박 사 학 위 논 문
Ph.D. Dissertation

효율적인 사이버 물리 시스템 목표 검증을 위한 모방 학습을 사용한 데이터 기반 환경 모델 생성

Data-Driven Environment Model Generation Using Imitation Learning For Efficient Cyber-Physical System Goal Verification

2023

신 용 준 (愼 鏞 俊 Shin, Yong-Jun)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

효율적인 사이버 물리 시스템 목표 검증을 위한 모방 학습을 사용한 데이터 기반 환경 모델 생성

2023

신 용 준

한 국 과 학 기 술 원

전산학부

효율적인 사이버 물리 시스템 목표 검증을 위한 모방 학습을 사용한 데이터 기반 환경 모델 생성

신 용 준

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2022년 12월 1일

심사위원장 배 두 환 (인)
심 사 위 원 고 인 영 (인)
심 사 위 원 유 신 (인)
심 사 위 원 민 상 윤 (인)
심 사 위 원 신 동 환 (인)

Data-Driven Environment Model Generation Using Imitation Learning For Efficient Cyber-Physical System Goal Verification

Yong-Jun Shin

Advisor: Doo-Hwan Bae

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
December 1, 2022

Approved by

Doo-Hwan Bae
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 신용준. 효율적인 사이버 물리 시스템 목표 검증을 위한 모방 학습을 사용한 데이터 기반 환경 모델 생성. 전산학부 . 2023년. 100+v 쪽. 지도교수: 배두환. (영문 논문)
Yong-Jun Shin. Data-Driven Environment Model Generation Using Imitation Learning For Efficient Cyber-Physical System Goal Verification. School of Computing . 2023. 100+v pages. Advisor: Doo-Hwan Bae. (Text in English)

초 록

사이버 물리 시스템은 환경을 관측하고 행동을 결정하는 소프트웨어 컨트롤러를 통해 물리적 환경과 지속적으로 상호 작용한다. 엔지니어는 사이버 물리 시스템의 필드 운행 테스트 로그를 분석하여 분석 대상 소프트웨어 컨트롤러가 주어진 목표를 어느 정도 달성할 수 있는지 검증할 수 있다. 그러나 통계적으로 유의미한 검증을 위해 필드 운행 테스트를 다회 반복하는 것은 큰 비용이 든다. 시뮬레이션 기반 검증은 사이버 물리 시스템의 목표 검증에 필요한 필드 운행 테스트의 비용을 줄여주는 효율적인 대안이다. 그러나 이를 위해서는 사이버 물리 시스템과 상호작용하는 실제 환경을 대체할 수 있는 정확한 가상 환경 모델이 필요하고, 그 환경 모델을 수작업으로 만드는 것은 어렵다.

본 학위 논문은 소량의 필드 운행 테스트 로그에서 가상 환경 모델을 자동으로 생성하는 새로운 데이터 기반 기법을 제안한다. 이 기법은 실제 환경의 행동을 모방하는 환경 모델을 모방 학습을 사용하여 생성한다. 구체적으로 본 논문은 1) 환경 모델링에 대한 체계적이고 포괄적인 조사, 2) 사이버 물리 시스템의 목표 검증의 정형 프레임워크와 환경 모델 생성의 정형 문제 정의, 3) 모방 학습을 이용한 데이터 기반 환경 모델 생성 기법, 그리고 4) 자율 주행 시스템의 목표 검증 사례에 기반한 실험적 평가 및 재사용 가능한 데이터를 제공한다. 연구 결과는 본 기법이 사이버 물리 시스템 목표 검증을 위해 정확한 가상 환경 모델을 소량의 필드 운행 테스트 로그 데이터로부터 자동 생성할 수 있음을 보인다. 이로써 사이버 물리 시스템 소프트웨어 엔지니어는 물리 환경에 대한 지식이 부족하더라도 정확한 가상 환경 모델을 자동으로 얻을 수 있고, 컨트롤러를 시뮬레이션에 기반해 효율적으로 검증할 수 있다.

핵 심 낱 말 사이버 물리 시스템, 목표 검증, 환경 모델링, 모델 생성, 모방 학습

Abstract

Cyber-Physical Systems (CPS) continuously interact with their physical environment through software controllers that observe the environment and determine actions. Engineers can verify to what extent the software controller under analysis can achieve given goals by analyzing its Field Operational Test (FOT) logs. However, repeating many FOTs to obtain statistically significant results is expensive in practice. Simulation-based verification is an efficient alternative for reducing the FOT cost for CPS goal verification. However, it requires an accurate virtual environment model that can replace the real environment interacting with the CPS, and it is challenging to craft the environment model manually.

This dissertation proposes a novel data-driven approach that automatically generates the virtual environment model from a small amount of FOT logs. It generates an environment model that mimics the behavior of the real environment using Imitation Learning (IL). Specifically, this dissertation provides 1) a systematic and comprehensive survey on environment modeling, 2) a formal framework of CPS goal verification and a formal problem definition of environment model generation, 3) a data-driven environment model generation approach using IL, and 4) an empirical evaluation based on case studies of an autonomous driving system goal verification and reusable datasets. The evaluation results show that the approach can generate accurate virtual environment models for CPS goal verification with small

FOT log data. Therefore, CPS software engineers can automatically obtain accurate virtual environment models and efficiently verify the controller based on the simulation.

Keywords Cyber-Physical System, Goal Verification, Environment Modeling, Model Generation, Imitation Learning

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Introduction to CPS Goal Verification

Cyber-Physical Systems (CPS) utilize both physical and software components deeply intertwined to continuously collect, analyze, and control physical actuators at runtime [4]. CPS has been increasingly studied for many applications, such as autonomous vehicles [5, 6], robots [7, 8], smart factories [9, 10], and medical devices [11, 12]. When developing CPS, software engineers participate in implementing CPS's decision-making mechanism. The decision-making mechanism is the intelligence of selecting appropriate actions to achieve CPS's goals based on the current state of CPS recognized through CPS sensors.

One of the essential problems in CPS development is to verify to what extent the CPS under development can achieve its goals. To answer this, an engineer could deploy a CPS (e.g., an autonomous vehicle) into its operational environment (e.g., a highway road) and verify the CPS's goal achievement (e.g., lane-keeping) using the logs collected from the Field Operational Tests (FOTs). To perform FOT, the engineer must build a physical experimental environment in which CPS can be safely tested and design a test scenario to be performed within that environment. The engineer then runs the planned test scenario repeatedly, collecting a large number of FOT logs.

However, conducting FOTs is expensive, time-consuming, and even dangerous, especially when hundreds of repeats are required to achieve a certain level of statistical significance in the verification results. Therefore, an alternative is a simulation-based approach where the software controller of the CPS is simulated with a virtual environment model. In the simulation, the decision-making mechanism of CPS or the software controller of CPS runs in the virtual environment. This virtual environment or environment model replaces the environment of FOT. Therefore, CPS can be tested in a virtual environment rather than physically, making it cheap and safe to perform sufficient runs necessary for statistical goal verification.

This dissertation proposes to verify CPS's goals based on simulations to reduce the cost of FOT-based CPS goal verification. Specifically, it deals with the generation of the virtual environment model required for CPS simulation.

1.2 Challenges in the Environment Modeling for Simulation-Based CPS Goal Verification

Though the simulation-based CPS goal verification can reduce the cost and risk of the FOT-based CPS goal verification, it requires a highly crafted virtual environment model based on deep domain knowledge. Furthermore, it may not be possible if a high-fidelity simulator for the problem domain does not exist. It prevents the simulation-based approach from being better used in practice. Specifically, the following challenges exist in manual environment modeling.

First, accurately modeling the physical environment of CPS is complex. The environment model used in CPS simulations is the state transition mechanism of the environment sensed by CPS and affected by the CPS actions. However, the number of environmental states that CPS can observe is enormous. The number is proportional to the number of sensors in the CPS and the range of environmental state

values each sensor can recognize. It is also affected by the number of actions that CPS can choose. The number becomes infinite if the environmental state and the CPS actions contain continuous variables. As CPS becomes more complex, environmental models also become more complex, making it difficult for engineers to develop them manually.

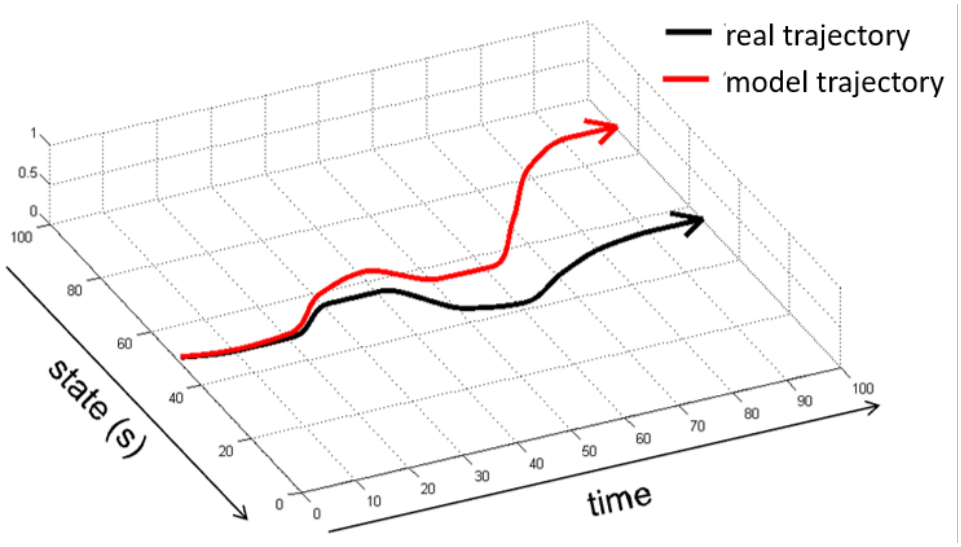


Figure 1.1: Compounding error problem [1]

Second, if a small error exists in the generated environment model, the error in the model may accumulate while simulating the model. The phenomenon in which small errors accumulate while simulating the model for a long time is called a compounding error problem. Figure 1.1 visualizes the compounding error. The black line is the real CPS trajectory, and the red line is the simulation trajectory. At the beginning of the simulation, reality and simulation were almost similar, but the simulation error accumulated over time. The model cannot reduce the accumulated error again. Therefore, it is difficult to make complex environmental models accurately and prevent compounding error problems in the simulation.

Finally, environment modeling requires a significant level of domain knowledge and effort. To accurately model the state transition of the environment interacting with CPS, it is necessary to accurately understand the physical laws in the environment of interest. It requires knowledge at the domain expert level. Also, even if a domain expert models the environment, it takes considerable time and cost. When environment modeling is performed to replace the FOT of CPS with simulation, the cost of generating the environment model should be less than the FOT cost to maximize the effectiveness of the simulation. However, if an expert performs environmental modeling manually, it is difficult to reduce the knowledge, cost, and time spent on the modeling.

1.3 Thesis Statement

This dissertation proposes a data-driven environment model generation for efficient CPS goal verification. We suggest using CPS simulation instead of CPS FOT by generating an environment model from data to reduce the goal verification cost. The environment model generation approach does not require significant domain knowledge to solve the challenges of manual environment modeling. The key idea behind the model generation is to leverage imitation learning for training a model that imitates the in-

teractions between the CPS controller and its real environment as recorded in (possibly very small) FOT logs. We then statistically verify the goal achievement of the CPS by simulating it with the generated model.

Therefore, the thesis statement is:

> By proposing a data-driven environment model generation approach, it is possible to make SW engineers automatically model the real environment of the CPS for simulation-based CPS goal verification.

