all MRs), but just as for our MRs, the test executions that did not terminate are ignored. Analogously to the approach followed for the definition of MRs, we increased the thresholds by 10% to allow small variations in the performance measurements.

Table IV gives the results obtained by the baseline. The "Tolerance" rows show the results obtained with the calculated thresholds increased by 10%, whereas the "Perfect" rows show the results for the exact thresholds. Note that the "Perfect" results are the best results that can be obtained with this approach

TABLE IV
BASELINE RESULTS ON THE AV

Thresholds	Metric	MS (%)		FDR (%)		FP (%)
Tolerance	TTD	90	95	6.89	16.12	0.00
	TTO	60		9.76		0.00
Perfect	TTD	90	95	7.02	16.74	0.00
	TTO	65		10.25		0.00

TABLE V
EVALUATION RESULTS ON THE AUTONOMOUS VEHICLE

MRIP	MR	MS (%)			FDR (%)	FP (%)
MRIP1	$MR1.1_{TTD}$	25	55	100	0.31	0.00
	$MR1.1_{TTO}$	50			2.08	0.00
	$MR1.2_{TTD}$	20			0.63	0.00
	$MR1.2_{TTO}$	55			2.29	0.00
	$MR1.3_{TTD}$	15			0.83	0.00
	$MR1.3_{TTO}$	55			2.86	0.00
MRIP2	$MR2_{TTD}$	25	80		1.05	0.00
	$MR2_{TTO}$	80			7.79	0.00
MRIP3	$MR3_{TTD}$	90	100		14.58	0.00
	$MR3_{TTO}$	65			11.98	4.17
MRIP4	$MR4_{TTD}$	85	85		3.18	0.00
	$MR4_{TTO}$	55			2.71	0.00

and test suite without FPs, since we use the same test suite for calculating the thresholds and for evaluating them.

As a sanity check, we validated our test oracles by generating and running an additional test suite of 100 random test cases and confirming that all the performance measurements where under the thresholds with 10% tolerance. The thresholds without tolerance, on the other hand, resulted in FPs, indicating that they may be too tight to be used in practice.

6) Experimental Results: This evaluation resulted in four FPs from $MR3_{TTO}$. The analysis from the corresponding test cases revealed that all the FPs were caused by the curves having a different sharpness when traversing them in either direction, as explained at the end of Section IV-C4 where we discuss the selected tolerance thresholds. Usually, such cases would be compensated by having balanced right- and left-hand side turns, and for short test cases, the effect would not be too significant. The test cases resulting in FPs had a duration of around 30 s and unbalanced right- and left-hand side curves, resulting in an accumulated difference in the TTO, which exceeded the tolerance threshold for the MRs. These FPs could be avoided by increasing the tolerance threshold for this MR, or alternatively, using only longer test cases (e.g., longer than 1 min), such that having significantly unbalanced curves becomes very unlikely.

Regarding their effectiveness, all the proposed MRs combined killed the 20 mutants, which means that the MS is 100%. On the other hand, there were 964 out of 23 000 metamorphic test failures on mutants. This corresponds to an FDR of 4.19%. Recall that, there are 20 mutants, 12 different MRs, and 100 test pairs per MR, but the simulation did not finish in four of the source test cases and in one of the follow-up test cases from MRIP2. Considering this, the number of metamorphic tests on all mutants is calculated as: $20 \times 12 \times 96 - 20 \times 2 \times 1 = 23000$.

