
Afsoon Afzal

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Claire Le Goues (Chair), Carnegie Mellon University
Michael Hilton, Carnegie Mellon University
Eunsuk Kang, Carnegie Mellon University
John-Paul Ore, North Carolina State University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © 2020 Afsoon Afzal
**Keywords:** testing cyberphysical systems; robotics testing; automated quality assurance; simulation-based testing; challenges of testing; automated oracle inference;
Abstract

Robotics and cyberphysical systems are increasingly being deployed to settings where they are in frequent interaction with the public. Therefore, failures in these systems can be catastrophic by putting human lives in danger and causing extreme financial loss. Large-scale assessment of the quality of these systems before deployment can prevent these costly damages.

Because of the complexity and other special features of these systems, testing, and more specifically automated testing, faces challenges. In this thesis proposal, I study the unique challenges of testing robotics and cyberphysical systems by conducting a number of qualitative, quantitative, and mixed method studies, and propose an end-to-end automated testing pipeline to provide tools and methods that can help roboticists in large-scale automated testing of their systems. My key insight is that we can use (low-fidelity) simulation to automatically test robotic and cyberphysical systems, and identify many potentially catastrophic failures in advance at low cost.

My thesis statement is: Robotics and cyberphysical systems have unique features such as interacting with the physical world and integrating hardware and software components, which creates challenges for automated, large-scale testing approaches. Software-in-the-loop (low-fidelity) simulation can facilitate automated testing for these systems. Machine learning approaches (e.g., clustering) can be used to create an automated testing pipeline, which includes automated oracles and automated test input generation.

To support this statement, I propose the following work. In the preliminary work, which is already completed, I conducted a qualitative study and interviewed robotics practitioners about their testing practices and challenges. I identified nine main challenges roboticists face while testing their systems. In a case study on ARDUPILOT autonomous vehicle software, I investigated the potential impact of using low-fidelity software-based simulation on exposing failures in robotics systems, and showed that low-fidelity simulation can be an effective approach in detecting bugs and errors with low cost in robotic systems.

I propose to further study features in robotics simulators that are the most important for automated testing, and the challenges of using these simulators by conducting a large-scale survey with robotics practitioners.

I propose an approach to automatically generate oracles for cyberphysical systems using clustering, which can observe and identify common patterns of system behavior. These patterns can be used to distinguish erroneous behavior of the system and act as an oracle for an automated testing pipeline.

Finally, I propose to investigate automated test generation for these systems by first, identifying a suitable quality metric to evaluate the quality of test suites, and second, automatically generating test suites that target under-tested behaviors.
Contents

1 Introduction 1
  1.1 Thesis Statement ......................................................... 4
  1.2 Contributions ............................................................ 4
  1.3 Proposal Outline ........................................................... 4

2 Review of Literature and Background 5
  2.1 Related work ............................................................... 5
  2.2 Background ................................................................. 8

3 Challenges of Testing Robotic Systems 10
  3.1 Testing in robotics: practices and challenges ......................... 10
  3.2 Software-based simulation for testing .................................. 13
  3.3 Simulators for testing .................................................... 14

4 Automated Oracle Inference 16
  4.1 Background ................................................................. 16
  4.2 Approach ................................................................. 17
  4.3 Evaluation ................................................................. 19
  4.4 Limitations and future work ............................................. 20

5 Automated Test Generation 22
  5.1 Motivation ................................................................. 22
  5.2 Methodology ............................................................... 23

6 Proposed Timeline 26

7 Conclusion 28

Bibliography 29
1 Introduction

Robotic systems are systems that sense, process, and physically react to information from the real world [3]. In the past decade, robotic systems have become increasingly important in everyday humans’ lives. In the past, the use of these systems were mostly limited to industrial settings, in isolation and under specific safe conditions, which prevented potential extreme damages to people. However, robotic systems are now frequently used in a variety of (unsafe) settings such as avionics, transportation and medical operations. It is now more important than ever to ensure the safety and quality of these system before deploying them as failures in these systems can be catastrophic. In October 2018 and March 2019, two Boeing 737 MAX airplanes crashed due to planes’ aggressive dive caused by erroneous angle of attack sensor data, killing all 346 people abroad [1].

Automated quality assurance, or more specifically automated testing, is widely used in software development [24]. However, robotic systems have specific properties that make deployment of automated testing challenging in practice: 1) robots are comprised of hardware, software, and physical components, which can be unreliable and non-deterministic [43, 57, 73], 2) they interact with the physical world via inherently noisy sensors and actuators, and are sensitive to timing differences [73], 3) they operate within the practically boundless state space of reality, making emergent behaviors (i.e., corner cases) difficult to predict [43], and 4) the notion of correctness for these systems is often non-exact and difficult to specify [79]. As a result, in many cases the integration and system-as-a-whole testing is performed manually, mostly in the field [11]. Field testing is an important part of robotics development, but it could result in expensive and dangerous failures. In addition, field testing is highly limited by the scale of the scenarios and environments that it could be applied to. For example, testing an autonomous drone under highly windy conditions require either an expensive setup to artificially recreate high speed wind or waiting for the condition to naturally happen. None of these options are practical, which leaves some features untested in practice.

The ultimate goal of this work is to make robotics and cyberphysical systems safe, and improve the quality assurance of these systems. For this purpose, we first need to identify the challenges of testing commonly faced by robotics developers. Even though many studies have investigated the state of testing (especially automated testing) of software systems in practice [29, 41, 47, 88, 98], little attention has been paid towards automated testing of robotics and cyberphysical systems. As part of this proposed thesis, I conducted a qualitative study with robotics practitioners, to better understand the robotics testing practices currently being used, and identify challenges and bottlenecks preventing roboticists from automated testing [11]. I identified three themes of challenges in testing robotic systems: 1) real-world complexities, 2)
Figure 1.1: Automated testing pipeline for a cyberphysical system using simulation. An automated test input generation technique (Chapter 5) creates test inputs that will be executed on the CPS in simulation, which results in execution traces. These traces are provided to an automated oracle (Chapter 4) to be labeled as either correct or erroneous.

Even though hardware testing is an essential part of quality assurance for robotic systems [7, 104], it is (almost) orthogonal to the quality of operating software; if the behavior of the hardware and the environment can perfectly be simulated, we no longer need the actual hardware and a real environment to test the software of the system. However, to this day no simulator is able to perfectly simulate every aspect of the environment. In fact, most of the simulators provide low fidelity and are very limited [35, 36, 119]. In our qualitative study, we found that robotics practitioners find simulation ineffective and difficult to use [11]. In this thesis, I propose to conduct a quantitative study to inspect features and limitations of robotics simulators, and identify the most pressing issues that prevent or discourage developers from using robotic simulators for performing automated tests.

Although low-fidelity software-in-the-loop (SITL) simulators are not perfect, they can still be very effective tools in detecting and preventing failures by allowing cheap, large-scale, automated testing. The ExoMars lander crash that occurred in 2016 is an interesting example that shows potential effectiveness of simulation-based testing. The crash that cost approximately $350 million was later recreated in the simulation environment, which shows the role that simulation-based testing could play in preventing failures in the field [5]. In a similar case, a report issued by the National Transportation Safety Board (NTSB) on Boeing 737 MAX crashes illustrates that the specific failure modes that could lead to uncommanded plane dive (e.g., erroneous sensor data) were not properly simulated as part of functional hazard assessment validation tests, and as a result, were missed by NTSB’s safety assessment [2]. To illustrate the extent to which low-fidelity simulation-based testing may be used to detect failures in robotics systems, we conducted an empirical case study on a robotic system [107]. In this study, presented in Section 3.2, we showed that more than half of the bugs found in the system over time can manifest in low-fidelity SITL simulation. Specifically, we found that of all bugs, only 10% require particular environmental conditions (not available in low-fidelity simulation) to manifest, and 4% only manifest on physical hardware.