1.4 Scope and Contributions of the Thesis

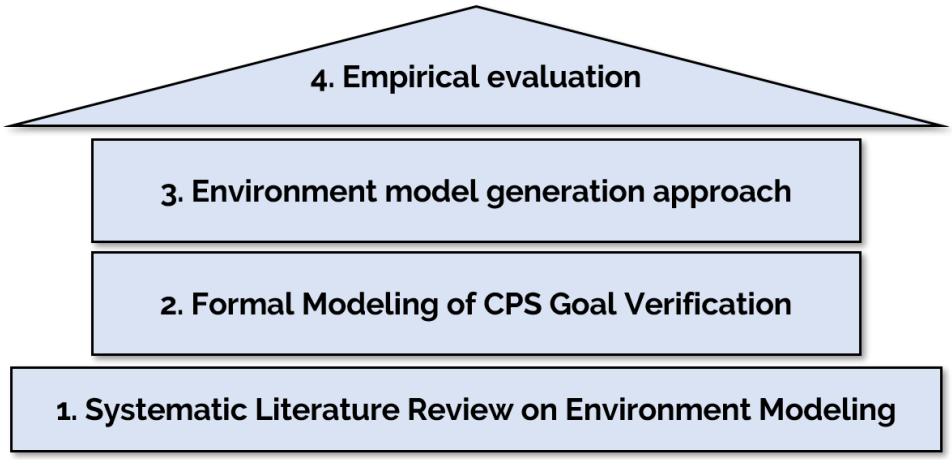


Figure 1.2: Thesis scope

Figure 1.2 shows the scope of this dissertation. This dissertation first conducts a systematic literature review in environment modeling to provide the current research landscape. We then formally model the CPS goal verification process and define the environmental model generation problem from that model. We propose a novel data-based model generation technique that can efficiently solve the model generation problem by utilizing Imitation Learning (IL). Finally, we apply our approach to the goal verification of real CPS controllers.

In summary, below are the contributions of this paper:

1) We provide a landscape of the concepts and models of the environment in the current environment modeling studies.

2) We shed light on the problem of environment model generation for CPS goal verification with a formal problem definition.

3) We propose a novel data-driven approach for environment model generation utilizing IL.

4) We empirically assess the application of our approach through case studies with real CPSs.

1.5 Thesis Organization

This dissertation is organized as follows: Chapter 2 provides the systematic literature review on the environment modeling and compare the related environment modeling approaches and our approach. Chapter 3 defines the formal framework of CPS goal verification and define the problem of environment modeling. Chapter 4 proposes a novel environment model generation approach. Chapter 5 performs an empirical evaluation based on real CPS case studies. Chapter 6 concludes this dissertation by summarizing the achievements of this thesis and suggesting some future works.

Chapter 2. Literature Review on Environment Modeling

2.1 Introduction

This chapter introduces the concept of the Cyber-Physical System (CPS) and the CPS controller which is the engineering target of this thesis. In addition, this chapter introduces the concepts and models of the environment by systematically surveying the environment modeling studies.

2.2 Cyber-Physical Systems and Related System Types

Cyber-Physical Systems

Cyber-Physical Systems (CPS) is a system having both physical and software layers deeply intertwined [4]. The physical layer sense the system and its operational environment through sensors, and actuate the systemaction through its actuators such as motors. In the physical environment the CPS continuously collect, analyze, and behave at runtime. Many modern systems are CPSs, so we can find many applications, such as autonomous vehicles [5, 6], robots [7, 8], smart factories [9, 10], and medical devices [11, 12].

Figure 2.1: An example of the CPS: Autonomous vehicle

Figure 2.1 shows a representative example of the CPS, an autonomous vehicle. Autonomous vehicles are one of the most actively studied CPS these days. Autonomous vehicles are huge machines that exist in the physical environment, and at the same time, they are also computers that judge and act intelligently. According to the basic structure of CPS, an autonomous vehicle has software, the cyber layer, that is the intelligence of the vehicle and physical layer that makes the vehicle's behavior in reality.

In this dissertation, we especially focus on the development of the software part of the CPS, which is the intelligence, decision-making mechanism, or controller of the CPS. In the CPS development, software engineer participates in the development of the CPS controller (sometimes called controller system) that decides the action of the CPS. The CPS controller is given the current CPS state which is observed by sensors. The controller then decides an action to achieve CPS goals (e.g., safety, performance). Therefore, the CPS controller can be defined as a module or a function that returns a proper CPS action depending on the given CPS state.

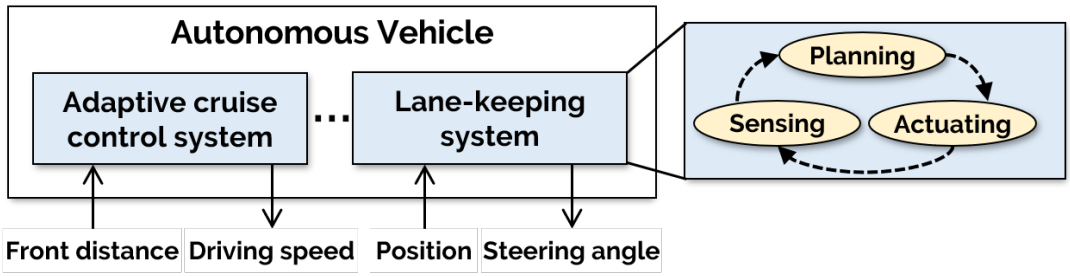


Figure 2.2: Controllers Constituting Autonomous Vehicle

Figure 2.2 shows examples of the controllers of the autonomous vehicle. The adaptive cruise control system of an autonomous vehicle recognizes the forward distance and controls the speed to maintain a safe distance. The lane-keeping system recognizes the position of the vehicle on the lane and controls the angle of the handle so that the vehicle faces the center of the lane. The software engineer who develops the adaptive cruise control system or the lane-keeping system of the autonomous vehicle implement effective mechanisms to return an optimal driving speed or steering wheel angle (i.e., CPS action).

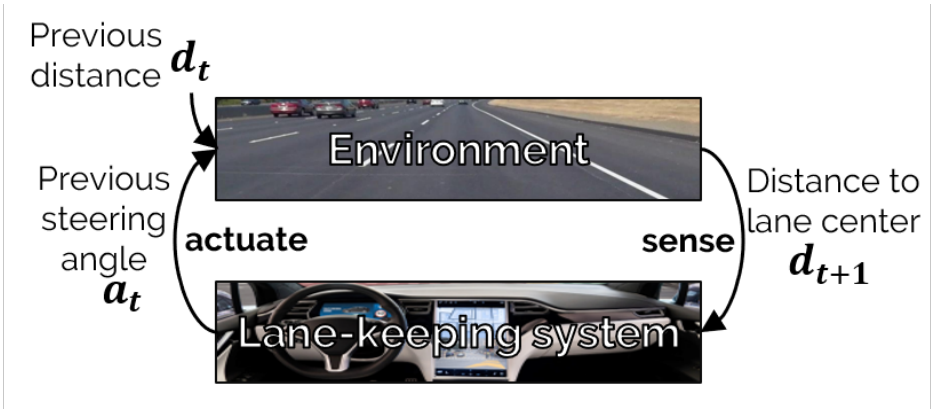


Figure 2.3: Interaction between the CPS controller and the environment

More specifically, a CPS controller continuously interacts with the CPS environment as shown in Figure 2.5. The CPS controller senses the environment according to the sensing capability of the CPS and receives the information. It returns CPS action based on the observed environmental state. The CPS action executed makes a change in the next environmental state. Therefore, the future state of the environment is affected by the previous CPS actions and the environmental states. The changed environmental state is again observed by the CPS and used in the CPS's action-decision. The CPS continues to repeat this process.

Self-Adaptive Systems

Self-adaptive systems (SASs) continuously adapt their behavior or structure to satisfy goals in changing environments [13]. SAS has a feedback loop consisting of a controller that determines the configuration for adaptation [3], which continuously checks the goal achievement and decides on the configuration based on its adaptation strategy. Based on the adaptation strategy, an SAS respond to unanticipated situations of the system itself or its operating environment [2]. These unanticipated situations are referred as uncertainty. Uncertainty can come from imperfect requirements, defective

SAS design or implementation, or the runtime environment [14]. Among these various reasons, the environment is one of the most interesting and challenging entities to address in SAS development. It is difficult to fully anticipate at design time the environment that an SAS will encounter during its operation, and modern systems have environments that are complex and open.

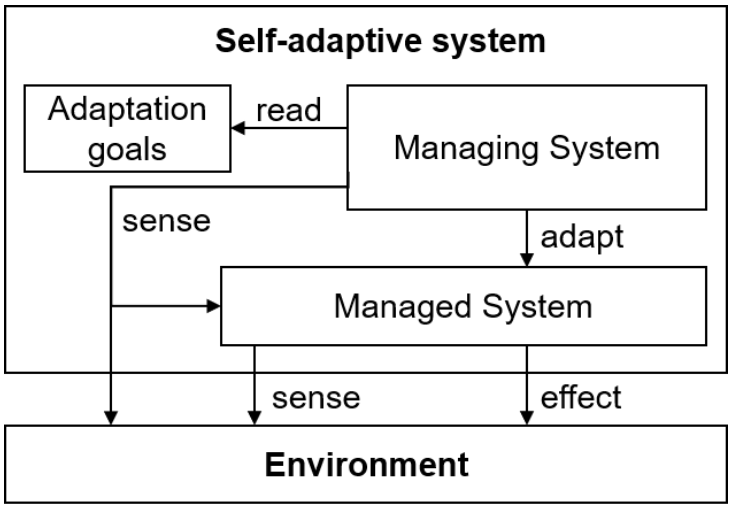


Figure 2.4: Conceptual model of an SAS [2]

Some papers that introduce SAS engineering provide a fundamental comprehension of SASs and the environment [15, 2, 13]. Fig. 2.4 illustrates a conceptual model of an SAS [2], including the relationships between the SAS and the environment. The environment is an external world comprising observable physical and virtual entities where the SAS operates. Given that the environment is regarded as uncertain, the SAS continuously senses it to reliably achieve its adaptation goals. The sensed environmental condition affects the decisions of the SAS, and these decisions, in turn, can have new effects on the environment.

As shown in the Figure 2.4, SAS is a system that interacts with its operational environment. Based on the observation, the SAS decides its action to achieve its adaptation goal. In this perspective, CPS is also a kind of SAS. CPS also observes and interact with environment, and continuously makes decision to achieve its goal in the uncertain and changing physical environment. Almost of the modern CPSs have intelligence for adaptive action-decision, so almost of the CPSs can be regarded as the SAS, and this dissertation also considers them. However a system can be self-adaptive even if it does not have any physical part, so the concept of the CPS is partially subsumed by the SAS.

System of Systems

A System-of-Systems (SoS) is a large-scale complex system which consists of independent and autonomous constituent systems (CSs) contributing to achieve an SoS-level common goal(s) [16]. The SoS goal is one that is hard to achieve with a monolithic system, thus it is achieved by collaborations and cooperations of the CSs as they interact with each other utilizing SoS resources [17]. To guarantee the goal achievement of an SoS, CSs in an SoS should effectively provide their own capabilities in the form of collaborations. In addition, interactions between CSs should enable an effective collaboration within an SoS.

This type is an important and interesting type of the modern systems. Examples of SoS are clusters of vehicles or drones, smart factories where many robotic systems work together, and complex defense

systems with multiple weapon systems. As the size and influence of SoS increase, an important objective of SoS engineering is to ensure that SoS goals are achieved stably regardless of uncertainty. Therefore, for SoS engineers, it is important to verify whether an SoS goal can be successfully fulfilled by the collaboration. For the SoS verification, statistical verification is widely used for quantitative verification [18, 19].

In this perspective, the CPS having multiple CPS controllers is SoS [20, 21]. Independent CPS controllers have their own goals. For example, the adaptive cruise control system of autonomous vehicles aims to maintain a safe distance, and the lane keeping system aims to drive along the center of the lane. Each CPS controller operates independently, but at the same time, it can be seen that it works together for the higher goal of the CPS within the same CPS. Therefore, it can be seen that there is an intersection between CPS and SoS.

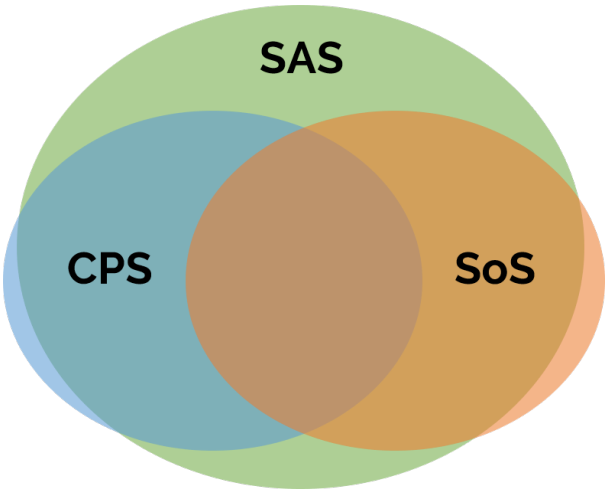


Figure 2.5: Relationships of CPS, SAS, and SoS

Figure 2.5 summarizes the relationships of CPS, SAS, and SoS. There is the intersection of the CPS and SoS, which represents the case of CPSs having multiple autonomous CPS controllers. In addition, almost of the CPS and SoS aim to have the adaptation capability to the uncertainties in the environment or the system itself. Therefore large portion of the CPS and SoS are overlapped in the concept of SAS. In this dissertation, we consider the CPS that adaptively decides its action responding the uncertain environment, and also have one or more CPS controllers in the CPS.