Table V gives the MS, FDR, and FPs obtained by each MR, as well as the total results for each MRIP and the total aggregate results. Furthermore, Fig. 5 shows the number of mutants killed by all the MRs derived from each MRIP and the intersection

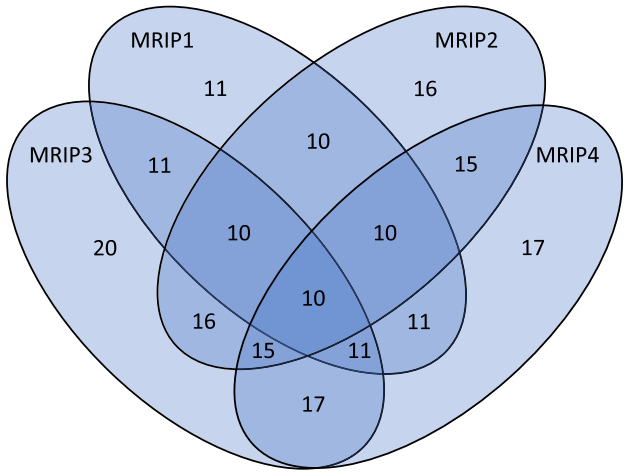

Fig. 5. Mutants killed per MRIP (out of 20).

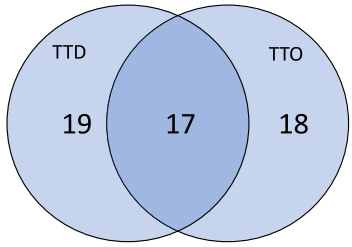

Fig. 6. Mutants killed per QoS metric (out of 20)

of mutants killed if the MRIPs are combined, and Fig. 6 shows the mutants killed by the MRs grouped by the QoS metric they use. The MS for each individual MR ranged from 15% to 90%, whereas the FDR ranged from 0.31% to 14.58%.

Comparing our approach with the baseline (i.e., thresholds-based oracle, see Section IV-C5), our MRs killed one more mutant than even the best possible thresholds. In this case, both approaches have the same cost, so our MRs can be considered more efficient in terms of MS. The mutant that could not be killed with the baseline approach and our evaluation test suite were detected by $MR2_{TTO}$, $MR3_{TTD}$, and $MR4_{TTD}$. On the other hand, one of the MRs did yield four FPs, while the baseline resulted in none. Furthermore, the baseline approach appears to obtain a better FDR than most MRs, although the top three MRs with the best MS do obtain comparable or better FDRs.

The analysis of the individual MRs shows that there is a great gap between their performances regarding the evaluation metrics we use. $MR3_{TTD}$ obtained both the highest MS (i.e., 90%) and the highest FDR (i.e., 14.58%), which makes it the most effective individual MR. On the other hand, MRIP2 seems to be the most effective MRIP when using the TTO metric, since the MS obtained by $MR4_{TTO}$ (i.e., 80%) is significantly higher than the best MS obtained by any of the other MRs, which uses TTO (i.e., 65%).

As for the MRs derived from MRIP1, their results are clearly inferior to the ones obtained by MRIP2, MRIP3, and MRIP4. They only accomplished an MS of up to 55%, and their FDRs are

also much lower on average. Taking a look at MRIP1 (faster vehicles), we generated three different variations, where the nominal speed increase was different (10%, 20%, and 30% faster), and we can appreciate small differences in the result. For the MRs based on TTO, a larger speed increase appears to slightly boost the MS and FDR obtained by the MR without resulting in any FPs. This makes sense, since a larger speed increase makes it harder for the navigation controller to maneuver the vehicle if the throttle is not properly adjusted, so excessive throttle and similar issues are easier to detect with higher nominal speeds. On the other hand, the effect of a larger speed increase is not so obvious for TTD. This may be because even if the controller makes some errors in handling the vehicle, the TTD may still improve or remain similar because the car is moving faster, so at higher speed increases, all but the most severe failures can be masked by the naturally smaller TTD when using this metric. Nevertheless, the effect on the results that different speed increases have is not very significant compared with the differences with the results from other MRIPs.

Ultimately, MRIP1, MRIP2, and MRIP4 seem to be redundant based on the results from this experiment, since MRIP3 alone can kill every mutant, and both of the MRs derived from this pattern also have the highest (i.e., 14.58%) and second highest (i.e., 11.98%) FDRs by a significant margin. $\mathrm{MR2}_{\mathrm{TTO}}$ is the only other MR with comparable results. There could be failure modes, which can only be detected with TTO, and for this metric, $\mathrm{MR2}_{\mathrm{TTO}}$ is the most effective MR in terms of MS.