As a result, in absence of high-fidelity simulators, low-fidelity SITL simulation can be used for systematic, large-scale automated testing of CPSs, as its low fidelity only prevents discovering of a small number of bugs in the system. Figure 1.1 presents an automated testing pipeline of a CPS. In this thesis, I propose techniques to achieve this pipeline by creating an automated oracle (Chapter 4), and automated test input generation techniques (Chapter 5) that can effectively expose faulty system behavior of the system in simulation, before field testing. My proposed
techniques will not require pre-defined specifications or models of the system.

SITL simulation-based testing, as any other testing method, requires an oracle that can differentiate between correct and incorrect system behavior \[22\]. This oracle can take many forms, from formal specifications to manual inspection (human judgment) \[10, 63, 76, 78, 94, 122\]. For large-scale, automated testing, we require an oracle that can automatically label executions of the system and detect failures. However, manually defining such oracle for a robotics system requires extensive knowledge about the (usually very complex) system and considering all possible scenarios that the robot may face in advance \[82\]. For example, the oracle for a self-driving car may simply specify that the vehicle should not collide with any objects. Even though this oracle can detect failures in conditions where the vehicle hits a pedestrian or an object, it does not take into account cases where colliding with an object, such as a plastic bag in the air, does not impose any danger and should be allowed. In other words, robotic systems’ correct behavior may vary based on the conditions that are affected by the unpredictable environment, system’s configurations, timing and randomness. An accurate oracle needs to consider all possible conditions and specify the correct behavior of the system in those conditions \[69\].

An automated approach of generating the oracle can take us one step closer to a pipeline for systematic, large-scale automated testing presented in Figure \[1.1\]. Existing approaches in automated specification mining and invariant inference, formal verification and statistical models have tackled this problem in the past \[17, 25, 27, 37, 38, 39, 42, 44, 51, 58, 71, 72, 89, 90, 91, 115, 122\]. In Section \[2\], I describe these techniques in more details and discuss their advantages and limitations. Broadly, robotics and autonomous systems have features that limit the application of existing approaches. One such feature is the fact that many of these systems involve third-party components without access to the source code. This feature heavily limits the application of many existing tools as they require complete access to the source code to extract system’s specifications. Another feature of robotics systems as already mentioned involves context-dependent behavior of the system, which eventually requires a method that is able to learn disjunctive models, depending on context. Finally, the amount of noise and randomness in the environment in which these systems operate requires special attention.

My key insight is that by observing many executions of the system, as a blackbox, I can find patterns of correct or normal behavior of the system, and mark executions that do not comply with these patterns as abnormal or erroneous behavior. I propose techniques to generate these blackbox models from a set of previously observed executions of the system, and use them as an oracle that can differentiate between correct and erroneous behavior of the system. I evaluate the accuracy of these models by their ability to correctly label a set of traces. The details to this work is presented in Chapter \[4\].

Finally, studies have shown that the quality of test inputs highly affect exposure of faulty behaviors in testing \[60, 62, 92\]. An automated testing pipeline, as the one presented in Figure \[1.1\] requires an automated method of generating effective test inputs. In this work, I propose to investigate different methods of automated test generation, including (but not limited to) random and evolutionary test generation methods, which target corner cases and rarely observed behaviors to expose more bugs and failures. However, to be able to compare different test suites, I first need to identify a test suite quality metric that can measure the ability of the test suites to reveal faults in the system. I propose to study the performance of several test suite quality metrics (e.g., code and model coverage) on robotic and cyberphysical systems, and use the most suitable metrics to
develop and compare automated test suite generation methods.

1.1 Thesis Statement

Robotics and cyberphysical systems have unique features such as interacting with the physical world and integrating hardware and software components, which creates challenges for automated, large-scale testing approaches. Software-in-the-loop (low-fidelity) simulation can facilitate automated testing for these systems. Machine learning approaches (e.g., clustering) can be used to create an automated testing pipeline, which includes automated oracles and automated test input generation.

1.2 Contributions

This proposal contains a set of qualitative and quantitative studies, where I conduct interviews and surveys and use grounded theory to analyze the data. In addition, I conduct a case study on bugs in popular open-source ARDUPILOT system. I use the same ARDUPILOT system, in addition to possibly more cases, to evaluate performance of the proposed techniques both in terms of automatically creating more accurate oracles, and generating more fault revealing test inputs.

My thesis will contribute in the following ways:

1. It identifies the challenges of automated testing for robotics systems and discovers the practices currently being used in the field of robotics.
2. It shows that simulation-based testing can be an effective approach in identifying faults in these systems.
3. It identifies the challenges of using simulators for the purpose of (automated) testing, and the most prominent issues with currently available simulators.
4. It proposes a black-box approach to automatically infer oracles for these systems based on observed executions of the robot in the simulated environment.
5. It proposes methods to automatically generate tests that explore various system behaviors, specifically targeting rarely observed and under-tested behaviors to expose failures in rare conditions.

1.3 Proposal Outline

In this proposal I first provide an overview of literature on the topics related to this proposal, and background on related topics (Chapter 2). In the following chapters Chapter 3, 4 and 5, I identify three research thrusts and describe the preliminary work done for each thrust as well as the proposed work. In Chapter 6, I propose a timeline for completing the ongoing research and finally in Chapter 7, I conclude this thesis proposal.
2 Review of Literature and Background

The following sections give an overview of related work and background that inform the proposed work.

2.1 Related work

Robotic systems  Robots are systems that sense, process, and physically react to information from the real world [3]. Robotic systems are a subcategory of cyberphysical systems [66], which include non-robotics systems such as networking systems or power grids. However, robotic systems are subject to system constraints that do not apply to CPSs broadly (such as a need for autonomy, route planning, and mobility).

Robotic systems differ in several important dimensions [40, 43, 57, 73, 79, 101] as compared to conventional software: (1) Robots are comprised of hardware, software, and physical components, which can be unreliable and non-deterministic [43, 57, 73]. (2) Robots interact with the physical world via inherently noisy sensors and actuators, and are sensitive to timing differences [73]. (3) Robots operate within the practically boundless state space of reality, making emergent behaviors (i.e., corner cases) difficult to predict [43]. (4) For robotic systems, the notion of correctness is often non-exact and difficult to specify [79]. These characteristics introduce unique challenges for testing, such as the need to either abstract aspects of physical reality or conduct extensive testing in the real world.

Challenges of testing robotics and CPSs  A number of studies have investigated software testing practices broadly, and the challenges facing these practices [29, 41, 47, 88, 98]. Runeson [98] conducted a large-scale survey on unit testing with 19 software companies, and identified unit test definitions, strengths, and problems. Causevic et al. [29] qualitatively and quantitatively study practices and preferences on contemporary aspects of software testing.

In a technical report, Zheng et al. [118] report on a study of verification and validation in cyberphysical systems. The paper finds that there are significant research gaps in addressing verification and validation of CPS, and that these gaps potentially stand in the way of the construction of robust, reliable and resilient mission-critical CPS. The paper also finds that developers have a lack of trust in simulators, and one of the main research challenges they identify is integrated simulation. Seshia et al. [101]. introduce a combination of characteristics that define the challenges unique to the design automation of CPSs. Marijan et al. [79] speculate over a range of challenges involving testing of machine learning based systems.
Duan et al. [40] extract 27 challenges for verification of CPSs by performing a large-scale search on papers published from 2006 to 2018. Alami et al. [14] study the quality assurance practices of the Robot Operating System (ROS) community by using qualitative methods such as interviews with ten participants, virtual ethnography, and community reach-outs. They learn that implementation and execution of QA practices in the ROS community are influenced by social and cultural factors and are constrained by sustainability and complexity. However, their results only apply to a specific robotics framework and cannot be generalized to non-ROS systems.