In the following section, we systematically survey the environment modeling of the SAS, which embraces the concept of CPS. To provide a domain-general concepts and models of the environment, not limited to the physical environment, we surveyed the environment modeling of the SAS, not the CPS. Because the environment of SAS also represents the broad concept of the CPS environment, we first understand the environment of SAS, and we then utilize the survey result for the CPS environment model generation.

2.3 Systematic Review Protocol on Environment Modeling

To develop a system that is adaptive to an uncertain environment, such as SAS and CPS, numerous engineering approaches have been proposed, such as eliciting adaptive requirements from the environment [22, 23], analyzing SAS design while considering an uncertain environment [24, 25], testing an SAS implementation with environmental inputs [26, 27], and updating environmental knowledge for optimal

runtime decision making of an SAS [28, 29]. In this context of active research on an SAS in an uncertain environment, one shortcoming we noticed is that the meanings of "environment" and "uncertain environment" are inconsistent across different studies. For example, different papers describe "uncertain environment" as an environment that changes itself over time, an environment that has been changed by an SAS, or an environment that has been misrecognized by sensor noise, among other definitions. This inconsistent understanding makes it difficult to compare different studies.

Although there could be many reasons for this inconsistent understanding of the environment, what we focus on is the lack of overall knowledge of how other researchers have interpreted the environment of an SAS. In the software engineering community for SASs, an implicit agreement on the concepts of the environment has been reached, but this agreement has led to ad hoc interpretations. We believe that the various interpretations of the environment of an SAS are all meaningful in establishing a concrete knowledge of it. Therefore, in this paper, we conducted a systematic literature review (SLR) to gather and analyze these interpretations. We specifically tried to find out:

- how various researchers commonly understand the concept of the environment of SAS, and
- if there are cases in which their understanding of the environment is expressed as concrete models.

For the purpose, we automatically and manually searched 3719 papers and selected 128 papers as primary studies. We examined the how the studies defined and described the SAS environment and how existing studies abstracted it as models. Specifically, in our SLR, we found and provided:

- five common characteristics of the environment of SAS and their trends in the primary studies
- two common sources of environmental uncertainty and their trends in the primary studies, and
- 14 reference environment models for SAS with different purposes and expressiveness for the characteristics.

On the basis of some guidelines for SLR [30, 31, 32], we designed a review protocol that includes the review steps and specific inputs and outputs for each step (Figure 2.6). Designing a review protocol in advance prevents a biased or subjective survey, and disclosing it ensures a reproducible review. According to the goal of this SLR, we specified the RQs, automated search engines, manual search venues, and the search string. The papers searched were evaluated to determine whether they were primary studies[1] under the predefined criteria. The selected primary studies were then examined thoroughly. These studies also became the sources of cross-reference searching, a step in which all the references of the primary studies were exhaustively explored to minimize the possibility of missing important papers. Any newly discovered paper was evaluated according to the selection criteria. In particular, we utilized the "snowballing" method[2]. When searching finished, we extracted predefined data items from the primary studies. The extracted data were analyzed, and the analysis results are reported in Sections 2.4 and 2.5. The rest of this section describes the elements of this protocol.

The purpose of this SLR is to show the trends of how the concepts of the SAS environment have been understood and abstracted as environment models of SAS in software engineering. To achieve this purpose, we specified questions that will be answered, as shown in Table 2.1. Regarding RQ1, to understand the environment of SAS, we surveyed how primary studies have explicitly defined the environment.

[1] In this case, a primary study refers to a paper subject to review, and the SLR itself is a secondary study [30].

[2] The snowballing method exhaustively explores all the backward references (cited by the subject paper) and forward references (citing the subject paper) until no additional papers are discovered [33].

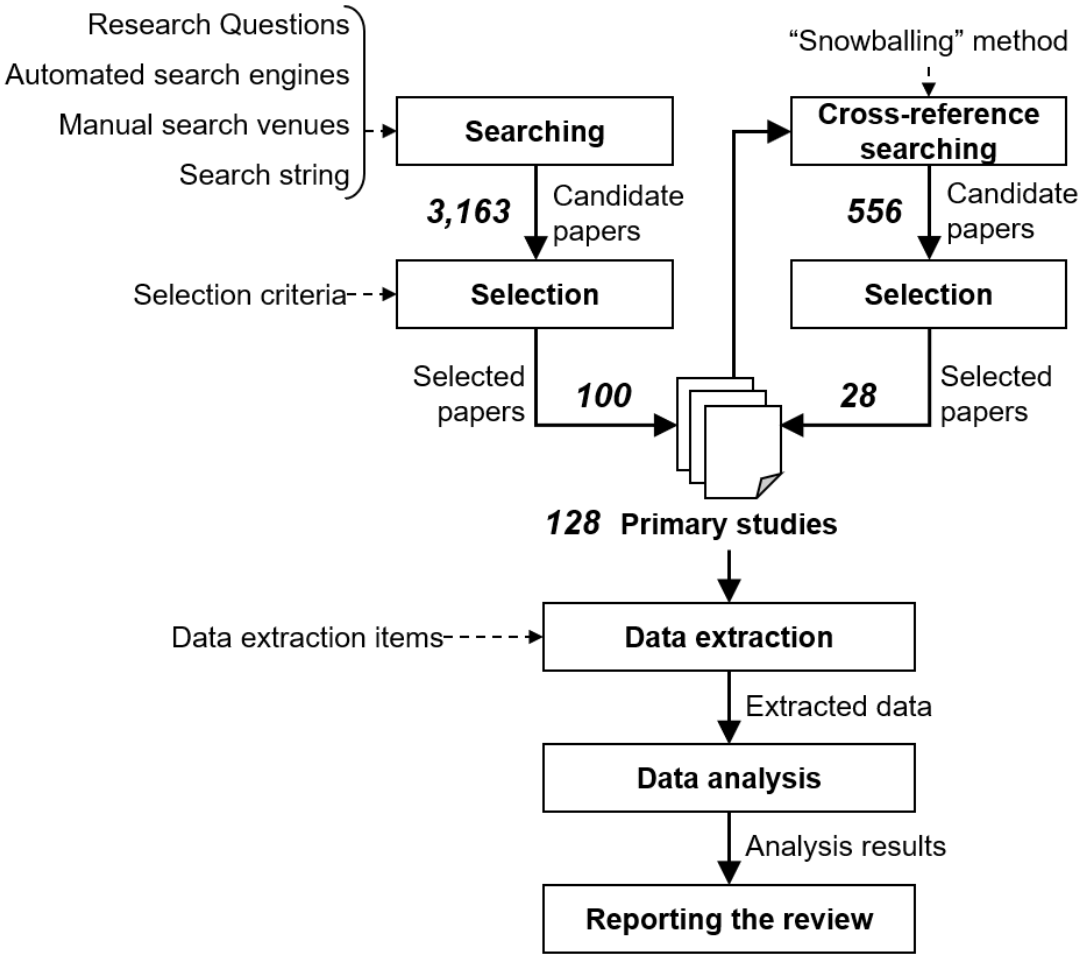


Figure 2.6: Overview of the review protocol

For RQ2, both the explicit definitions and characteristics used to describe the environment were clarified. RQ3 was included because environmental uncertainty is a huge area of interest in software engineering for SAS but remains an ambiguous term. In this SLR, we surveyed sources of the environmental uncertainty and their coverage in primary studies. For RQ4, we selected papers from the primary studies that proposed environment models and surveyed these modeling methods. RQ5 looked at the application of the environment models. Finally, in RQ6, we examined the expressiveness of the environment models, especially how the characteristics of the environment were represented in each model.

Different automated search engines that could help find related papers were utilized to collect appropriate primary studies for answering the RQs. The selected search engines are listed in Table 2.2. Widely used computer science article search engines were selected, and various multi-disciplinary search engines were also used to search exhaustively for as many related works as possible. In addition, we conducted a manual search for publications in related journals and conferences (Table 2.3) for added focus on high-end software engineering and SAS-related venues.

The following search string was used to find related papers:

{(**self-** OR **adapt**) AND (**software** OR **system**) AND (**environment**) AND (**uncertain**)}

Papers focusing on "software" or "system" with "self-" prefixed properties or "adapt" (as in "adaptive,"

Table 2.1: Research questions of SLR

Category	ID	RQ
Concepts of the environment of the adaptive CPS	RQ1	*Definitions of the environment.* How did primary studies explicitly define the "environment" of an adaptive CPS?
	RQ2	*Characteristics of the environment.* What characteristics of the environment of an adaptive CPS did primary studies mention in describing it?
	RQ3	*Sources of the environmental uncertainty.* What did primary studies consider to be the sources of environmental uncertainty?
Models of the environment of the adaptive CPS	RQ4	*Modeling of the environment.* Who models, how do they model, and why do they model the environment of the adaptive CPS?
	RQ5	*Application of the environment models.* When and how are the environment models used?
	RQ6	*Expressiveness of the environment models.* How are the characteristics of the environment expressed in the models?

Table 2.2: Automated search engines

Discipline	Search engine
Computer science and related subjects	IEEE Xplore (http://ieeexplore.ieee.org/)
	ACM Digital Library (http://dl.acm.org/)
	DBLP Computer Science Bibliography (https://dblp.org/)
Multi-disciplinary	Web of Science (http://www.webofknowledge.com/)
	SpringerLink (http://link.springer.com/)
	Scopus (http://www.scopus.com/)
	Wiley Online Library (http://onlinelibrary.wiley.com/)
	World Scientific (https://www.worldscientific.com/)
	ScienceDirect (http://www.sciencedirect.com/)

"adaptiveness," etc.) were searched. The "self-" prefix identifies the most general terms of various adaptive properties [34]. We likewise searched for studies explicitly referencing the uncertain environment or environmental uncertainties of SAS, which were both caught by our specification of forms of "environment" and "uncertain." This search string was used for both the automated and manual search; the search scope included titles, abstracts, and author keywords of the papers.

The searched papers were evaluated using the predefined selection criteria in Table 2.4. There were both inclusion and exclusion criteria. If a paper satisfied all the inclusion criteria and none of the exclusion criteria, then it was selected as a primary study. Inclusion criteria IC4 evaluated whether a paper was appropriate to answer our RQs. Our purpose was to gain a general knowledge of the environment of an SAS from papers on developing systems to be adaptive to the environment, so only domain-general SAS engineering papers were included. All the authors of this work read the abstracts of the papers (and the introductions if needed) and together judged if the papers were appropriate to answer our RQs. Other criteria helped control the discipline focus, quality, and form of the primary studies.

Extracted data items were identified for each RQ (Table 2.5). Following our predefined review protocol, we searched 3163 papers (2987 automatically, 176 manually) and selected 100 primary studies.

Table 2.3: Manual search venues

Type	Venue
Journal	ACM Transactions on Software Engineering and Methodology
	ACM Transactions on Autonomous and Adaptive Systems
	IEEE Transactions on Software Engineering
	Journal of Systems and Software
	Information and Software Technology
Conference	Intl. Conference on Software Engineering
	Intl. Symposium on the Foundations of Software Engineering
	Intl. Conference on Automated Software Engineering
	Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems
	Intl. Conference on Self-Adaptive and Self-Organizing Systems

Data extraction was conducted manually, and the collected data were analyzed to answer the RQs. Using the "snowballing" method, we searched an additional 556 references and selected 28 more primary studies. Thus, a total 128 primary studies were surveyed (Figure 2.6). The details of searching and selection, such as the number of papers for each engine and venue or the criteria evaluation results, are accessible on our website but not fully described here[3]. From the primary studies, we extracted data and analyzed these to answer the six RQs. Throughout all the review steps, to create a reproducible and objective survey, we recorded all the outputs for each step and made all the review data, including extracted raw data, accessible[3]. In this section, we report the analysis results for each RQ.