D. Discussion

In what follows, we further explore the results from both case studies and what they tell us about the RQs.

1) RQ1: FPs: One of the main limitations of performance testing lies in the presence of FPs. The inherent nondeterminism of performance measurements, the inaccuracies of the simulators and the sensor readings and the approximate nature of some CPSs algorithms may lead to some violations of the MRs when there is no real observable failure, resulting in false alarms. To mitigate this, we used tolerance thresholds, under which MR violations were dismissed. Adjusting such thresholds required some preliminary work with the SUTs and the MRs. We observed, for example, that some MRs require more restrictive thresholds than others in order to avoid FPs, while others did not seem to require any tolerance threshold whatsoever. Finding the right balance is difficult; higher thresholds will result in fewer FPs, but it will also limit the failure-detection capability. In our work, we adopted a conservative approach, mostly prioritizing the removal of FPs over failure-detection to avoid engineers spending too much time on manual triage. Despite this, the MSs obtained, ranging between 88% and 100% make us confident in the feasibility of the approach. It is worth remarking that although adjusting the threshold requires some extra work, it is an upfront investment that should not need to be repeated once appropriate thresholds have been defined.

During our evaluation, we found three instances where the proposed MRs yielded FPs in the elevation case study and four of such instances in the autonomous driving system case study.

For the ones related to the elevator dispatcher, two out of three FPs actually revealed some abnormal behavior in the system. While reporting, both of these cases would have been desirable at some point of the development of the dispatching algorithm, they are currently scenarios that are recognized and dismissed by the domain experts during manual testing. Although the number of FPs appears to be manageable, aggregating the MR violations into specific issues, so that those already marked as "invalid" or "wontfix" can be automatically ignored, would be desirable in order to minimize manual checking. In order to achieve full automation in this case, the test failures could be classified based on the features of the test cases, similar to the approach for detecting flaky test failures proposed by Lampel et al. [30].

As for the remaining FP from the elevation case study and the four FPs from the autonomous driving system, they were a consequence of the MRs not being fully accurate. Such properties would ideally be implemented based on a statistical distribution from multiple test executions, i.e., statistical MT [23], rather than being checked over individual executions. However, this approach may not be feasible if the cost of test executions is very high, since collecting enough results for a meaningful statistical analysis might not be affordable.

In view of these results, we can answer RQ1 as follows.

> *RQ1:* Some of the MRs triggered FPs, but the number is manageable. Some of them could be avoided by classifying the MR violations and ignoring duplicates, while others may require more advanced statistical techniques in order to mitigate them.

2) RQ2: Effectiveness of Performance-Driven MT of CPSs: Our results show that performance of MT, and in particular MRs derived from the proposed PV pattern, are effective at detecting failures in CPSs (with MSs of 88.76% and 95% in our case studies), alleviating the oracle problem and enabling a high degree of automation. Although this approach can be relatively expensive, since MT requires multiple test executions for the oracle, the cost is still affordable, especially when the only alternative is manual testing. Furthermore, while the definition of the MRs usually requires domain knowledge, we show that very simple relations can still yield useful results. It is also noteworthy that once defined, MRs can be reused as long as the system specification does not change. Besides providing a fully automated oracle for cases where a regular oracle is not feasible, these MRs will be more resilient to hardware or configuration changes (e.g., increasing the nominal speed of the elevators), since the outputs of follow-up test cases are evaluated against those observed in the source test cases [49]. This last point is particularly important in the Elevator case study, where the SUT will be deployed into many installations with significantly different configurations.

It is also worth noting that the experiments presented in this article use random testing, which is the simplest and most naive approach. The use of more sophisticated technique for the source

test case generation could surely improve the cost-effectiveness of the presented approach (better FDR and MS).