Luckcuck et al. [77] systematically surveyed the state of the art in formal specification and verification for autonomous robotics, and identified the challenges of formally specifying and verifying (autonomous) robotic systems. Their study focuses on formal specification as a method of quality assurance and does not provide information regarding other testing practices within the wider field of robotics.

Wienke et al. [112] conducted a large-scale survey to find out which types of failures currently exist in robotics and intelligent systems, what their origins are, and how these systems are monitored and debugged. Sotiropoulos et al. [103] performed a study of 33 bugs in academic code for outdoor robot navigation. The study found that for many navigation bugs, only a low-fidelity simulation of the environment is necessary to reproduce the bug. Koopman and Wagner [69] highlight the challenges of creating an end-to-end process that integrates the safety concerns of a myriad of technical specialties into a unified approach. Beschastnikh et al. [26] looked at several key features and debugging challenges that differentiate distributed systems from other kinds of software.

Anomaly and intrusion detection There are a number of studies on anomaly detection in cyberphysical systems [32, 48, 53, 56, 86, 93, 110, 122]. He et al. [55] propose an approach for creating autoregressive system identification (AR-SI) oracles for CPSs. Based on the assumption that many CPSs are designed to run smoothly when noises are under control, AR-SI automatically determines whether a trace is erroneous or correct by checking the smoothness of the system’s behavior. Theisslet et al. [105] propose an approach that reports anomalies in the multivariate time series which point the expert to potential faults.

Chen et al. [30] build models by combining mutation testing and machine learning: they generate faulty versions (mutants) of the tested system and then learn SVM-based models using supervised learning over the resultant data traces corresponding to system execution. They evaluate on a model of a physical water sanitation plant. Ghafouri et al. [49] show that common supervised approaches in this context are vulnerable to stealthy attacks. An unsupervised technique [59] evaluated on the same treatment plant model trains a Deep Neural Net (DNN) to identify outliers. Ye et al. [117] use a multivariate quality control technique to detect intrusions by building a long-term profile of normal activities in information systems and using the norm profile to detect anomalies.

Other approaches target the detection of particular attack classes specifically. Choi et al. [33] present a technique that infers control invariants to identify external physical attacks against robotic vehicles. Alippi et al. [18] learn Hidden Markov Models of highly correlated sensor data that are then used to find sensor faults. Abbaspour et al. [6] train adaptive neural networks over

[https://ros.org]
Hutchison et al. [57] outline a framework for automated robustness testing of autonomy systems. Abdelgawad et al. [8] introduce a systematic model-based testing approach to evaluate the functionality and performance of a real-time adaptive motion planning system. Menghi et al. [84] propose an automated approach to translate CPS requirements specified in a logic-based language into test oracles specified in Simulink’s simulation language for CPS.

The oracle problem  Fully automated testing for CPSs requires oracles that can determine whether a given CPS behaves correctly for a given set of inputs [22]. In typical research and practice, domain experts manually provide CPS oracles in the form of a set of partial specifications, or assertions [10, 63, 76, 78, 94, 122]. However, manually writing such specifications is tedious, complex, and error-prone [50, 82].

A number of techniques proposed approaches for inferring invariants or finite state models describing correct software behavior perform what is known as dynamic specification mining. Existing dynamic specification mining techniques can be classified into four categories based on the kind of models that they produce: data properties (a.k.a. invariants) [37, 42, 51, 89, 90], temporal event properties [25, 27, 72, 115], timing properties [91, 100], and hybrid models [17, 71, 91] that combine multiple types of model. These techniques are generally poorly-suited to the CPS context. Most require source code access or instrumentation, and none are suitable for time series data. Techniques like Daikon [42] and its numerous successors (e.g., DySy [57], SymInfer [90], Nguyen et al. [89], or Dinv [51], among others) learn source- or method-level data invariants rather than models of correct execution behavior. Techniques like Texada [72] and Perracotta [115] do learn temporal properties between events but do not model or learn temporal data properties, a key primitive in CPS execution (Artinali [17] comes closest to this goal, learning event ordering and data properties within an event).

As another way of approaching the oracle problem for CPSs, studies have used metamorphic testing to observe the relations between the inputs and outputs of multiple executions of a CPS [74, 106, 120]. Lindvall at al. [74] exploit tests with same expected output according to a given model to test autonomous systems. Zhou and Sun [120] use metamorphic testing to specifically detect software errors from the LiDAR sensor of autonomous vehicles. Tian et al. [106] introduce DeepTest, a testing tool for automatically detecting erroneous behaviors of DNN-driven vehicles. As an oracle, they use metamorphic testing by checking that properties like steering angle of an autonomous vehicle remain unchanged in different conditions such as different weather or lighting.

Automated Test generation  Software test automation significantly improves the quality, and automated test suite generation significantly affects the software test automation, and is a very integral part of the automation process [67]. Automated test suite generation for CPSs includes numerous model-based approaches [80, 85, 113]. These approaches require a model of the system in a particular format (e.g., Simulink or MATLAB models) to generate a set of test cases that reach the highest coverage of the model. In the absence of such models, search-based techniques have shown promise in automated test suite generation of CPSs [50, 54, 108, 109].

Search-Based Software Testing (SBST) is a method for automated test generation based on
optimization using meta-heuristics \cite{15, 81}. The SBST approaches require a fitness function that has crucial impact on their performance \cite{15, 99}. In a study on java programs, Salahirad et al. \cite{99} showed that fitness functions that thoroughly explore system structure should be used as primary generation objectives, supported by secondary fitness functions that explore orthogonal, supporting scenarios. Fraser et al. \cite{45} propose a novel paradigm in which whole test suites are evolved with the aim of covering all coverage goals at the same time while keeping the total size as small as possible.

Arrieta et al. \cite{19} propose a search-based approach that aims to cost-effectively optimize the test process of CPS product lines by prioritizing the test cases that are executed in specific products at different test levels. By applying SBST to automated driving controls, Gladisch et al. \cite{50} show that SBST is capable of finding relevant errors and provide valuable feedback to the developers, but requires tool support for writing specifications. Bagschik et al. \cite{20} propose a generation of traffic scenes in natural language as a basis for a scenario creation for automated vehicles. Similarly, Gambi et al. \cite{46} recreate real car crashes as physically accurate simulations in an environment that can be used for testing self-driving car software.

To take uncertainty that is unavoidable in the behaviors of CPSs into consideration at various testing phases, including test generation, Ali et al. \cite{16} propose uncertainty-wise testing, arguing that uncertainty (i.e., lack of knowledge) in the behavior of a CPS, its operating environment, and in their interactions must be explicitly considered during the testing phase.

**Mutation Testing** Mutation testing is a fault-based testing technique which provides a testing criterion called the *mutation score*. The mutation score can be used to measure the effectiveness of a test set in terms of its ability to detect faults \cite{60}. Achieving higher mutation scores improves the fault detection significantly \cite{92}. On safety-critical systems mutation testing could be effective where traditional structural coverage analysis and manual peer review have failed \cite{21}.

## 2.2 Background

**ArduPilot** The open-source ARDUPILOT project\footnote{http://ardupilot.org} written in C++, uses a common framework and collection of libraries to implement a set of general-purpose autopilot systems for use with a variety of vehicles, including, but not limited to, submarines, helicopters, multirotors, and airplanes. ARDUPILOT is extremely popular with hobbyists and professionals alike. It is installed in over one million vehicles worldwide and used by organizations including NASA, Intel, and Boeing, as well as many higher-education institutes around the world\footnote{http://ardupilot.org/about}.

I use ARDUPILOT as one of my case studies due to being highly popular and open-source, and its rich version-control history, containing over 30,000 commits since May 2010, and for its consistent bug-fix commit description conventions. ARDUPILOT has been widely used in studies on CPSs as it represents a fairly complex open-source CPS \cite{9, 55, 74, 111, 122}, and contains 300,000 lines of code (measured using SLOC).