Following section reports the review results for each RQs.

2.4 Review Result 1: Concepts of the Environment

2.4.1 Definitions of the Environment

We first collected explicit definitions to understand the environment of an SAS. We searched sentences explicitly defining "environment," such as "environment is defined as..." or "environment means...." Owing to the strict format of sentences, only three explicit definitions were found, as listed in Table 2.6[4]. [35] defined environment as external and observable objects. [36] highlighted the fact that it is not under the direct control of an SAS. By contrast, [37] defined environment as circumstances interacting with the SAS. In paraphrasing the existing definitions, we can say that *the environment of an SAS is a set of external and observable objects that are not under the control of the SAS but interact with it.*

The definitions are acceptable and indicate some key characteristics of the environment, such as *diverse factors*, *externality*, *observability*, and *interaction*. However, only a few of the selected studies explicitly defined environment, and they varied considerably in terms of the authors' perspectives. Such differences made it difficult to get considerable knowledge about the concept of environment from the existing definitions only. This outcome confirmed the assumptions that drove our motivation to conduct this SLR. Fortunately, the studies without explicit definitions implicitly shared a common understanding

[3]Access the SLR website for all the review data: https://sites.google.com/se.kaist.ac.kr/sas-environment-slr/

[4]Citation numbers for primary studies begin with "*P*". A list of the primary studies is provided on our SLR website[3] due to lack of space.

Table 2.4: Inclusion and exclusion criteria

Inclusion criteria	
IC1	Papers written in English
IC2	Research papers peer-reviewed and published in conferences, journals, or books
IC3	Papers in computer science field
IC4	Papers on the topic of a domain-general software engineering approach for self-adaptive systems' adaptation to the environments
Exclusion criteria	
EC1	Duplicated papers
EC2	Papers whose contents were not fully accessible
EC3	Papers not in the form of full research papers (i.e., abstracts, tutorials, or reports)
EC4	Collections of studies (i.e., books or proceedings)
EC5	Papers summarizing existing studies or concepts (i.e., overviews, introductions, keynotes, roadmaps, or surveys)

Table 2.5: Data extraction items

RQ	**Data items**
RQ1	Explicit definition of the "environment" of an SAS
RQ2	Expressions explicitly mentioned to describe characteristics of the environment
RQ3	Sources of environmental uncertainty addressed in the primary studies
RQ4	Environment modeling details (modeling agent, effort, purpose, formalism, process, etc.)
RQ5	Environment model application details (application time, usage, supportive techniques, etc.)
RQ6	Characteristics of the environment expressed in the models

about the environment. consequently, we attempted to gather this understanding in answering RQ2 on the basis of these definitions.

2.4.2 Characteristics of the Environment

To establish the concept of the environment of an SAS, we collected characteristics of this environment. Despite the limited explicit definitions in RQ1, almost all the primary studies described the environment of an SAS of their interest. We searched for all the sentences that included "environment" in the primary studies and collected and categorized the numerous adjectives and nouns from the sentences that described the environment, as shown in Table 2.7. The expressions in the primary studies are organized in the first column and then listed in the second column. We discussed how to classify various expressions into some common characteristics and, finally, organized the five common characteristics of an SAS environment. Descriptions for each characteristic and the related expressions are also given in Table 2.7.

Diversity: Environment comprises diverse environmental factors. The term environment does not only mean a specific object, but a set of environmental factors of interest. Specification of the

Table 2.6: Definitions of environment

Explicit definition of the "environment" of SAS	**Ref.**
"anything observable by the software system, such as end user input, external hardware devices and sensors, or program instrumentation"	P6
"the physical world or computing elements that are not under control of the system"	P24
"circumstances that interact with or affect the system"	P77

environment requires a set of specifications of each environmental factor of interest. An environmental factor could be cyber, physical, human, external service or systems, or even time. As there can be various environmental factors, they may each have their own constraints or rules, such as law of physics. Therefore, the environment of an SAS should be finally defined according to the domain knowledge.

Externality: Environment is outside the SAS boundary. Therefore, only objects that are outside the system boundary can be regarded as environmental factors. Given its externality, environment is not under the direct control of the SAS. It is not directly modifiable by an SAS like a system variable, but an SAS can give a stimulus to the environment through actuators and so on.

Observability: Every external object of interest can be regarded as an environmental factor of an SAS, but a constraint is that the object must be observable by the SAS. Therefore, defining an environment of an SAS is related to the monitoring capability of the SAS. In SAS academia, we do not regard all external things as an environment but as external and observable things. Environment is observable by monitoring components of an SAS, so the SAS can respond to the environment.

Interactivity: Environment and SAS interact and thus affect each other which is why the adaptation of a system to the environment is needed. Environment specification should specify the mutual influence of the environment and the SAS. Environmental influence on the SAS can be adverse or supportive of SAS goals. An external and observable object not related to and interacting with the SAS does not need to be regarded as an environmental factor.

Uncertainty: Environment is not certainly anticipated at design time. It is uncertain because it is an external element. If SAS engineers have considerable domain knowledge, then a better expectation of the runtime environment can be made, but a complete knowledge of the external factor is almost impossible. Continuous environment monitoring of the SAS reduces the uncertainty. Numerous expressions implying limited and incomplete knowledge about the environment, such as unknown, change, dynamic, probabilistic, and so on, are used.

Figure 2.7 shows the summary of the primary characteristics of the environment.

Figure 2.8 shows how many papers mentioned each characteristic of the environment. This information indicates what environmental characteristics were relatively familiar to the researchers as expressed in their writing. For example, "dynamic operating environment" was one of the most widely used expressions to describe the environment. The figure shows trends in the characteristics mentioned. More important than the trends, however, is that in addressing RQ2, the various characteristics and expressions were organized to help understand the environment more comprehensively. Although there were few clear definitions, the SAS research community has established a significant and implicit agreement on the characteristics of an SAS environment. Lastly, we were able to make these agreed upon characteristics explicit and visual.

Table 2.7: Characteristics of the environment of SAS and their expressions

Characteristics	**Explicit expressions**	**Description**
Diversity	(Diverse factors) computing/physical (environment element), user (human), system, service, time, factor	The environment consists of diverse (environmental) factors/elements, for example, computing or physical elements, users (humans), or external systems (services).
	(Constrained) constraints	An environmental element has its own constraints.
Externality	(External) external, surrounding	The environment is outside of the SAS boundary.
	(Operational) operation, execution, deployment, runtime	The environment is where a system is deployed, operates, and executes at runtime.
	(Out of control) no control, indirect	An SAS cannot (directly) control its environment.
Observability	(Observable) observable, sense, monitor, measure	The environment is observable by an SAS.
	(Interpretable) parameter, attribute, variable, value, data, input, condition, event, phenomena	An SAS perceives and interprets its environment based on data or values of the environmental variables or parameters. Perception is the SAS's environmental input condition or event.
Interactivity	(Interaction) interaction, influence, affect, impact, interface, trigger	The environment interacts with an SAS, so it affects and is affected by the SAS.
	(Media) sensor, effector/actuator	An SAS interacts with the environment through its sensors and effectors (actuators).
	(Incompleteness) error, noise, variation in sensing, signal interference, failure	Interactions between environment and SAS can be incomplete or may be failures.
	(Adverse influence) disturbing, unsafe, adverse, disruptive, unfavorable, threat	The environment may adversely affect SAS goal satisfaction.
	(Supportive influence) resource	The environment may be supportively used for SAS goal satisfaction.
Uncertainty	(Unpredictable) uncertain, unforeseen, unexpected, unpredictable	The environment is not fully anticipated at the design time of an SAS.
	(Misunderstanding) unknown, lack of knowledge, missing	Knowledge of the environment may be incomplete. SAS can meet an unknown environment. Missing environmental parameters could exist.
	(Dynamic) change, fluctuation, dynamic	The environment dynamically changes its states or behavior over time.
	(Probabilistic) non-deterministic, probabilistic, stochastic	The environmental knowledge is non-deterministic.

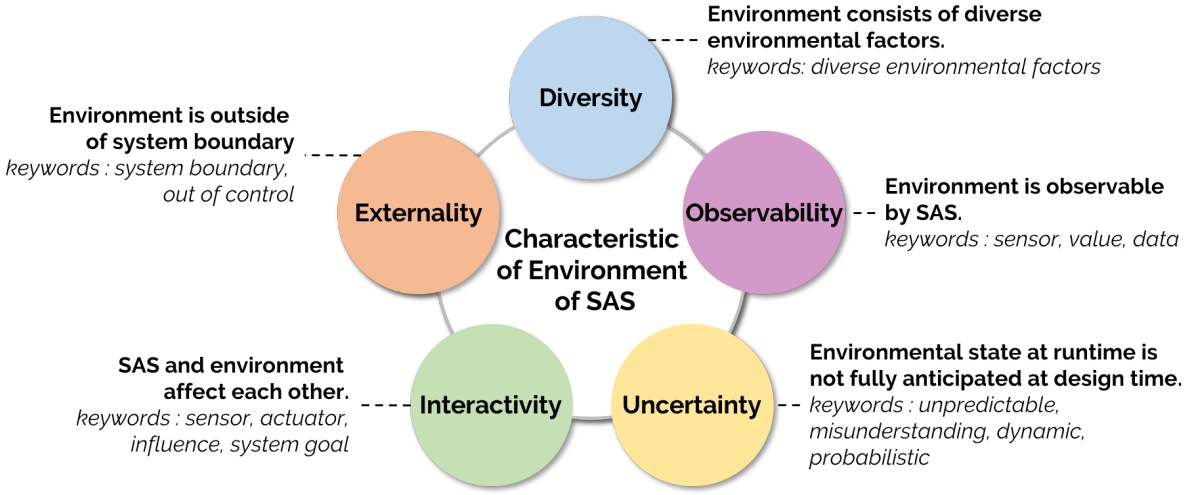


Figure 2.7: Environment characteristics

2.4.3 Sources of the Environmental Uncertainty

Among the characteristics of an environment, uncertainty is one reason that a system should monitor and adapt continuously to the environment. However, the use of the term "uncertainty" is typically conceptual and ambiguous and can thus cause inconsistent understanding among engineers. To tackle ambiguous understanding[5], we examined concrete sources that cause environmental uncertainty. In the selected primary studies, we found three papers [P22, P94, P102] that proposed taxonomies of environmental uncertainty sources. We leveraged their taxonomies to analyze which sources were widely addressed in the primary studies. We summarized these taxonomies of sources[3] and reorganized them as presented in Table 2.8.

As the descriptions of existing sources had overlapping meanings, so we reorganized the sources into two common sources. The first common source of environmental uncertainty is ***limited environmental knowledge***. An SAS engineer may have limited knowledge about the environment because the environment changes or the environment was not fully identified. Sometimes, SAS engineers can miss some environmental factor in consideration. The primary studies have divided this source into different types of environmental factors (cyber, physical, and human). However, the common reason for the uncertainty is the limited environmental knowledge no matter the type of factor.

A second orthogonal source of environmental uncertainty is ***incomplete interaction with the environment***. Even if the environment is well specified, environmental uncertainty arises if the interaction with the environment is not as expected. SAS interacts with the environment through sensing and effecting. If sensing or effecting fails or returns inaccurate or noisy results, then the environmental uncertainty would increase.

We reorganized the common sources of environmental uncertainty, but the existing source terms and descriptions are cited in Table 2.8 for reference.