As for the comparison with the baselines, the proposed approach beat them in both cases in terms of MS, showing that MT can identify failure modes that are difficult to detect with regular oracles. Nevertheless, some of the MRs did result in some FPs, whereas the baseline approaches had none. These FPs would result in some unnecessary efforts from the test engineers. Furthermore, comparing the FDRs from the autonomous driving system case study shows that the baseline oracles detect more failures than most MRs on the same test suite. Nevertheless, the MRs with the highest MS have similar or better FDRs than the baseline. Overall, the MRs appear to be more effective as long as the cost of tolerating some FPs is acceptable. Beyond the evaluation metrics, it is worth noting that the MRs are much more flexible regarding changes to the systems. For instance, if the nominal speed of the elevators or the autonomous vehicle were to change, the proposed MRs would still be valid as they are, whereas the baseline approaches would require new reference executions.

In view of these results, we can answer RQ2 as follows.

> *RQ2:* Performance MT, and in particular MRs derived from the proposed PV pattern, are effective at detecting nine out of every ten faults in CPSs, alleviating the oracle problem and achieving a high degree of automation. This approach shows clear benefits over automated regression test oracles.

3) RQ3: Differences in the Performance of the MRIPs and Performance Metrics: We observed significantly different performance among the proposed MRs in terms of failure-detection capability. In fact, we observed that some of the relations are largely subsumed by others. A similar observation was made when comparing the results of MRs grouped by MRIP or performance metric.

In the elevation case study, we found that one of the MRIPs obtained the best overall results by a great margin, whereas the rest could only make relatively modest contributions to failure detection. However, some of the other MRs also obtained outstanding results for specific evaluation metrics and are still able to complement the results from the best MRIP.

As for the autonomous navigation system, we found that one of the MRIPs completely dominated the others in our evaluation. Nevertheless, some of the MRs derived from the other MRIPs still achieved good results and might not be redundant for detecting some failure modes not considered in our evaluation, so keeping them would still be reasonable.

Generally, the best results seem to be obtained when combining specific MRIPs and performance metrics, and all of the performance metrics seemed to be able to achieve good results when combined with the right MRIP in both case studies, so none of them can be said to be useless. Finding the effectiveness of each MR and identifying redundant ones require an extensive evaluation of all of them, ideally performed with real test cases from the SUT, or otherwise by using techniques, such as mutation testing. Before such an evaluation is performed, in-line with the results in the field of MT [35], we advocate for defining *diverse* relations in terms of input changes and performance metrics.

In view of these results, we can answer RQ3 as follows.

> *RQ3:* Some MRIPs and QoS metrics perform significantly better than others. In-line with previous results in MT, MRs should be as diverse as possible.

V. Threats to Validity

In this section, we describe the sources of internal and external validity threats, which may have influenced our work, and how they have been mitigated.

A. Internal Validity

Internal validity threats are related to issues that might have affected the results of our evaluation. A potential threat for our experiments is that amount of mutants employed might have been too small. For the autonomous driving system, the amount of mutants we employ is similar to other studies where Simulink models are used [4], [34], [38]. As for the elevation case study, we have employed an even larger set of mutants, at the cost of approximately a month of execution time. Furthermore, we also checked for equivalent mutants, as recommended by Papadakis et al. [43], [44].

It is also worth noting that we employed manual fault seeding in order to generate the mutants. Unfortunately, the dispatcher needs to be compiled with a specific toolchain in order to make it compatible with the simulator, which prevented us from using existing mutation testing tools.

B. External Validity

The external validity threats are related to the generalizability of the results obtained from the experiments. In this work, we evaluate the application of the PV pattern in two case studies, which may not be enough to conclude its effectiveness for CPSs in general. Nevertheless, both of our case studies are highly complex systems, and they both have significantly different characteristics. Furthermore, the elevation case study employs a real-world industrial CPS, which is used in most of the multi-elevator installations deployed by Orona.

On the other hand, the manual step of defining effective MRs may be too complex for some types of systems, which might make this approach unfeasible in practice. In this work, we deliberately present and evaluate minimal MRs, which only consider a limited set of inputs and a single output metric in order to demonstrate that this approach can yield useful results in complex systems with relatively simple MRs.