To facilitate rapid prototyping and reduce the costs of whole-system testing, ARDUPILOT offers a number of simulators for most of its vehicles (excluding submarines). In general, those
platforms simulate the dynamics of the vehicle under test, feed artificial sensor values to the controller, and relay the state of its actuators to the physics simulation. Hardware-in-the-loop (HIL) simulators are used to perform testing on a given flight controller hardware device by directly reading from and writing to it. In contrast, software-in-the-loop (SITL) simulators test a software implementation of the flight controller by running it on a general-purpose computer.

**Robot Operating System (ROS)**  The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. In ROS, nodes are processes that perform computation, and they communicate with each other by passing messages. A node sends a message by publishing it to a given topic, which is simply a string such as “odometry” or “map”. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others’ existence.

ROS is a relatively young framework (first released in 2009), and is currently used by thousands of people around the world; ROS has more than 34,000 registered users on ROSanswers, the main Q&A platform for ROS users. It follows an annual release model that is both similar to and linked to Ubuntu, and to this day, there have been 12 official, released ROS distributions (e.g., Lunar, Kinetic, and Jade). ROS is designed with the purpose of encouraging collaborative robotics software development by allowing robotics developers to build upon each other’s work.

---

[https://ros.org](https://ros.org)

3 Challenges of Testing Robotic Systems

This section describes parts of the completed work of this thesis proposal, as well as proposed work. As described in Chapter 1, robotics and cyberphysical systems have features such as non-deterministic behavior and noisy sensors that make them different from conventional software systems. These features can specifically create challenges for automated testing approaches. Even though many studies have focused on understanding the challenges of testing in conventional software systems, to our knowledge, no such study has been done on robotics.

To better understand the state of (automated) testing in robotics, my collaborators and I first conducted a series of qualitative and quantitative studies. We first identified challenges of (automated) testing in the field of robotics, and testing practices currently being used in this field by conducting a series of qualitative studies with robotics practitioners. In this study, we identified 9 challenges that robotics practitioners face while testing their robotic systems.

Secondly, we investigated the potential impact of using low-fidelity software-based simulation on exposing failures in robotics systems by conducting a case study. In this study, we showed that low-fidelity simulation can be an effective approach in detecting bugs and errors with low cost in robotic systems.

Finally, as features of SITL simulators highly impact the automated testing of these systems, I propose to conduct a large-scale survey to identify features in robotics simulators that are the most important for automated testing, and the challenges of using these simulators. The results of this study can encourage simulator designers and developers to prioritize features that make a simulator more suitable for automated testing of robotic systems.

3.1 Testing in robotics: practices and challenges

As described in Chapter 2, unique features of robotics and cyberphysical systems such as interaction with the real world through noisy sensors and actuators, introduce challenges to automated testing and validation of these systems. However, no prior studies have identified testing practices and challenges for robotic systems. Our goal in this study is to gain an in-depth understanding of existing testing practices and challenges within the robotics industry. We conducted a series of semi-structured interviews with robotics practitioners from a diverse set of companies. The result of this study is published in the International Conference on Software Testing, Verification and Validation (ICST 2020) [11].

Interviews are useful instruments for getting the story behind a participant’s experiences, acquiring in-depth information on a topic, and soliciting unexpected types of information [83, 102]. We developed our interview script by performing a series of iterative pilots.
We recruited our participants through a variety of means. Our goal was to select participants from a broad range of positions and to sample across a diversity of industries, company size, and experience. We interviewed 12 robotics practitioners with a variety of backgrounds and experiences, representing 11 robotics companies and institutions ranging from small startups to large multi-national companies. Our participants cover a variety of educational backgrounds such as computer science, mechanical engineering, math and physics, and their institutional role includes developer, test engineer, project manager, and CTO.

**Interviews and coding**  We conducted semi-structured interviews that lasted between 30 to 60 minutes over phone, video chat, or in person. We prepared an interview script with detailed questions providing insight into our research questions. A subset of questions on the interview script are presented in Table 3.1. However, we only used the script to guide the interviews. We adjusted interview questions based on the experience of the participant to gain a deeper understanding of their testing practices and challenges. We took notes from interviewee responses and recorded the interviews with their consent to validate our notes. We then used a grounded, iterative approach to code our notes. We first labeled responses based on their relevance to our research questions. Then, we iteratively coded the notes based on common themes, discussed the codes and redefined them.

**Validation** To validate the results of our study and conclusions, we sent a full draft of the results to our participants. We asked participants to inform us of their level of agreement with our conclusions and to provide their thoughts on our results. In total, six of the participants responded to our request. Four responded in total agreement with the results. The other two participants that responded provided specific feedback on our interpretation of their responses, and we incorporated their feedback into the final version of this paper.

---

Table 3.1: Sample questions on the interview script.

<table>
<thead>
<tr>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What are all the different types of testing you do?</td>
</tr>
<tr>
<td>• Can you describe your test running/writing process?</td>
</tr>
<tr>
<td>• How much of your testing is done for certification?</td>
</tr>
<tr>
<td>• Which types of tests find the most problems?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What is difficult about writing tests?</td>
</tr>
<tr>
<td>• Have these difficulties ever made you giving up on writing the tests at all?</td>
</tr>
<tr>
<td>• Is there any part of writing tests that is not difficult?</td>
</tr>
<tr>
<td>• What types of tests do you have the most difficulty running?</td>
</tr>
<tr>
<td>• In your experience, is there anything that helps with making it easier to run tests?</td>
</tr>
<tr>
<td>• For your tests that are not fully automated, why are they not?</td>
</tr>
<tr>
<td>• What tools/frameworks/techniques do you use to simplify running tests?</td>
</tr>
<tr>
<td>• Do you use simulation?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What do you think is the most important bottleneck in the way of testing in robotics?</td>
</tr>
<tr>
<td>• How do you think the difficulties of testing in robotics differ from your other experiences in other software development domains?</td>
</tr>
</tbody>
</table>
Table 3.2: A summary of the challenges of designing, running, and automating tests for robotics that we identified based on participant responses.

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Unpredictable corner cases</td>
<td>The challenge of attempting to anticipate and cover for all possible edge cases within a large (and possibly unbounded) state space when designing tests.</td>
</tr>
<tr>
<td>C2</td>
<td>Engineering complexity</td>
<td>A disproportionate level of engineering effort is required to build and maintain end-to-end test harnesses for robotic systems with respect to the benefit of those tests.</td>
</tr>
<tr>
<td>C3</td>
<td>Culture of testing</td>
<td>The challenge of operating within a culture that places little value on testing and provides developers with few incentives to write tests.</td>
</tr>
<tr>
<td>C4</td>
<td>Coordination, collaboration, and documentation</td>
<td>A lack of proper channels for coordination and collaboration among multiple teams (especially software and hardware teams), and a lack of documentation.</td>
</tr>
<tr>
<td>C5</td>
<td>Cost and resources</td>
<td>The cost of running and automating the tests in terms of human hours, resources and setup, and running time.</td>
</tr>
<tr>
<td>C6</td>
<td>Environmental complexity</td>
<td>The inherent difficulties of attempting to account for the complexities of the real world when simulating, testing, and reproducing full-system behavior.</td>
</tr>
<tr>
<td>C7</td>
<td>Lack of oracle</td>
<td>The challenge of specifying an oracle that can automatically distinguish between correct and incorrect behavior.</td>
</tr>
<tr>
<td>C8</td>
<td>Software and hardware integration</td>
<td>Difficulties that arise when different software and hardware components of the system are integrated and tested.</td>
</tr>
<tr>
<td>C9</td>
<td>Distrust of simulation</td>
<td>A lack of confidence in the accuracy and validity of results obtained by testing in simulation and synthetic environments, and a sole reliance on field testing.</td>
</tr>
</tbody>
</table>

Results  After iterative coding of the interviews, we identified 12 testing practices that are mentioned by our participants. These practices include a wide range of manual and automated testing approaches such as unit testing for automated testing of small individual code-level software components, and manual field testing of the system in a real-world environment that shares similarities with deployed environment.