Figure 2.9 shows the summary of the sources of the environmental uncertainty and their relationships. The environmental uncertainty is cause by two primary sources, 'limited environmental knowledge' and 'incomplete environmental interaction'. Environment changes dynamically and sometimes its state is non-deterministic. It makes the environmental state unpredictable before runtime of the SAS. Con-

[5]We also surveyed definitions of "uncertainty" and "environmental uncertainty," but these were not included in this paper due to lack of space. Please refer to our website[3]

Table 2.8: Sources of environmental uncertainty

(Reorganized) Source	**(Existing) Source**	**Existing description (P22, P94, P102)**	**Existing terms (P22, P94, P102)**
1. Uncertainty from limited environmental knowledge	Overall (Cyber factor)	"current operating conditions, which are continuously invalidated due to changes" [P22] "the context of execution changes" [P22] "the environment conditions are close to a change" [P102] "Events and conditions in the environment that cannot be anticipated" [P94]	"Uncertainty in the context" [P22,P102], "Uncertainty of parameters in future operation" [P22,P102], "Unpredictable environment" [P94]
	- Physical factor	"the effect of physical world on the software is a subset of context, which was described in the previous source (uncertainty in the context)" [P22]	"Uncertainty in cyber-physical systems" [P22,P102]
	- Human factor	"the behavior of the crew (human) may be very unpredictable" [P22]	"Uncertainty due to human in the loop" [P22,P102]
2. Uncertainty from incomplete environmental interaction	Inaccurate sensor	"A sensor ... may return a slightly different number every time ..., even if the actual value ... is fixed." [P22] "Random and persistent disturbances that reduce the clarity of a signal" [P94]	"Uncertainty due to noise" [P22,P102], "Sensor noise" [P94]
	Sensor failure	"When a sensor cannot measure or report the value of a property" [P94]	"Sensor failure" [P94]
	Inaccurate effector	"system's ability ... is not only a function of the accuracy of its software, but the precision in the physical steering components (actuator)" [P22] "An adaptation that alters the execution environment in unanticipated ways" [P94]	"Uncertainty in cyber-physical systems" [P22,P102], "effector" [P94]
	Effector failure	"an actuator ... can either fail during an adaptation or ... introduce adverse effects upon the execution environment" [P94]	"effector" [P94]

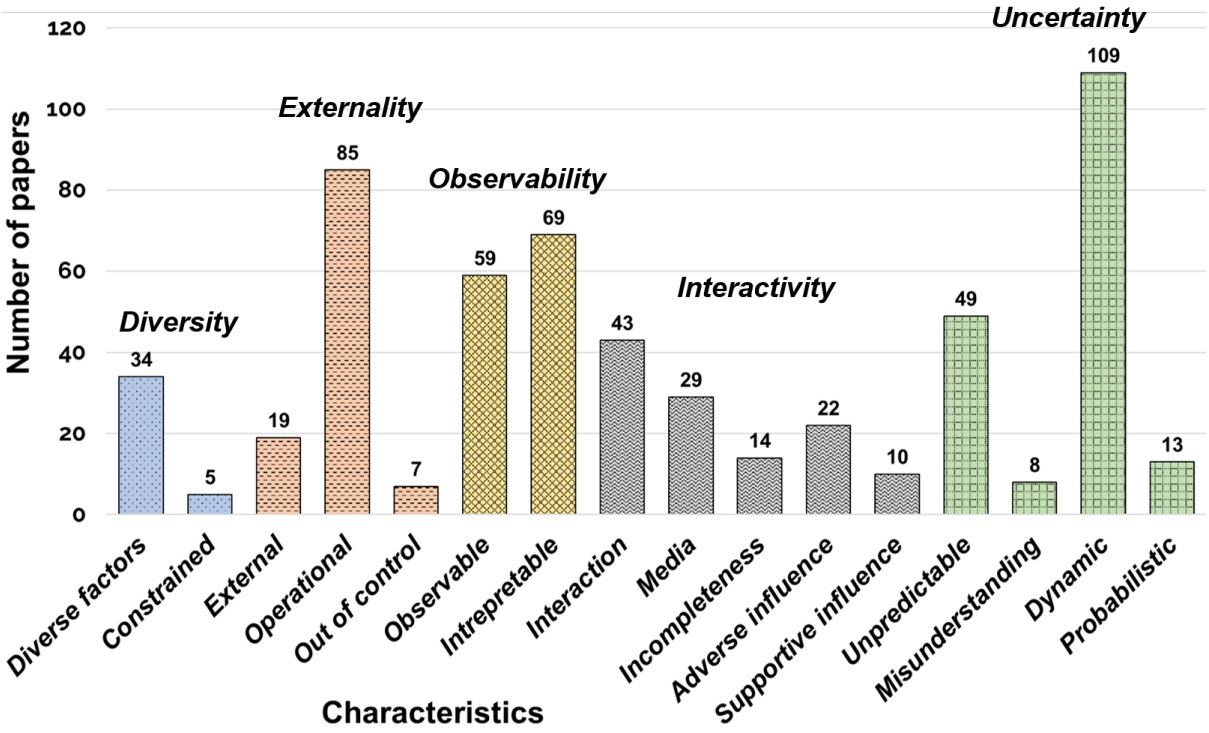


Figure 2.8: Number of mentions of characteristics of an SAS environment

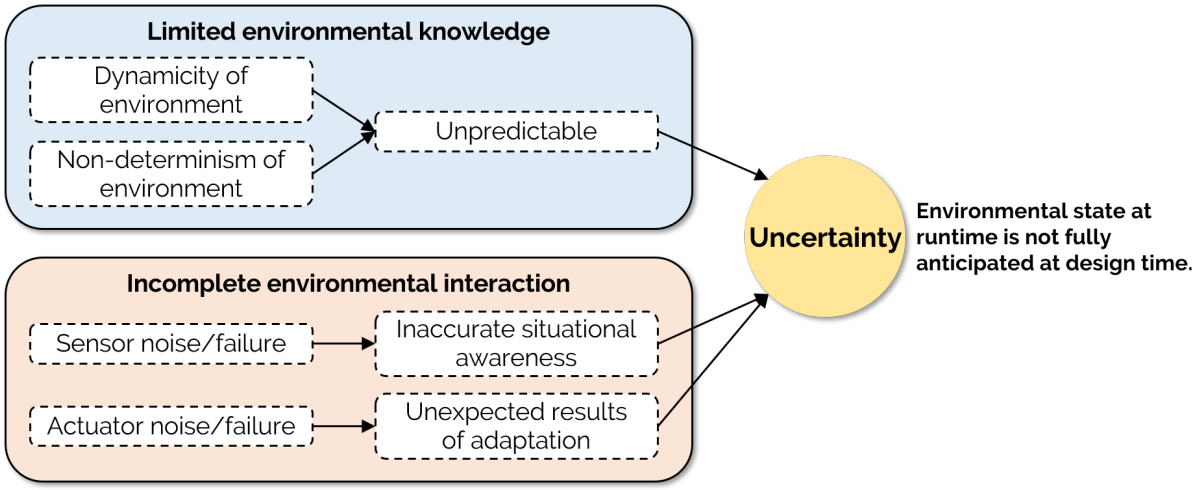


Figure 2.9: Environmental uncertainty sources

sequently the environment becomes uncertain to engineers. On the other hand, the SAS interacting with the environment observes the environment using noisy and faulty sensors. It makes the system recognize the environmental situation inaccurately. In addition, SAS affects the environment through inaccurate actuators, hence it is hard to accurately expect the environmental effect of the SAS actions. This inaccuracy makes the environment uncertain to both engineers and the SASs.

In Figure 2.10, we also analyzed how many primary studies addressed each source of environmental uncertainty. In these sources, environmental uncertainty caused by limited environmental knowledge was addressed more than uncertainty from incomplete interaction. However, limited knowledge about the human environmental factor was rarely addressed compared to the others. With regard to the sources of

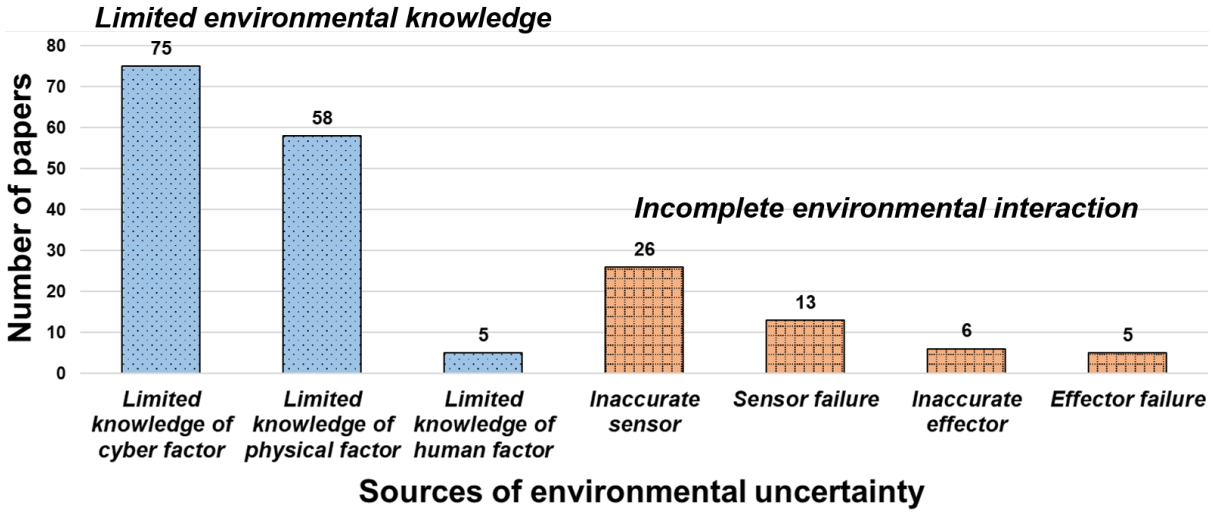


Figure 2.10: Frequency of addressing each source of environmental uncertainty

incomplete interaction, the sources related to sensors were relatively familiar to researchers, as evidenced in the writing, more so those related to effectors. In noting the trends, we must also acknowledge that even if various studies are addressing "environmental uncertainty," their use of this term does not necessarily rely on the same source. Therefore, researchers need to specifically explain their concerns regarding a particular source of environmental uncertainty to prevent misinterpretation.

2.5 Review Result 2: Models of the Environment

2.5.1 Environment Modeling Methods

A model is an abstraction of a subject that represents its important features, and so examining existing environment models allows us to find important features of the environment. In RQ4-RQ6, we provided an analysis of reference environment models. We found 14 unique models that represent the environment of an SAS from the 128 primary studies, and these are listed in Table 2.9. If a paper named the model, then the name is presented in the table; otherwise, a descriptive name we created for the model is listed. All the models provide an abstraction of the environment of an SAS, but their representations varied depending on the purpose of the modeling and the authors' perspectives. In addition, the formalism of the model was decided based on the authors' purpose. Some models followed standardized formalisms, while others were created using the authors' modeling languages or rules. These details are summarized in the table. While the models are not explained individually in detail (the reader is directed to the original reference for this information) due to a lack of space, the insights obtained from their analysis (modeling process and modeling effort) are shown.

We summarized the modeling processes for each model[3] and noticed common milestones for the modeling of the environment of an SAS. The milestones were as follows:

- Modeling the system boundary and environmental factors
- Modeling the environmental impact on the system goal
- Modeling interfaces of the system-environment interactions

Table 2.9: Models of the environment of SAS

ID	Model name	Representation	Modeling purpose / usage	Formalism	Ref.
M1	Interactive app model	Program, program interface, environment, environment interface, uncertainty, configuration	SAS testing environment	-	P17
M2	RELAX-marked conceptual environment model	Environment, environmental factors, sensors, etc.	Uncertainty-aware requirements elicitation	UML class diagram	P25
M3	PRISM stochastic environment game player model	System, environment, sensor model	Uncertainty-aware formal analysis	MDP	P28
M4	Environment model of Tropos4AS	Environmental artifacts, relationships to system agents	Testing environment code generation	UML class diagram	P36
M5	DTMC environment model	Stochastic environmental change	Optimal adaptation decision making	DTMC	P44
M6	Environmental constraint graph	Environmental states and their correlations	Improving model checking validity	Graph	P52
M7	Learning Petri net environment model	Environmental states	Formal analysis of SAS behavior and environment	Petri net	P63
M8	Environment domain model	Environmental state changes responding to system actions	Runtime behavior model revision	-	P66
M9	Interactive state machine and uncertainty specification	Environmental change, sensor and actuator noise, and environmental constraints	Uncertainty-aware and realistic verification	State machine	P72
M10	Ragnarok uncertainty genome	Numeric information of uncertainty sources	Exploring adverse environmental conditions	-	P82
M11	Game of testing environment model	Environmental state change responding to system actions	Environment model learning for runtime testing	MDP	P97
M12	Contextual variable dependency tree	Contextual variable states and their dependencies	Environment-aware requirements elicitation	Tree	P124
M13	System-environment interaction state model	Interactions between a software system and environment	Optimal adaptation decision making	State machine	P127
M14	Environment configuration variability and reconfiguration model	Environment situation variabilities and reconfiguration process	Environmental condition test case generation	-	P128

- Modeling the variability of the environment

All 14 modeling processes included at least one milestone. The first milestone was identifying the system boundary and enumerating the environmental factors that are outside of the system boundary. The second milestone focused on the goal of the SAS and modeled how the environment affects the goal. The third milestone highlighted the boundary between the SAS and the environment. It represented how the SAS and the environment utilize their interfaces, such as sensors and actuators. The fourth milestone modeled the variability of the environment. It expressed how the environment is able to change itself over time or is changed by the SAS. It is not necessary to achieve all the milestones, and they do not need to be achieved in a sequential order. The choice of milestones depends on the modeling purpose.