VI. Related Work

A. Metamorphic Testing

In our previous work, we proposed the use of MRs based on domain-specific performance metrics to test multielevator systems [5], and we present experimental results with new

MRs for this case study in Section IV-B. Furthermore, we also presented a work in progress version of the the autonomous driving system experiment from Section IV-C [58].

Regarding autonomous vehicle systems, various MRs have already been applied to several types of vehicles. Lindvall et al. [33] proposed several MRs for model-based testing of the autonomous drone controller. In their approach, they employed input transformations similar to the ones we use for our autonomous driving system case study, such as altering the path of the vehicle in a way which should not affect the outcome or modifying the obstacles in the vehicle's path. As for self-driving cars, many approaches have employed input transformations, which simulate different driving conditions (e.g., clear day versus rain) in order to detect erroneous behavior [55], [66]. However, our work presents the novel approach of using output relations based on the performance metrics of the system, as opposed to checking the internal state or the outputs of the system.

An early precedent of performance-based MR can be found in [14], where the testing of a wireless sensor network application was performed by comparing the power consumption of multiple nodes for an equivalent computation. This MR is designed to detect bugs in the software, which may cause excessive power consumption, i.e., nonfunctional failures.

More recently, the concept of performance MT was presented in [49], where several MRs, which follow a similar pattern, were proposed to search for and identify nonfunctional failures on a system. In that work, they proposed MRs for general applications and web browsers based on the execution time, memory usage, and energy consumption of the test cases. In contrast, we propose the use of MRs following this pattern not only as a means to detect nonfunctional failures, but also in order to identify potential functional failures from the violation of these properties, and we apply this approach in the domain of autonomous driving and elevator control systems.

Performance MT has already been applied in the context of software testing. In [26], an MR based on the statistical distribution of page load times was used in order to discover a race condition in the Adobe Launch Tag Manager. However, our work is one of the first to apply such MRs in the domain of CPSs.

B. Testing CPSs

Testing is the main technique used by developers to verify that CPSs achieve an acceptable level of conformance and reliability. As a result, in the last few years, the scientific community has focused on devising novel techniques for automated and scalable CPSs testing, some of which aim to alleviate the test oracle problem.

Menghi et al. [39] proposed a method to generate online test oracles for Simulink models based on a set of properties expressed in signal temporal logic. Boufaied et al. [12] defined signal-based properties of CPSs, which can be used for the definition of test oracles. We have previously proposed the application of MT in the context of an industrial CPS in order to automatically test elevator dispatching algorithms [5]. This technique has also been used to test other CPSs, such as autonomous vehicles [66], but its application in this domain remains largely unexplored. In this work, we propose an MR pattern to facilitate the adoption of performance-based MT for CPSs.

Besides MT, an alternative approach to alleviate the test oracle problem is to employ machine learning techniques in order to predict the outputs or learn invariants of the CPS under test [10]. Chen et al. [18] employed traces from normal and abnormal (with seeded software faults) system executions in order to learn a support vector machine classifier able to detect anomalous behaviors in a water purification plant testbed. Shahamiri et al. [52] presented an approach to derive test oracles by using artificial neural networks (ANNs). This approach consists in training an ANN for every output of the system and using the predicted outputs as a reference to evaluate the real system outputs. The verdict from the ANN oracle is calculated as the mean squared error between the real and predicted outputs, which means that the oracle can calculate quantitative verdicts [52]. We also proposed the application of machine learning algorithms to alleviate the test oracle problem in the domain of elevation, both for functional [3] as well as nonfunctional faults [22]. Other domain-specific approaches based on machine learning have also been proposed, such as an unsupervised approach for autonomous vehicles testing [54]. Another approach for the autonomous vehicle domain is performing a human study to find the correlation between the quality metrics used in the domain and the human perception of driving quality, which can then be used to generate test oracles that approximate human oracles [24].