In addition, we identified 9 challenges that roboticists face while designing, running, and automating tests for their systems summarized in Table 3.2. Many of these challenges such as unpredictable corner cases and lack of oracle are well-known challenges of testing systems in other fields as well. However, features of robotic systems such as interacting with real world through sensors make these challenges even harder to resolve. We identified 3 major themes
among the identified challenges: 1) real-world complexities, 2) community and standards, and 3) component integration. We supported these themes by showing quotes from our participants, and later speculated on the implications of each theme and provided suggestions for tackling their associated challenges.

One of the contributions of this study is identifying the challenges of testing created by real-world complexities. Software-based simulation could be a promising approach towards abstracting away a number of these complexities and prepare an opportunity to conduct large-scale tests on the system. However, as pointed out by C9, robotics practitioners generally distrust the results of executions in simulation because of the low fidelity of the simulated environment and too much abstractions of the real world.

3.2 Software-based simulation for testing

As described in Section 3.1, robotics practitioners generally distrust low fidelity simulation, and find it ineffective in exposing robotics bugs. To evaluate the validity of this common belief, we conducted a case study on ARDUPILOT system (presented in Section 2.2). This study is published in the International Conference on Software Testing, Verification and Validation (ICST 2018) [107].

We first identified potential bug-fixing commits within the version-control history of the ARDUPILOT project using a number of automated filters on the edited file types and keywords in the commit message. We manually inspected all 333 commits that passed this initial filtering stage and identified 228 bug-fixing commits.

After reaching a consensus on the list of bug-fixing commits, we manually inspected each commit to determine whether the bug can be reproduced in simulation, and if so, what are the requirements for triggering and detecting it. We labeled each historic bug based on the following 7 questions:

1. Does triggering or observing the bug rely on physical hardware?
2. Is the bug only triggered when handling concurrent events?
3. Which kinds of input are required to trigger the bug?
4. At which stage in the execution does the bug manifest?
5. Is the bug only triggered under certain configurations?
6. Is the bug only triggered in the presence of certain environmental factors (e.g., wind, human presence)?
7. How does the bug affect the behavior of the system?

Responding to these questions allows us to realize whether it is possible to trigger and manifest these bugs in low-fidelity SITL simulation. Requiring a physical hardware, or presence of certain environmental factors, for instance, makes triggering and manifesting a bug in low-fidelity simulation extremely difficult (if not impossible).

**Results** Our results, based on an empirical study of historical bugs in real-world robotics system, dispute the common belief among robotics practitioners that the fidelity of simulators are
not sufficient enough to catch most of the bugs in robotics systems. We discovered that, contrary to our expectations, the majority of bugs can indeed be reproduced using SITL simulation approaches without the need for complex triggering mechanisms (e.g., environmental conditions, concurrent events, component failures). We found that only a small minority—approximately 10%—of bugs are dependent on particular environmental factors. Below, we discuss the findings of our analysis in terms of the proposed questions:

1. In total, only 10 of 228 bugs relied on the presence of physical hardware for their detection or observation.
2. We determined that only 13 out of 228 bugs (5%) require concurrent events in order to be triggered.
3. Mimicking continuous radio-controller (RC) inputs, usually provided by a human-operator, is significantly more challenging than supplying the system with discrete, well-formed inputs (i.e., ground control system (GCS) commands, and preprogrammed missions). Encouragingly, our findings show that 165 of the 212 bugs do not rely on continuous input.
4. We observed that almost 80% of bugs (184 out of 228) occur during the normal operation of the robot.
5. We determined that 81 of 228 bugs depend on either a particular static (53) or dynamic (20) configuration, or a combination of both (8).
6. To our surprise, once again, we discovered that only 22 of 228 bugs depend on environmental factors.
7. We found that 205 bugs resulted in observable, behavioral changes to the program, and the rest result in corruption of log files or system crash.

The findings of our study strongly support the idea of applying cheap, simulated-based testing approaches to the problem of detecting bugs in robotics systems. However, we also found that continuous events, in the form of radio-controller inputs, and specific configurations are required to trigger a large number of bugs. We believe that both of these challenges, whilst difficult, can be overcome by developing specialized testing methods and leveraging and building upon existing knowledge in, e.g., testing of highly configurable systems [64].

### 3.3 Simulators for testing

Our case study on ARDUPILOT system described in Section 3.2 showed that low-fidelity SITL simulation can be used for systematic, large-scale automated testing of CPSs to discover a large number of bugs in the system, and can take us one step closer to the automated testing pipeline in Figure 1.1.

However, in the qualitative study presented in Section 3.1, our participants referred to a number of challenges of using software-based simulators, and discussed them in depth. These challenges included the low-fidelity of simulators, their hard to use interface, and the cost and effort required to set them up for a particular system.

I propose to conduct a large-scale survey with robotics practitioners to explore challenges that prevent developers from using the SITL simulators for automated testing, and identify simulator
features that are the most useful for automated testing. In this study, I will design a survey mainly focusing on the following questions:

1. Do robotics developers perceive inherent value in simulation-based testing?
2. What are the most common problems associated with simulation-based testing?
3. How could software-based simulation platforms be improved to encourage developers to use simulation for software-based testing?
4. Are developers using software-based simulation for automated testing?
5. What prevents developers from using software-based simulation to automate their testing?

The questions on the survey will include both multiple choice and open-ended questions. To analyze the open-ended questions, I will use an iterative coding approach to summarize common themes of responses. I will distribute this survey among robotics practitioners through multiple platforms including Twitter, Reddit, and robotics forums and mailing lists.

The results of this study can encourage simulator developers and designers to implement and enhance features in SITL simulators that improve application of automated testing.
4 Automated Oracle Inference

This section describes parts of the completed work of this thesis proposal. As presented in Figure 1.1, an oracle that can automatically distinguish between correct and incorrect behaviors of the system is essential to achieve an automated testing pipeline. However, generating an oracle is a well-known problem in software engineering [22], and as presented in Section 3.1 is one of the challenges of testing robotics systems. Although many approaches have been proposed to address the oracle problem for pure software (cyber) systems [10, 63, 76, 78, 94, 122], they are not appropriate for robotics and cyberphysical systems (CPSs) for the following reasons:

1. CPSs often contain proprietary third-party components (such as cameras or other sensors) for which source code is unavailable, and so techniques should minimize or avoid relying on source code access.

2. CPSs are inherently non-deterministic due to noise in both their physical (e.g., sensors, actuators, feedback loops) and cyber components (e.g., timing, thread interleaving, random algorithms) and may react to a given command in a potentially infinite number of subtly different ways that are considered to be acceptable. That is, for a given input and operating environment, there is no single, discrete response that is correct, but rather an envelope of responses that are deemed correct. And so techniques should be robust to small, inherent deviations in behavior.

3. The CPS may respond differently to a given instruction based on its environment, configuration, and other factors (i.e., its operating context). For example, a flying copter may refuse to fly to the specified point if its battery is depleted. And so, techniques must be capable of capturing contextual behaviors for a given command.

To tackle the above mentioned challenges, I propose Mithra: a novel oracle learning approach, which identifies patterns of normal (common) behaviors of the system by applying a multi-step clustering approach to the telemetry logs collected over many executions of the system in simulation. Mithra uses the identified clusters as the core of its oracle and determines correctness of system’s executions based on their similarity to identified clusters. It tackles all three above mentioned challenges as it does not require source code access, avoids over generalization and is robust to small deviations from expected behavior, and identifies contextual behaviors.