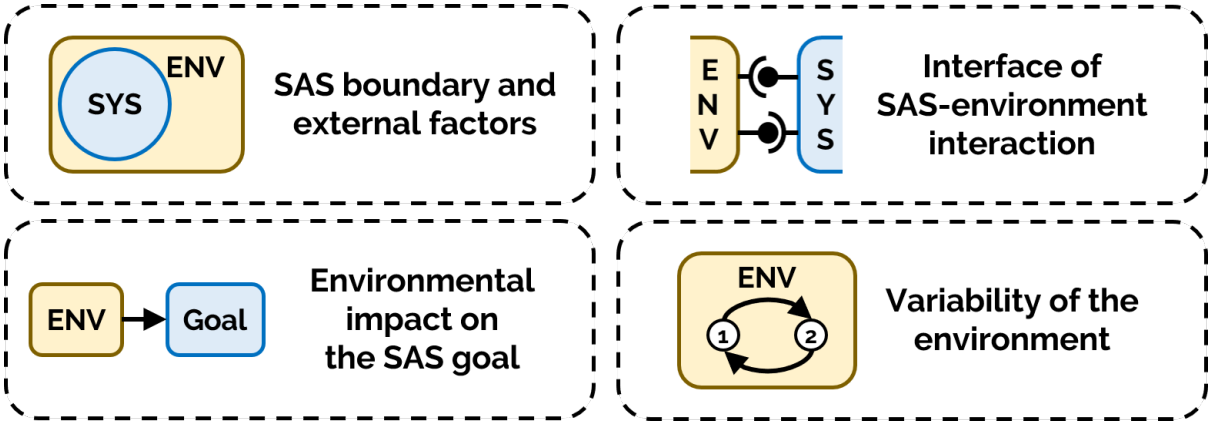


Figure 2.11: Four major perspectives of environment modeling

Based on the findings, not restricted to specific RQs, we also learned the following four common perspectives of primary studies for specifying or modeling the environment of an SAS shown in Figure 2.11.:

- *SAS boundary and external factors*: Identifying a system boundary is essential in defining the environment; identifying the external environmental factors then follows. This perspective guides engineers to clarify a boundary of an environment of SAS under consideration.
- *Environmental impact on the SAS goal*: Understanding how the environment affects the SAS goal is important to clarify the purpose of adaptation. This perspective guides engineers to elicit purposes and appropriate methods of adaptation.
- *Interface of the SAS-environment interaction*: Interfaces between the environment and the SAS, such as monitored environmental variables, actuating variables of the SAS, or specification of incomplete interaction (e.g., noise or failure), should be identified. This perspective helps to define an environment in the view of an SAS. It also specifies the limited amount of information about the environment and control capability over the environment that the SAS can have.
- *Variability of the environment*: Change of an environment over time or by the SAS should be identified for analysis by the SAS in the environment. This perspective helps to enumerate possible environmental states that an SAS will encounter during runtime. It also reveals the insufficiency of domain knowledge of SAS engineers and guides to define the degree of adaptiveness required for the SAS.

Although these four perspectives of environment specification or modeling were not always fully covered in each primary study, these must be sufficiently understood to have concrete knowledge about a specific environment of an SAS. These four common perspectives will help researchers sufficiently consider various aspects of the environment throughout the whole development process of the SAS and in the modeling.

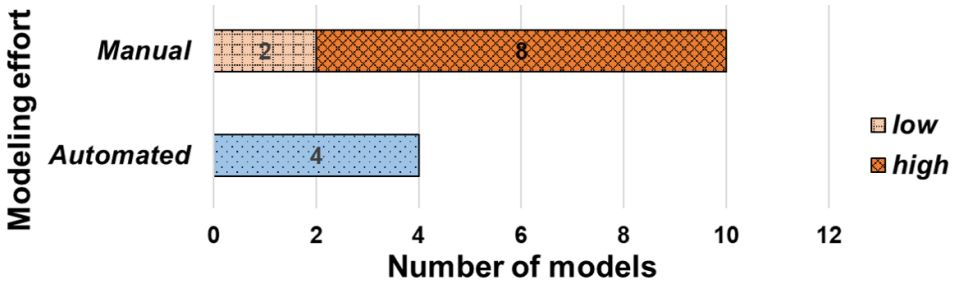


Figure 2.12: Modeling efforts of the environment models

We also examined the modeling efforts for each model, and these are summarized in Fig. 2.12. We divided the modeling efforts into automated and manual modeling. Automated modeling generated environment models automatically through the use of data by their methods (M5, M6, M8, and M11). Manual modeling was divided into two cases. The first case (high) is when significant expert-level environment knowledge, such as how environment behaves or which environmental conditions are expected, is required (M2, M3, M4, M7, M9, M10, M13, and M14). The second case (low) is when modeling can be completed with assistance, such as data, without significant knowledge (M1, and M12). Among the 14 models, only four were modeled automatically. The others were manually modeled. It is natural for engineers to build models manually for their purposes. However, the fact that most manual models require significant environmental knowledge suggests that the results of many engineering techniques using environment models can vary, depending on the quality of the engineer's knowledge.

2.5.2 Application of the Environment Models

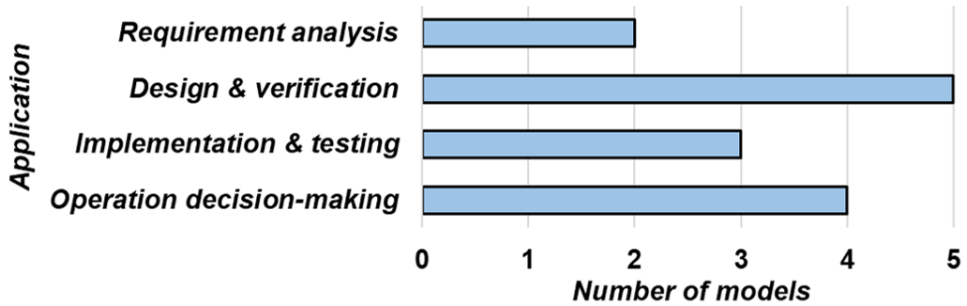


Figure 2.13: Application of the environment models

In answering this RQ, we examined how the environment models were used. We summarized the applications of the models in Fig. 2.13. We categorized the four usages of the models. The first was requirement analysis. Some environment models were used to explicitly identify environmental factors and elicit requirements they affected (M2 and M12). Another application was using the environment models as verification environments to mimic the actual environments of SASs (M3, M6, M7, M9, and M10). This was the most common usage. Another way to use the environment model was in generating

testing inputs for an SAS (M1, M4, and M14). The environment models of verification and testing were used to explore failures of an SAS that were triggered by the environment. The last application of the models was for the decision making of an SAS during operation (M5, M8, M11, and M13). The environment models were generated or updated during runtime, and they helped an SAS make optimal decisions in the runtime environment. The usage of each model is presented in Table 2.9.

We also summarized techniques that support leveraging the models but did not present them here due to a lack of space (they are available on our website[3]). However, one point that we would like to share here is that a common supportive technique of 11 models was simulation, which is regarded as the most fundamental use of environment models.

2.5.3 Expressiveness of the Environment Models

Finally, we examined how the environment characteristics (revealed in answering RQ2) were represented in the models. Fig. 2.14 shows the analyzed results for each characteristic (the details of each model can be found on our website[3]). For diversity (Fig. 7a), five models (M2, M4, M6, M10, and M12) required explicit modeling of each environmental factor. They highlighted the independence of the factors and could also represent the interaction among the factors. The other models implicitly showed that an environmental condition comprising diverse variables. For externality (Fig. 7b), 10 environment models (M1-M6, M10-M12, and M14) were decoupled from the system model. However, the other models were coupled with the system model, and externality was implicitly a part of their modeling process. For observability (Fig. 7c), only three models (M1-M3) explicitly described how the environment is monitored by an SAS representing sensor interfaces for environment observation. The others did not show an observation mechanism in the model but just assumed it.

For interactivity (Fig. 7d), all the models illustrated interactions between the environment and the SAS, but the direction of interaction influence can be in either direction. First, environmental conditions can affect the SAS; second, SAS behaviors can affect the environment. Nine models (M2-M7, M10, M12, and M14) represented only how the environment affects the system. They showed how the SAS goal is affected by or how the SAS reacts to the numerous environmental conditions. Only five models (M1, M8, M9, M11, and M13) represented two-way interactions. They modeled how the environment is changed by the SAS's behaviors, in addition to the SAS's reaction to the environment. When modeling the interactions, the incompleteness of the environment was also represented in some models. Among the models that expressed environmental influence on the SAS (Fig. 7d-1), only four representations of inaccurate sensors (M1, M3, M8, and M9), such as sensor noise, and one representation of sensor failure (M10) were found. Among the models expressing the SAS behavior's influence on the environment (Fig. 7d-2), only one representation of an effector or actuator possibly being inaccurate was found (M9). This result demonstrates that, so far, most models assume ideal interactions.

With regard to uncertainty (Fig. 7e), although there may be various ways to represent this in the environment, we included how models represented the variability of the environment because most models did this. Twelve models (M1, M3, and M5-M14) explicitly represented the variability of the environment, but two models (M2, and M4) just assumed the environmental condition can vary over time and did not represent it. Among the 12 models (Fig. 7e1), six models (M1, M7-M9, M11, and M13) represented how the environment responds to the SAS operation, and three others (M3, M5, and M14) modeled autonomous changes in the environment over time. These nine models usually specified environmental states and reactive or autonomous state transitions. The remaining three models (M6, M10, and M12) represented variability as an enumeration of possible environmental states. In answering