VII. Conclusion

In this article, we presented a performance-driven MT approach for CPSs. Specifically, we proposed a novel MR pattern, PV, which encourages testers to exploit input changes with a predictable impact in the system performance. In practice, the PV pattern eases the identification of performance MRs in CPSs, alleviating the test oracle problem. For the evaluation, we assessed the effectiveness of MRs derived from the PV pattern in detecting failures in an industrial elevator dispatcher and an open-source autonomous car by using seeded faults. Results showed that MRs derived from the PV pattern are effective in detecting 88.76% and 100% of the seeded faults, respectively, keeping the number of FPs at no more than 4%. The definition of the MRs and their implementation is a costly endeavour, but it pays off because the oracles are highly reusable. Potential lines of future work include, on the one hand, evaluating the cost-effectiveness of this approach when combined with more efficient test case generation, selection, or prioritization techniques, which would be more representative of its full potential. On the other hand, there are several aspects of this approach that could be further automated. One of them is the identification of the MRs themselves, which could be automated by defining generic templates for PV, similarly to how the approach in [57] identifies MRs for model transformations, or even generated by

an evolutionary algorithm based on a dataset of test executions labeled as correct or incorrect [6]. Another potentially automatable process is the fault localization for failures detected by the MRs, for which approaches based on metamorphic slices already exist [61].

References

[1] H. Agrawal et al., *Des. of mutant operators for the C program. lang.*, Softw. Eng. Res. Center, Dept. Comput. Sci., Purdue Univ., West Lafayette, IN, USA, Tech. Rep., 1989.

[2] J. Ahlgren et al., "Testing web enabled simulation at scale using metamorphic testing," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.: Softw. Eng. Pract.*, 2021, pp. 140–149.

[3] A. Arrieta, J. Ayerdi, M. Illarramendi, A. Agirre, G. Sagardui, and M. Arratibel, "Using machine learning to build test oracles: An industrial case study on elevators dispatching algorithms," in *Proc. IEEE/ACM Int. Conf. Automat. Softw. Test*, 2021, pp. 30–39.

[4] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, "Search-based test case prioritization for simulation-based testing of cyber-physical system product lines," *J. Syst. Softw.*, vol. 149, pp. 1–34, 2019.

[5] J. Ayerdi, S. Segura, A. Arrieta, G. Sagardui, and M. Arratibel, "Qos-aware metamorphic testing: An elevation case study," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 104–114.

[6] J. Ayerdi, V. Terragni, A. Arrieta, P. Tonella, G. Sagardui, and M. Arratibel, "Generating metamorphic relations for cyber-physical systems with genetic programming: An industrial case study," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1264–1274.

[7] J. Ayerdi, P. Valle, S. Segura, A. Arrieta, G. Sagardui, and M. Arratibel, "Replication package for the autonomous driving system," Accessed: Jul. 2022. [Online]. Available: https://github.com/pablovalle/MT-AutonomousVehicle

[8] R. Baheti and H. Gill, "Cyber-physical systems," *Impact Control Technol.*, vol. 12, no. 1, pp. 161–166, 2011.

[9] G. Barney and L. Al-Sharif, *Elevator Traffic Handbook: Theory and Practice*. Evanston, IL, USA: Routledge, 2015.

[10] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, May 2015.

[11] T. Nguyen et al., "Mutation operators for simulink models," in *Proc. 4th Int. Conf. Knowl. Syst. Eng.*, 2012, pp. 54–59.

[12] C. Boufaied, M. Jukss, D. Bianculli, L. C. Briand, and Y. I. Parache, "Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization," *J. Syst. Softw.*, vol. 174, 2021, Art. no. 110881.

[13] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "Leveraging metamorphic testing to automatically detect inconsistencies in code generator families," *Soft. Testing, Verification. Rel.*, vol. 30, no. 1, 2020, Art. no. e1721.

[14] W. K. Chan et al., "Towards the testing of power-aware software applications for wireless sensor networks," in *Proc. Int. Conf. Reliable Softw. Technol.*, 2007, pp. 84–99.