4.1 Background

Statistical machine learning approaches deal with the problem of finding a predictive function based on data. These approaches are applied on a collection of instances of the data, which acts
as the input to the learning process (i.e., training data). What the algorithm can learn from the training data varies in different approaches [121].

Supervised learning methods take a collection of training data with given labels (e.g., “male” and “female”), and learn a predictive model \( y = f(x) \), which can predict the label \( y \) of a given input \( x \). Depending on the domain of label \( y \), supervised learning problems are further divided into classification and regression. Classification is the supervised learning problem with discrete classes of labels, while regression is applied on continuous labels. Support vector machines (SVM), decision trees, linear and logistic regressions, and neural networks are all examples of supervised learning algorithms [70].

Unsupervised learning techniques work on an unlabeled training data. Common unsupervised learning tasks include: (1) clustering, where the goal is to separate the n instances into groups, (2) novelty detection, which identifies the few instances that are very different from the majority, and (3) dimensionality reduction, which aims to represent each instance with a lower dimensional feature vector while maintaining key characteristics of the training sample. Clustering approaches such as \( k \)-means [75], \( k \)-medoids [65] and hierarchical clustering [61], in general split the training data into \( k \) clusters, such that instances in the same cluster are similar, according to a similarity measure, and instances in different clusters are dissimilar. The number of clusters \( k \) may be specified by the user, or may be inferred from the training sample itself [114].

The distance measure selected for learning approaches highly impacts their performance, and depends on the type of data, the goal of learning, and the distribution of the data. A common distance metric is Euclidean distance (i.e., L2 norm) between two datapoints, which is inexpensive to compute. In this chapter, we use Dynamic Time Warping [23] and Eros [116] to measure similarity between two time-series data based on their shape.

4.2 Approach

Given a CPS that accepts a vocabulary of discrete commands and produces a log of its telemetry (e.g., ArduCopter), the goal is to learn a set of contextual behaviors for each of those commands from a training set of telemetry logs. To do so, we created Mithra, which extracts the set of relevant execution traces for each command from a set of telemetry logs, and applies a novel, three-step clustering process to those traces. Using its learned behavioral clusters, Mithra constructs a classifier for each command that marks execution traces as either CORRECT or ERRONEOUS based upon their similarity to the contextual behaviors represented by those clusters.

Mithra applies time series clustering to a training set of execution traces collected from many normal command executions for a CPS to learn the set of unique, contextual behaviors for those commands. These traces can be collected by executing different scenarios or missions in simulated environment or in real-world execution, and should cover a diverse set of system behaviors.

Studies have shown that approaches for clustering and classifying time series that are based on comparing differences in shape are often superior in terms of performance than those that compare differences in time [13, 96]. As such, Mithra clusters execution traces based on their overall shape using an approach inspired by three-step clustering of large time series datasets [12]. Figure 4.1 provides a high-level overview of Mithra’s clustering approach. During the first step (preclustering), Mithra computes a set of initial clusters, referred to as preclusters, based on a
Figure 4.1: An overview of Mithra’s three-step clustering approach of preclustering, subclustering, and merging. Solid lines represent individual traces, and dashed lines represent cluster centroids.

lower-fidelity approximation of trace similarity. In the second step (purifying), each precluster is split into a smaller set of subclusters based on the Eros similarity of traces within that precluster. In the final step (merging), Mithra creates the set of behavioral clusters by merging subclusters with highly similar centroids based on their dynamic time warping (DTW) distance. Finally, Mithra computes $\mu_\beta$ and $\sigma_\beta$ for each behavioral cluster $\beta$ based on the distance from the traces within $\beta$ to the centroid of that cluster, which Mithra uses to construct the decision boundary for $\beta$. Every step in this approach is designed to improve the overall accuracy of the behavioral clusters while keeping the approach scalable to a large number of traces.

The behavioral clusters detected by Mithra for each command are representative of the qualitatively different modes of behavior observed for that command. These behaviors include both behaviors that are frequently observed and assumed to be correct (e.g., clusters with more than one hundred traces) and behaviors that are rarely observed and suspected to be erroneous (e.g., clusters with fewer than five traces).

Given a previously unseen execution trace $\tau$ for a command, Mithra first finds the behavioral cluster $\beta_\tau^* \in BC$ that most closely resembles $\tau$ based on the DTW distance between $\tau$ and the centroid of each cluster:

$$\beta_\tau^* = \arg \min_{\beta \in BC} DTW(\tau, c_\beta)$$

Mithra then uses $\beta_\tau^*$ to predict the label $\ell_\tau$ for that trace as:
\[ \ell_{\tau} = \begin{cases} 
\text{ERRONEOUS} & \text{if } |\beta_{\tau}^*| < \rho \\
\text{ERRONEOUS} & \text{if } DTW(\tau, c_{\beta_{\tau}^*}) > \mu_{\beta_{\tau}^*} + \theta_{\beta_{\tau}^*} \\
\text{CORRECT} & \text{otherwise} \end{cases} \]

where \(|\beta|\) is the number of traces within \(\beta\), \(\rho \in \mathbb{Z}^+\) is the rarity threshold, and \(\theta \in \mathbb{R}^+\) is the acceptance rate. If \(\beta_{\tau}^*\) contains fewer than \(\rho\) traces, it is assumed to represent a rare and thus, erroneous behavior, and so, \(\tau\) is marked as ERRONEOUS. In the more likely case where \(\beta_{\tau}^*\) contains at least \(\rho\) traces, then \(\beta_{\tau}^*\) itself is assumed to represent a common and thus, correct behavior. In that case, Mithra uses the precomputed DTW distance to determine whether \(\tau\) lies within the decision boundary of \(\beta_{\tau}^*\), and if so, labels it as CORRECT. The acceptance rate \(\theta\) is used to alter the extent of the decision boundary and provides the user with a means of controlling the precision-recall tradeoff of the classifier to their preferences.

4.3 Evaluation

We evaluated Mithra on ArduCopter system described in Section 2.2, which has been widely used in studies on CPSs as it represents a fairly complex open-source CPS. Evaluating Mithra on a CPS requires a set of traces to be used as training data, and a set of ground truth ERRONEOUS and CORRECT traces to be used as evaluation (test) data. As a source of training data for Mithra, we recorded traces for 2500 randomly generated missions in simulation. To provide idempotency, we use Docker\(^1\) to run each mission in simulation in its own sandboxed container.

As the source of evaluation data, we obtained 24 bugs for the ArduPilot system: 11 of which are real-life bugs, and 13 are inspired by real-life bugs. For each bug scenario, we used a handwritten mission template, tailored to that scenario, to randomly generate 10 missions that trigger and manifest the bug. After running each mission, we collect line coverage of the execution to ensure that the executed mission does in fact execute the lines of interest (i.e., faulty lines). Finally, we use the generated missions to construct a balanced evaluation set of correct and erroneous traces.

We publicly release our dataset of real-world ArduPilot bugs, mission templates, and generated traces to allow others to reproduce and extend our experiments. Additionally, we believe that this dataset can serve as a valuable benchmark to the community for future CPS fault detection studies.

Our research questions and results are as follows:

**RQ1 (Accuracy)** How accurately does Mithra’s clustering method distinguish between correct and erroneous traces?

To assess the accuracy of Mithra, we classify every trace in the evaluation set. As the acceptance rate is increased, recall decreases and precision increases, resulting in a more conservative model that detects fewer erroneous traces overall, but ensures that traces marked as erroneous are more likely to be truly erroneous. Naturally, the overall accuracy of Mithra remains fairly steady as the acceptance rate is increased, demonstrating the

\[^{1}\text{https://docker.org}\]
tradeoff between the number of false negatives and false positives. Mithra ($\theta = 1.5$) successfully detects 56% of erroneous traces and marks 75% of truly correct traces as correct.