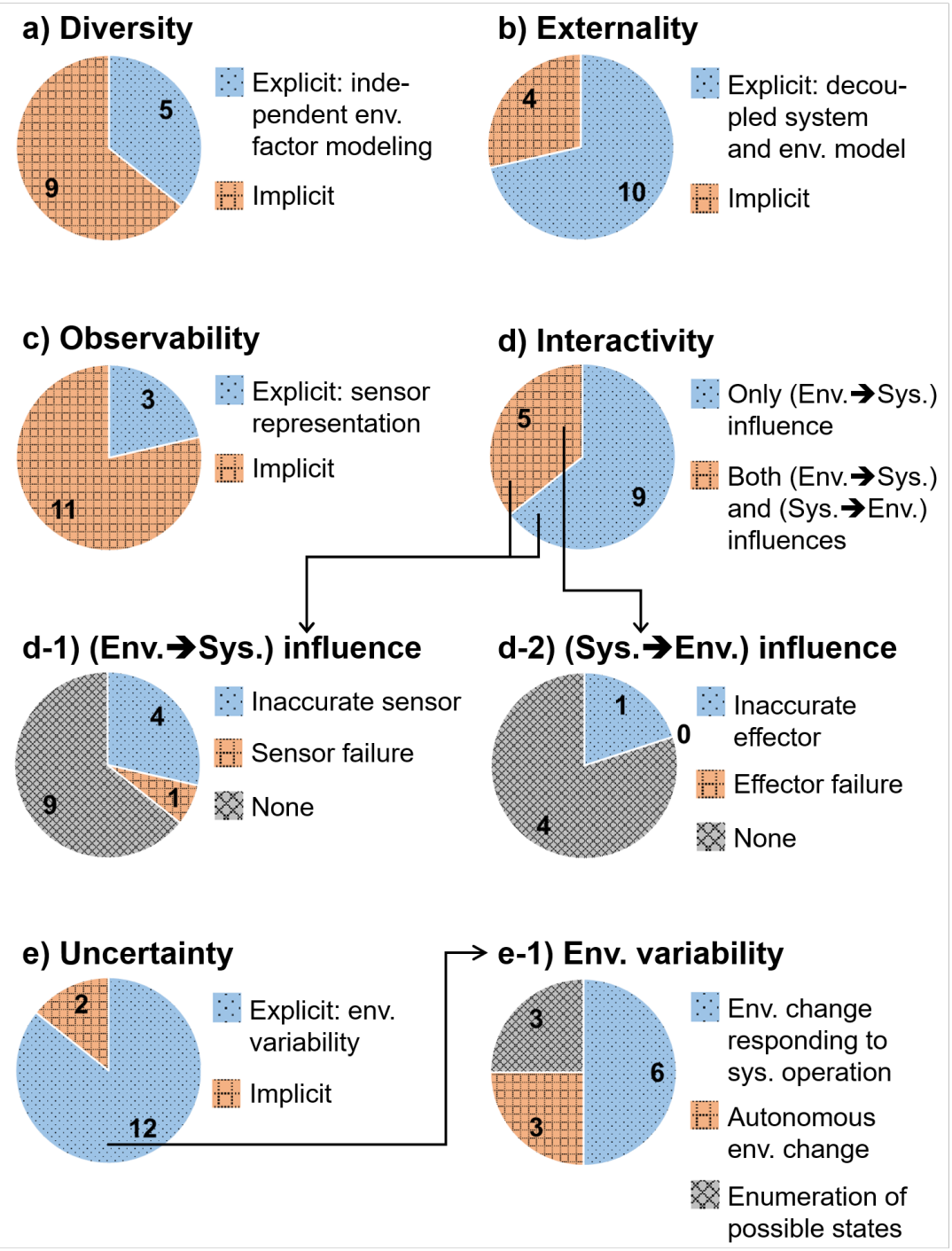


Figure 2.14: Representations of the SAS environment characteristics in the models

RQ6, we found that every model had a unique expression for the characteristics of the environment depending on the perspective, and we showed the trends of those expressions in this work.

Finally, we also identified some research challenges and limitations of the existing environment modeling for SAS as follows:

- *Limited consideration of various environmental characteristics*: Few papers systematically identified the characteristics of the SAS environment prior to this work, so the various characteristics of

the environments were often not explicitly expressed. Future modeling should reflect the diverse characteristics and perspectives of the SAS environment.

- *Limited consideration of various sources of environmental uncertainty*: Although there are various sources of environmental uncertainty, existing models did not represent them comprehensively. Future research should also address complex environmental uncertainty in which various sources are combined.

- *Considerable manual effort and domain knowledge required for modeling*: Adaptations based on environment models are increasing, but they still rely on manual models and domain knowledge. For the effective use of the environment model, additional research on automated or data-driven model generation is needed.

2.6 Comparison of Related Works and the Thesis

This section introduces environment modeling approaches related to our environment modeling approach that was collected in Chapter 2.

The environment models used to verify the goal achievement of the CPS controller under analysis in the physical environment often represent how the environmental state changes by the CPS actions. Figure 2.15 shows the definition of environment model for CPS goal verification. The environment model is the transition function of the state observed by the CPS controller. The state is an environmental state that is sensed through the CPS sensors. Because the CPS operates in the physical environment, the sensed environmental state represents the CPS state in the environment. Based on the observed state, the CPS controller decides on an action. The next state is decided on the current state and current CPS action. Therefore, the environment model abstracts the state transition, so the environment model can also be called a state transition function.

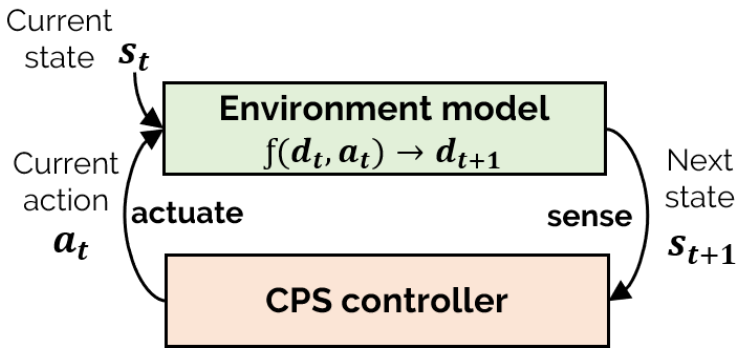


Figure 2.15: Environment model for CPS goal verification

In the SLR of Chapter 2, 14 reference environment models were collected. Among them, we found eight environment models representing the state's transition function observed by the CPS controller. This section introduces the eight environment modeling approaches.

A. Reichstaller, et al. proposed a testing environment in which human testers can dynamically interact with the system [38]. The environment model is given the action of the system under test and returns the next state of the system and a reward that quantifies the testing criteria satisfaction. The interaction between the environment and the system under test is modeled as a closed-loop interaction,

but the human testers can also inject testing actions during the system execution. This study provided the testing environment framework, but the engineers should adopt a domain-specific environment of the system, and not a domain-general solution for the environment modeling is provided.

W. Yang, et al. proposed an environment model that represents sequential state transition of the CPS, such as a robot vehicle [39]. The purpose of the environment modeling is the realistic consideration of the environmental constraints during the formal verification of the CPS behavior. Engineers model how the environmental state is changed after a specific CPS action is executed. In addition, the environmental uncertainties, such as the sensor noise or the unexpected consequence of a CPS action, are explicitly specified in the state transition.

Y. Qin, et al. proposed a model representing the interaction between the CPS and environment [27]. The model explicitly embraces the concept of environmental uncertainties. The first uncertainty specified in the model is the wrong recognition of the environmental state by the CPS. The second uncertainty specifies the unexpected effect of the CPS actuation on the environmental state. This model comprehensively captured uncertain interaction between the CPS and its physical environment. However, the environment model is conceptual, so engineers are needed to utilize a high-fidelity simulator to concertize the environment model in practice.

Z. Ding, et al. modeled the environmental state transition using an extended Petri-Net [24]. The environmental state transition and the CPS action decision model are tightly coupled in a single Petri-Net model. When the skeleton of the Petri-Net model is developed, the details of the model can be automatically learned to form the operation data. However, modeling the initial Petri-Net model still requires expert-level knowledge of both Petri-Net language and the interaction between the CPS and the physical environment.

Moeka Tanabe, et al., and D. Sykes, et al. modeled the state transition of the CPS using a state machine [40, 28]. A state is modeled as an environmental state, and the transition between the states is triggered by the CPS's actions. The model is used to abstract the high-level behavior of the CPS because there is a state explosion problem in the state machine.

J. Camara, et al., and G. A. Moreno, et al. modeled the environment state transition model using Markov Decision Process (MDP) [41, 42]. The environment model MDP is implemented in PRISM model checker language. The models were used to predict the next state of the environment affected by the system action for the better runtime decision-making of the system. The structure of the environment model is made by engineers, but the time-series forecasting algorithms automatically calculate concrete values in the MDP environment model.

The next section discusses the limitations of these related environment models from the perspectives of the challenges of the environment modeling described in Chapter 1. We compared the related environmental modeling approaches based on the challenges of manual modeling mentioned in Chapter 1 in Table 2.10.

From the perspective of the modeling of a complex environment, all existing approaches can model the environmental state as a discrete state by significantly abstracting the environment to reduce the complexity. Some approaches can represent continuous environmental state variables to represent more complex environments. In addition, some models also expressed environmental uncertainty, such as inaccurate monitoring of the state of the environment on the system and stochastic state variations.

From the compounding error problem perspective, in most approaches, small errors are accumulated during the long simulation. It makes it difficult to perform an accurate simulation. Therefore, engineers can simulate most models only for small simulation time steps. The environment models should be

updated for each simulation time step for long-term simulation. Some models are proposed for long-term simulation, but the approaches assume domain experts give an accurate environment model.

From the manual modeling effort perspective, most modeling approaches assume or require expert-level domain knowledge for accurate environment model generation. For example, the concrete environmental state transition should be given as a function or many rules. Some models require engineers to design the environmental state transition mechanism skeleton. However, the requirement is unachievable to human engineers when the CPS and its environment are very complex. Some modeling approaches provide some data-driven automation methods to learn some part of the environment model, but large manual effort is still required for modeling.

Table 2.10: Limitations of the related works

Modeling approach	Complex environment modeling		Solving compounding error	Manual modeling effort	
	Continuous state variable	Environmental uncertainty		Required domain expertise	Data-driven automation
[38]	O	X	O	High	None
[39]	X	O	O	High	None
[27]	O	O	O	High	None
[24]	O	O	X	High	Partial (Petri-Net learing)
[40]	X	X	X	High	Partial (Differential learning)
[28]	X	O	X	High	Partial (Rule learning)
[41]	X	O	X	Medium	Partial (Simple data analysis)
[42]	X	O	X	Medium	Partial (Simple data analysis)
This	O	O	O	Low	Largely (Imitation learning)

Our approach automatically generates the environment model using Imitation Learning (IL) compared to the related environment modeling approaches. It only requires little domain knowledge to define the environment model's input and output. The environmental state transition mechanism is learned from solely CPS operation data. In addition, the generated model represents an environmental state as a continuous variable, and the environmental uncertainty, such as sensor noise, is embraced in the environment model. The environment model is generated for the long-term simulation of solving the compounding error problem.

2.7 Summary

In this chapter, we introduced CPS and environment modeling. We described the CPS and its controller of this thesis's interest and compared the concept of CPS with the related system types (i.e., SAS

and SoS) to comprehensively understand CPS. In addition, we systematically surveyed the environment modeling studies. Based on the SLR results, we concertized the concept of the environment and analyzed some related environment models. Finally, we compared the environment models abstracting the state transition of the real environment based on the challenges of the manual environment modeling to show their limitations comparing to the thesis. In summary, the related environment modeling approaches are not appropriate to model the complex environment, be used for the long-term simulation, and be used with less domain knowledge. We will provide a novel model generation approach to solving the challenges of manual environment modeling in the following chapters.

Chapter 3. Formal Framework of CPS Goal Verification

3.1 Introduction

This chapter introduces a mathematical framework for modeling how the CPS under analysis interacts with its environment to achieve its goals. First, the interaction between the CPS controller and its environment is modeled. Second, embracing the model, we define the formal framework of CPS goal verification. Third, based on the formal framework of CPS goal verification, we formally define the environment model generation problem of this thesis. The formal problem definition can be also shared to the related environment modeling studies.

3.2 CPS-Environment Interaction Model

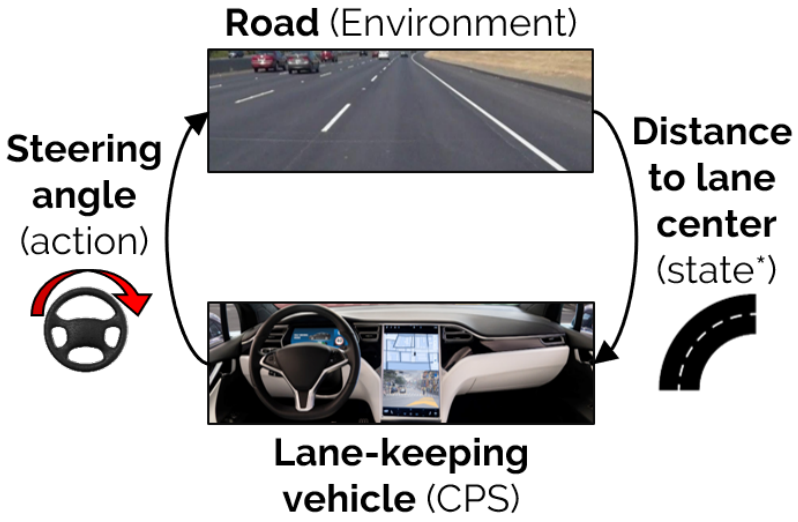


Figure 3.1: Closed loop interaction between CPS and environment

In this section, we first formally model the interaction between the CPS controller and its operational environment. To limit the scope of modeling and divide the modeling concerns, in this dissertation, we assume the CPS and the environment interact with each other in a closed loop. This means that there is no external factor that affects to the CPS and the environment. Only the CPS affects the environment, and only the environment affects the CPS. In short, we call the interaction CPS-ENV interaction.