[15] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Dept. Comput. Sci., Hong Kong Univ. Sci. Technol., Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[16] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proc. 4th Ibero-Amer. Symp. Softw. Eng. Knowl. Eng.*, 2004, pp. 569–583.

[17] T. Y. Chen et al., "Metamorphic testing: A review of challenges and opportunities," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–27, Jan. 2018.

[18] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 648–660.

[19] Y. Dajsuren, M. G. J. van den Brand, A. Serebrenik, and S. Roubtsov, "Simulink models are also software: Modularity assessment," in *Proc. 9th Int. ACM Sigsoft Conf. Qual. Softw. Architectures*, 2013, pp. 99–106.

[20] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non-testable programs," in *Proc. ACM'81 Conf.*, 1981, pp. 254–257.

[21] A. F. Donaldson, "Metamorphic testing of android graphics drivers," in *Proc. IEEE/ACM 4th Int. Workshop Metamorphic Testing*, 2019, doi: 10.1109/MET.2019.00008.

[22] A. Gartziandia, A. Arrieta, A. Agirre, G. Sagardui, and M. Arratibel, "Using regression learners to predict performance problems on software updates: A case study on elevators dispatching algorithms," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, 2021, pp. 135–144.

[23] R. Guderlei and J. Mayer, "Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing," in *Proc. IEEE 7th Int. Conf. Qual. Softw.*, 2007, pp. 404–409.

[24] G. Jahangirova, A. Stocco, and P. Tonella, "Quality metrics and oracles for autonomous vehicles testing," in *Proc. IEEE 14th Int. Conf. Softw. Testing, Validation Verification*, 2021, pp. 194–204.

[25] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *Proc. IEEE 7th Int. Wirel. Commun. Mobile Comput. Conf.*, 2011, pp. 1666–1671.

[26] O. Johnston, D. Jarman, J. Berry, Z. Q. Zhou, and T. Y. Chen, "Metamorphic relations for detection of performance anomalies," in *Proc. IEEE/ACM 4th Int. Workshop Metamorphic Testing*, 2019, pp. 63–69.

[27] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 654–665.

[28] A. Kane, T. Fuhrman, and P. Koopman, "Monitor based oracles for cyber-physical system testing: Practical experience report," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2014, pp. 148–155.

[29] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Syst. J.*, vol. 9, no. 2, pp 350–365, Jun. 2015.

[30] J. Lampel, S. Just, S. Apel, and A. Zeller, "When life gives you oranges: Detecting and diagnosing intermittent job failures at Mozilla," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1381–1392.

[31] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. Cambridge, MA, USA: MIT Press, 2016.

[32] Peters Research Limited, "Elevate," Accessed: Jun. 2021. [Online]. Available: https://peters-research.com

[33] M. Lindvall, A. Porter, G. Magnusson, and C. Schulze, "Metamorphic model-based testing of autonomous systems," in *Proc. IEEE/ACM 2nd Int. Workshop Metamorphic Testing*, 2017, pp. 35–41.

[34] B. Liu, S. Nejati, and L. C. Briand, "Improving fault localization for Simulink models using search-based testing and prediction models," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reengineering*, 2017, pp. 359–370.

[35] H. Liu, F-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem," *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 4–22, Jan. 2014.

[36] MathWorks, "MATLAB/Simulink," Accessed: Jun. 2021. [Online]. Available: https://www.mathworks.com/products/simulink.html

[37] MathWorks Student Competitions Team, "Mathworks/vehicle-pure-pursuit," 2022, Accessed: Jul. 2022. [Online]. Available: https://github.com/mathworks/vehicle-pure-pursuit

[38] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Test generation and test prioritization for Simulink models with dynamic behavior," *IEEE Trans. Softw. Eng.*, vol. 45, no. 9, pp. 919–944, Sep. 2019.

[39] C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for Simulink models with continuous and uncertain behaviors," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 27–38 .

[40] C. Murphy, K. Shen, and G. Kaiser, "Automatic system testing of programs without test oracles," in *Proc. 8th Int. Symp. Softw. Testing Anal.*, 2009, pp. 189–200 .