**RQ2 (Conceptual Validation)** How do Mithra’s individual steps influence the overall accuracy of the classifier?

Each of the three steps of Mithra’s clustering approach is designed to improve the accuracy of its detected clusters. To evaluate the individual impact of those steps, we use the output produced by each step (i.e., preclusters, subclusters, and behavioral clusters) as the input to the classifier, which we use to measure the performance of each step. Mithra’s precision and accuracy is significantly improved by its identification of subclusters (i.e., purifying). Merging effectively reduces the number of reported clusters while ensuring that information is preserved. By first applying preclustering to a low-resolution version of its training data, Mithra significantly improves its overall precision and accuracy.

**RQ3 (Effectiveness)** How does the classification accuracy of Mithra compare to AR-SI [55], a state-of-the-art oracle learning approach for CPSs?

To compare our approach with the state-of-the-art, we implement He et al.’s approach for creating autoregressive system identification (AR-SI) oracles for CPSs [55], and compare its performance on the evaluation data against Mithra. The median precision, recall, and accuracy of AR-SI are 62.2%, 39.0%, and 57.8% respectively, compared to Mithra’s 74.7%, 56%, and 69.3%. Mithra significantly outperforms AR-SI in terms of precision, recall, and accuracy, and identifies more erroneous traces while reporting fewer false positives.

Overall, we showed that Mithra achieves the highest median accuracy of 69.3%, successfully detecting 56% of erroneous traces, and labeling 75% of truly correct traces as correct. Mithra significantly outperforms AR-SI on accuracy, precision, and recall. In addition, we showed that Mithra correctly labels the traces more consistently than AR-SI and is less affected by randomness.

### 4.4 Limitations and future work

Our approach, like others, treats anomalous behavior as erroneous [17, 31, 42, 72, 91, 117]. However, anomalous behavior also includes corner cases and rare behaviors that are not observed during training, which are not necessarily erroneous. Although labeling exceptional-yet-correct behaviors as erroneous leads to more false positives, it also informs developers of under-tested functionality, and is in general valuable to the testing process. In addition, Mithra reports frequently-observed-but-erroneous behaviors as correct. We believe that frequently-observed behaviors are unlikely to be erroneous as developers would be able to easily spot them.

Like all dynamic specification mining techniques, the performance of our approach depends on data it is provided [17, 42, 72, 91]. If the provided traces do not provide sufficient coverage of the unique behaviors of the robot, our approach will fail to identify those behaviors. In the next chapter, I propose approaches to generate high quality tests which can eventually be used to generate training traces with higher coverage of unique behaviors of the system.
In future, I propose to apply Mithra on at least one more CPS, possibly a ROS system, to minimize the threat of solely evaluating on a single system and show Mithra’s effectiveness on ROS systems.
5 Automated Test Generation

In this section I present ongoing and proposed work. One of the steps in the automated testing pipeline presented in Figure 1.1 is automated test input generation. Overall, my goal is to identify the greatest number of faults in the system using automated testing in simulated environment. Automated test suite generation is an integral part of test automation process, significantly affecting how well the testing process can find all the flaws in the system (i.e., software test quality) [67]. Therefore, an automated software testing should include automated test suite generation.

5.1 Motivation

The input space in a CPS is massive (e.g., system’s configuration, environment, commands and parameters), making it infeasible to be exhaustively covered [97]. Therefore, searching in the input space for automatically constructing meaningful test cases is one of the most important needs in the CPS testing domain [50].

The automated test input generation approaches broadly can be divided into two categories: 1) Model-based testing, which provides techniques for automatic test case generation using models extracted from software artifacts, and 2) Search-Based Software Testing (SBST), which is a method for automated test generation based on optimization using meta-heuristics [81]. As mentioned in Section 2.1, many model-based approaches have been proposed for automated test generation of CPSs [80, 85, 113]. However, these approaches require a model of the system (e.g., Simulink models or finite state machines), which may not be available for many CPSs. SBST approaches on the other hand, do not require models of the system, and are shown to be effective in finding errors in CPSs (e.g., automated driving control applications [50]).

However, an evolutionary (search-based) approach for test generation requires a fitness function that can measure the quality of generated tests and evolve them towards higher quality. The fitness function can highly affect the SBST process and its effectiveness [15]. I hypothesize that conventional metrics such as code and branch coverage may not be the best measures of test quality in CPSs as these systems have special structures and features. For example, due to inevitable non-determinism and noise in these systems, uncertainty-wise model-based testing solutions have been proposed [16], which explicitly aim to generate tests with higher coverage of known uncertainties. In addition, scenarios that execute the faulty code may not result in a faulty behavior unless a specific condition arises. This phenomena is called Failed Error Propagation (FEP), and reduces the effectiveness of testing [34]. Therefore, generating tests with high code coverage may not lead to the manifestation of more failures in systems with high levels of FEP.
Table 5.1: An example measurements based on three metrics \((M)\) on three test suites \((TS)\), and the ground-truth test suite quality \((Q)\). The measurements are normalized to \(\mu = 0.5, \sigma^2 = 0.16\). In this example, metric \(M2\) has the lowest distance from the ground-truth test suite quality measure, and is more suitable to be used as a metric for test suite quality.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>TS(_1)</th>
<th>TS(_2)</th>
<th>TS(_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric (M_1)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>Metric (M_2)</td>
<td>0.00</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>Metric (M_3)</td>
<td>1.70</td>
<td>0.20</td>
<td>0.23</td>
</tr>
<tr>
<td>Test Suite Quality (Q)</td>
<td>0.23</td>
<td>1.70</td>
<td>0.20</td>
</tr>
</tbody>
</table>

I propose to identify a scalable, cost-efficient fitness measure for test suite quality of CPSs, and investigate different methods of evolutionary (search-based) automated test input generation methods to specifically target corner cases and rarely observed behaviors to expose more bugs and failures. I propose to answer two research questions:

**RQ1:** What metrics can best evaluate the quality of test suites in CPSs?

**RQ2:** What is the performance of different automated test generation approaches at exposing faulty behaviors in CPSs?

By responding to these research questions, I will identify quality measures for CPS test suites, examine the severity of FEP in CPSs, evaluate test generation approaches on these systems, and finally propose test generation approaches capable of generating higher quality test suites.

### 5.2 Methodology

**RQ1 (Quality Metrics)** In this work, I define the quality or effectiveness of a test suite as its ability to reveal faults in the system [67]. The first research question focuses on identifying the ability of different metrics to measure the quality of CPS test suites. The goal of this RQ is to identify metrics that can best represent a test suite’s quality. In other words, given a set of test suites \(TS\) and a ground-truth test suite quality measure \(Q\), we look for a metric \(M^*\) from the set of metrics \(M\) such that:

\[
M^* = \arg \min_{M_c \in M} \delta(Q, M_c; TS)
\]

where \(\delta\) measures the distance between two metrics on a set of test suites. A common distance measure between two metrics is Euclidean distance (i.e., L2 norm):

\[
\delta(M_1, M_2, TS) = \sqrt{\sum_{TS_a \in TS} (M_1(TS_a) - M_2(TS_a))^2}
\]

The Euclidean distance is inexpensive to compute, and the distance between any two test suites is not affected by the addition of a new test suite to the study. However, the distances can be greatly affected by differences in scale among the metrics [28]. Therefore, we first need to normalize the output of all metrics. In this study, I will use Z-score normalization [52]. Z-score normalization can be applied to raw scores gathered from different metrics, and takes into account both the mean value and the variability in a set of raw scores.
Table 5.1 presents an example of normalized measurements of three metrics on three test suites. In this case, metric $M_2$ has the lowest Euclidean distance from the ground-truth $TSQ$, and is more suitable to be used as a metric to measure test suite quality in this system.

To conduct this study I need to assemble three pieces; First, I need a set of metrics $M$ to evaluate. Second, I need to generate a set of test suites $TS$. Third, I need to be able to measure the ground-truth test suite quality $TSQ$. For the set of metrics, I will use a number of different metrics proposed in the literature such as statement and model coverage, and MC/DC, and possibly combinations of these metrics as suited. As test suites, I will use random or off-the-shelf test generation approaches to generate test suites. Finally, to be able to measure the quality of test suites for the ground-truth, I require a set of CPS bugs, and an accurate oracle that can correctly label whether an execution trace is erroneous or correct.

To create the set of bugs in a CPS we can use the same approach as the one described in Chapter 4. I can find historical bugs in the system. However, finding these bugs requires a lot of manual labor. On the other hand, we can inject bugs to the system. This process can be automated and is much faster. Even though injected faults may not be a good representative of real bugs in CPSs, and introduce a threat to validity of the evaluation [68, 87], they can also be effective measures for the quality of test cases [21]. For this part of the work, I will use mutation score [60] as ground truth metric of test suite quality, where the test suite’s score is based on the number of injected bugs it can expose.

To compute the mutation score for a given test suite, I require an accurate oracle that can act as ground truth and determine whether a test execution is faulty (i.e., exposes the injected bug). I can use the oracle created in Chapter 4 to label the test outputs. However, these oracles are not 100% accurate and are biased towards a specific approach. The evaluation of automated test generation should be done independently of how good the oracles are, as these two steps are orthogonal and can be replaced by another (better) approach at any time.

I have the following options to approach the problem of labeling test outputs as ground truth for computation of mutation score:

1. I can take the approach similar to the evaluation of test oracles in Chapter 4 where as ground truth I labeled all traces that executed the faulty code as erroneous, and correct otherwise. However, as mentioned, this is a simplifying assumption and a threat to validity of the evaluation. I can limit the set of CPS defects to those that manifest if the faulty code are executed. In this case, I can automatically check whether executing a test covers the faulty line and consider that as fault revealing test. The disadvantage of this approach is that it limits the CPS faults that can be used in the evaluation and is biased towards test suites with high code coverage.

2. For the CPS under test, I can use a set of specifications to determine the correctness of system’s behavior. If such specifications are already available I will use the provided specifications, otherwise, I will manually write these specifications or ask experts to write them. This approach can only be applied to fairly simple systems. Accurately specifying the behavior of a complex CPS is difficult [50]. Therefore, if I select this approach for generating ground truth data, I have to limit the evaluation to fairly simple systems. However, this approach has the advantage of being consistent, reproducible, and unbiased.

3. I can ask a number of experts to manually label system’s executions and create the ground
truth data. The advantage of this approach is that it can be applied to complex systems. However, it requires manual attention of experts, which may be biased and unreliable. In addition, since this is a very time consuming process, it can only be used for a very small number of traces, and will result in very small test suites.

Depending on the CPSs available for evaluation, I will select one of the above options, and measure the quality of test suites generated by different approaches.

Note that by responding to RQ1 with this study, I can also examine the severity of FEP in CPSs. In other words, I can study how often execution of faulty lines result in a failure. Understanding the severity of FEP in CPSs can guide future quality assurance tools and techniques.

RQ2 (Test Generation Performance) In the second research question, I will evaluate performance of different test generation approaches on CPSs based on the metric(s) identified in RQ1, and propose novel approaches of generating higher quality test suits for CPSs. I will focus on search-based approaches since their application to CPSs have been promising [19, 20, 50, 54, 108, 109], and they do not require an accurate model of the system, the way model-based approaches do [80, 85, 113].

I will start by evaluating evolutionary methods such as genetic algorithm to generate test suites, and later expand those methods to generate higher quality test suites. Since the fitness function of SBST approaches has significant impact on their performance [15, 99], I will substitute a better fitness functions informed by the answer to RQ1. For instance, if the study in RQ1 shows that code coverage is a strong metric for measuring the quality of tests, I will propose a test generation approach that aims for maximizing code coverage.

I envision to adapt ideas from model-based test generation approaches such as uncertainty-wise testing [16], and fuzzing techniques to generate a more diverse set of test inputs (e.g., command parameters, configurations, or simulation environments) that specifically target corner cases, and points of failure.

Time permitting, I will use higher quality tests to refine the automated oracle clusters in Chapter 4. I believe that having a more diverse set of traces as training data for Mithra can result in higher accuracy of behavioral clusters, which can lead to more accurate oracle.
6 Proposed Timeline

I propose the following schedule with an expected defense date of May 2021. At the time of writing this thesis proposal I have completed and published the qualitative study described in Section 3.1 and the case study on ARDUPILOT bugs described in Section 3.2. I prepared the survey in Section 3.3 and have collected responses to the survey. However, I need to analyze the collected data and prepare the article to be submitted to a robotics venue.

I have implemented automated oracle learning approach, Mithra, described in Section 4, and submitted to FSE 2020. I have made initial investigations into the work proposed in Section 5.

- **January-March 2020**
  - Complete analysis of the data collected from survey on robotics simulators and submit the results to IROS (deadline March 1).
  - Apply Mithra on a ROS system, and prepare the results to be submitted to FSE (deadline March 5).

- **March-June 2020**
  - Finish the proposal document, and carry out the thesis proposal.
  - Select a number of robotic systems as samples and prepare the engineering infrastructure needed to be able to run them in simulation.
  - Prepare infrastructure to generate tests for the selected systems.
  - Write specifications for the selected systems that can be used as oracle to the tests.

- **June-July 2020**
  - Prepare the infrastructure to be able to collect a number of quality metrics on a test suite, such as code coverage.
  - Prepare infrastructure to create mutants of system with faulty behavior.
  - Create a diverse set of test suites too be executed on the systems.

- **July-September 2020**
  - Collect the results of comparing different quality metrics for test suites.
  - Write up the results to be submitted to a software engineering venue (possibly ICSE 2021).

- **September-December 2020**
  - Literature review and initial study on test generation approaches on robotics and cyberphysical systems.
- Design and implementation of test generation for selected systems.

**December 2020-March 2021**
- Conduct experiments by generating test suites with the purpose of exposing under-tested behaviors and detecting higher number of failures.
- Collect the results of the experiments and prepare an article to be submitted to a software engineering venue (possibly FSE 2021).

**March-May 2021**
- Finishing up all works in progress, and resubmitting possibly rejected papers.
- Writing up thesis document.
- Prepare for defense.

**May 2021**
- Defend and graduate.
7 Conclusion

In summary, this thesis proposal covers a number of empirical and non-empirical studies with the purpose of improving automated testing of robotic and cyberphysical systems. The empirical studies identify the state and challenges of testing CPSs, while non-empirical studies propose an automated testing pipeline that improves the quality assurance of these systems in simulated environments.

My work encourages more research and studies to be conducted in this field by identifying the specific challenges of testing CPSs in simulation and illustrating the advantages of automated simulation-based testing. In addition, the automated testing pipeline I propose, which includes automated oracle inference and automated test generation, improves current state of automated testing, and enables future studies on improving different pieces of the proposed techniques (e.g., oracle refinement). The implementation of all the techniques discussed in this thesis proposal will be publicly released to be used by researchers in the future.
Bibliography


[52] Jiawei Han, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. 2011.


[73] Husheng Li. *Communications for control in cyber physical systems: theory, design and applications in smart grids*, chapter 1-Introduction to cyber physical systems. Morgan Kaufmann, 2016.


[82] Phil McMinn, Mark Stevenson, and Mark Harman. Reducing qualitative human oracle


[109] Cumhur Erkan Tuncali, Theodore P Pavlic, and Georgios Fainekos. Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles. In International Con-