[41] Orona, "Orona group," Accessed: Jul. 2022. [Online]. Available: https://www.orona-group.com/

[42] Open Source Modelica Consortium (OSMC), "Openmodelica," Accessed: Jul. 2022. [Online]. Available: https://www.openmodelica.org/

[43] M. Papadakis, C. Henard, M. Harman, Y. Jia, and Y. L. Traon, "Threats to the validity of mutation-based test assessment," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 354–365 .

[44] M. Papadakis, Y. Jia, M. Harman, and Y. L. Traon, "Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 936–946.

[45] S. Segura, "Metamorphic testing: Challenges ahead (keynote speech)," in *Proc. 3 rd Int. Workshop Metamorphic Testing*, 2018, pp. 1–4.

[46] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp 805–824, Sep. 2016.

[47] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of RESTful web APIs," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1083–1099, Nov. 2018.

[48] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Softw.*, vol. 37, no. 3, pp. 46–53, May/Jun. 2020.

[49] S. Segura, J. Troya, A. Durán, and A. Ruiz-Cortés, "Performance metamorphic testing: Motivation and challenges," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng.: New Ideas Emerg. Technol. Results Track*, 2017, pp. 7–10 .

[50] S. Segura, A. Durán, J. Troya, and A. Ruiz-Cortés, "Metamorphic relation patterns for query-based systems," in *Proc. IEEE/ACM 4th Int. Workshop Metamorphic Testing*, 2019, pp. 24–31.

[51] S. Segura, J. Troya, A. Durán, and A. Ruiz-Cortés, "Performance metamorphic testing: A proof of concept," *Inf. Softw. Technol.*, vol. 98, pp. 1–4, 2018.

[52] S. R. Shahamiri, W. M. N. Wan-Kadir, S. Ibrahim, and S. Z. M. Hashim, "Artificial neural networks as multi-networks automated test oracle," *Automated Softw. Eng.*, vol. 19, no. 3, pp. 303–334, 2012.

[53] N. Srinivas, N. Panditi, S. Schmidt, and R. Garrelfs, "Mil/sil/pil approach a new paradigm in model based development," 2022, Accessed: Jul. 2022. [Online]. Available: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/automotive/files/in-expo-2014/mil-sil-pil-a-new-paradigm-in-model-based-development.pdf

[54] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 359–371.

[55] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 303–314.

[56] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early," in *Proc. IEEE 11th Int. Conf. Softw. Testing, Verification Validation*, 2018, pp. 331–342.

[57] J. Troya, S. Segura, and A. Ruiz-Cortés, "Automated inference of likely metamorphic relations for model transformations," *J. Syst. Softw.*, vol. 136, pp. 188–208, 2018.

[58] P. Valle, "Metamorphic testing of autonomous vehicles: A case study on simulink," in *Proc. IEEE/ACM 43 rd Int. Conf. Softw. Eng.: Companion Proc.*, 2021, pp. 105–107.

[59] E. J. Weyuker, "On testing non-testable programs," *Comput. J.*, vol. 25, no. 4, pp 465–470, 1982.

[60] C. Wu, L. Sun, and Z. Q. Zhou, "The impact of a dot: Case studies of a noise metamorphic relation pattern," in *Proc. IEEE/ACM 4th Int. Workshop Metamorphic Testing*, 2019, pp. 17–23.

[61] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Inf. Softw. Technol.*, vol. 55, no. 5, pp. 866–879, 2013.

[62] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Testing, Verification Rel.*, vol. 22, no. 2, pp. 67–120, 2012.

[63] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, "Uncertainty-wise cyber-physical system test modeling," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1379–1418, 2019.

[64] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proc. 33 rd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2018, pp. 132–142.

[65] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Trans. Softw. Eng.*, vol. 46, no. 10, pp. 1120–1154, Oct. 2020.

[66] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Commun. ACM*, vol. 62, pp. 61–67, 2019.