Figure 3.1 shows an example of the CPS-ENV interaction. Here the CPS controller is the lane-keeping system of an autonomous vehicle, and the environment is the road observed by the lane-keeping system. The lane-keeping system observes the vehicle's state by sensing its road environment. Especially, it observes the distance to the lane center. Based on the observed data, it decides a steering angle towards the lane center. The steering angle is actuated and finally triggers change in the distance to the lane center that will be observed at the next time step.

Like the example above, a CPS achieves its goals by interacting with its physical environment.

Specifically, starting from an initial state of the environment, the CPS software controller observes the state and decides an appropriate action to maximize the likelihood of achieving the goals. Then, taking action causes a change in the environment for the next step, which the CPS will observe again to decide an action for the next step. To formalize this process, we present a novel *CPS-ENV interaction model* inspired by Markov Decision Process [43] that models an agent's sequential decision-making process under observation over its environmental states.

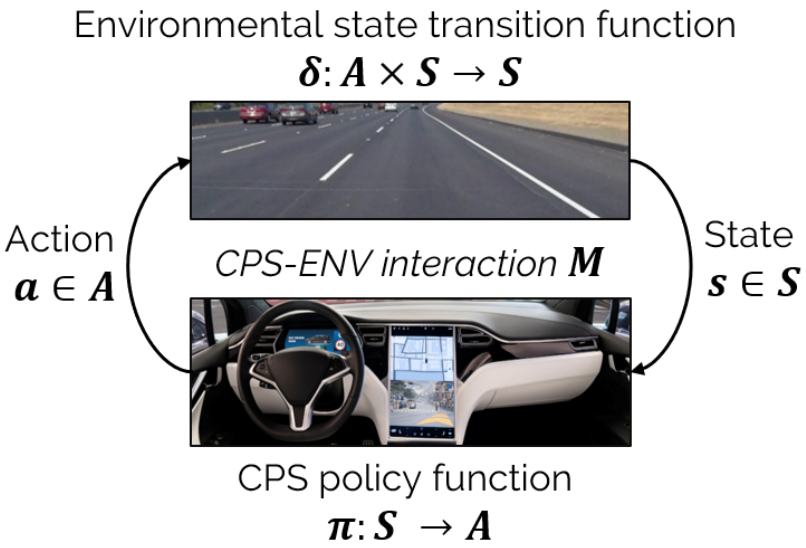


Figure 3.2: CPS-ENV interaction model

Specifically, a *CPS-ENV interaction model* is a tuple $M = (S, A, \pi, \delta, s_0)$ as shown in Figure 3.2, where

- S is a set of observable states of the environment under consideration, A is a set of possible CPS actions,
- $\pi : S \rightarrow A$ is a policy function that captures the software controller of the CPS,
- $\delta : S \times A \rightarrow S$ is a transition function that captures the transitions of environmental states over time as a result of CPS actions and its previous states[1],
- and s_0 is an initial environmental state.

For example, starting from s_0, the CPS makes an action $a_0 = \pi(s_0)$, leading to a next state $s_1 = \delta(s_0, a_0)$. By observing s_1, the CPS again makes the next action $a_1 = \pi(s_1)$, and so on.

In the *CPS-ENV interaction model* M, S and A are defined by the CPS sensing and actuating capability. S is limited by the sensors equipped in the CPS. If a sensor can observe an external integer variable ranging from 1 to 100, then the $S = \{1, 2, \ldots, 99, 100\}$. If there are multiple sensors, an $s \in S$ is a vector including multiple sensing variables. In the same way, an $a \in A$ is also a vector of actuating variables of the CPS. Therefore, the size of S and A is proportional to the number of sensing and actuating variables and the ranges of the variables.

[1]Though we use deterministic policy and transition functions for simplicity, they can be easily extended in terms of probability density, i.e., $\pi : S \times A \rightarrow [0, 1]$ and $\delta : S \times A \times S \rightarrow [0, 1]$, to represent stochastic behaviors if needed.

Based on the S and A defined by the CPS's sensing and actuating capability, the π and δ are the functions abstracting the CPS controller and the environmental state transition. Here, the π is the software controller under development, so the π is an engineering artifact. On the other hand δ is a mathematical representation of the real environment, so it just abstract an unknown state transition observed by the CPS controller. Therefore, δ represents the 'environment model' in this dissertation.

Based on the *CPS-ENV interaction model*, we formally define how the CPS controller interact with the environment and what the 'environment model' in this dissertation means.

3.3 Formal Framework of CPS Goal Verification Process

In this section, we introduce the formal framework of CPS goal verification process using the *CPS-ENV interaction model* M defined in the previous section. It formally abstracts the typical process of statistical CPS goal verification.

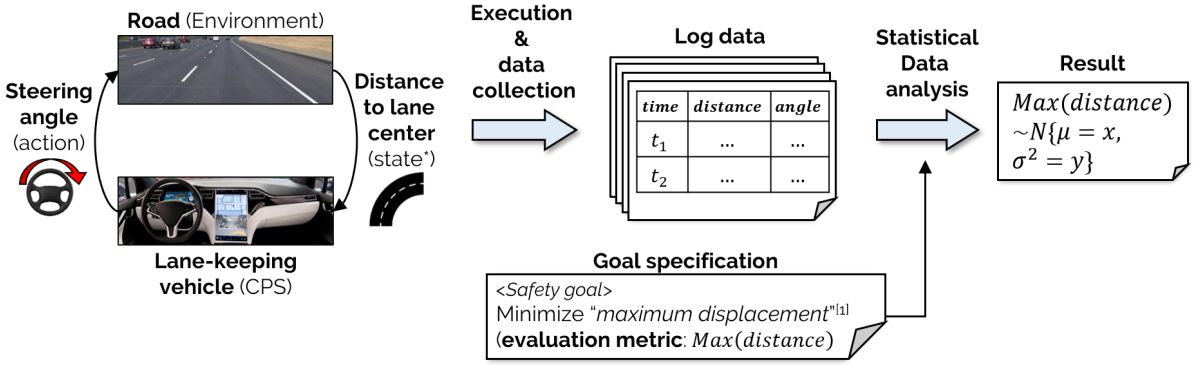


Figure 3.3: An example of CPS goal verification process

The CPS goal verification process is summarized as follows as shown in Figure 3.3. The engineer develop CPS controller first and run it in the environment. For example, suppose there is an engineer developing a lane-keeping system of an autonomous vehicle. The lane-keeping system is defined as a system in which the car receives the distance from the center of the lane as input and determines the angle of the steering. The determined angle controls the car and it makes a change in the next state of the vehicle. The Lane-keeping system repeats this process to control the car. The CPS controller collects driving data that records the states and actions. This driving log data is the time series data of the environmental state observed by the CPS for each time stamp and the CPS actions determined by it. The accumulated driving log data is analyzed to assess the goals of the vehicle. For example, the engineer can quantitatively evaluate the safety goal of the lane-keeping system, minimizing the "maximum displacement" from the center of the lane. The result is a statistical analysis result of the goal evaluation metric.

We define a formal framework of the CPS goal verification process as shown in Figure 3.4. For a *CPS-ENV interaction model* $M = (S, A, \pi, \delta, s_0)$, we can think of a sequence of transitions $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} s_n$ over n steps where $s_{t-1} \xrightarrow{a_{t-1}} s_t$ denotes a transition from a state s_{t-1} to another state s_t of the environment by taking an action a_{t-1} of the CPS. More formally, we define a *trajectory* of M over T time ticks as a sequence of tuples $tr(M, T) = \langle (s_0, a_0), \ldots, (s_T, a_T) \rangle$.

Since a trajectory of a *CPS-ENV interaction model* concisely captures the sequential interaction between the CPS under analysis and its environment, one can easily verify whether CPS goals are

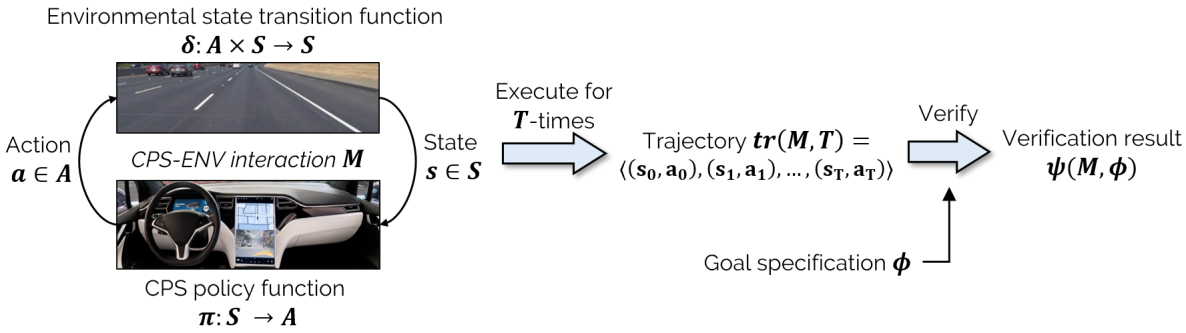


Figure 3.4: Formal framework of CPS goal verification process

achieved or not by analyzing the trajectory. Specifically, let ϕ be a requirement that precisely specifies a goal under verification. The achievement of ϕ is quantifiable. For a *CPS-ENV interaction model* M, the verification result of ϕ for M, denoted by $\psi(M, \phi)$, is computed by evaluating the achievement of ϕ on the trajectory of M.

Depending on the type of ϕ, the value of $\psi(M, \phi)$ can be Boolean (expressing the success or failure of a requirement with clear-cut criteria) or Float (expressing the measurement of an evaluation metric of ϕ). For example, one of the evaluation metrics of the lane-keeping requirement is the distance the vehicle is away from the center of the lane. As a result of the verification of the lane-keeping goal, the average or maximum distance from the center is computed.

3.4 Problem Definition of the Environment Modeling

3.4.1 Original Definition

Using the formal framework of CPS goal verification process, This section formally defines the problem of environment model generation of this thesis.

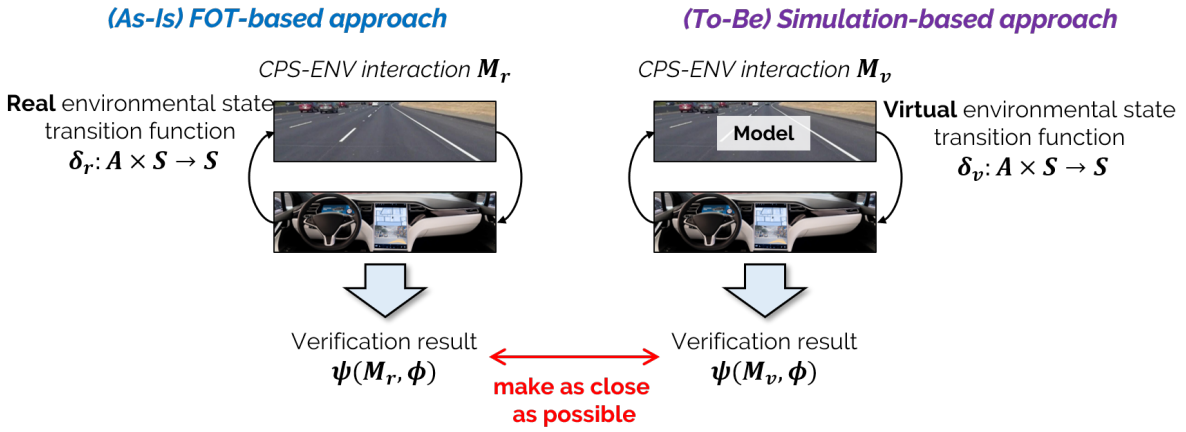


Figure 3.5: Formal problem definition of the environment model generation

The formal problem definition of the virtual environment model generation for CPS goal verification in this thesis is shown in Figure 3.5. The problem of virtual environment model generation for simulation-based CPS goal verification is to find an accurate virtual environment model that can

replace the real environment of the CPS goal under verification while maintaining the same level of verification accuracy. Specifically, for the same CPS under analysis, let a *CPS-ENV interaction model* $M_r = (S, A, \pi, \delta_r, s_0)$ representing the interaction between the CPS and its real environment (in FOT) and another model $M_v = (S, A, \pi, \delta_v, s_0)$ representing the interaction between the same CPS and its virtual environment (in simulations). Notice that we have the same S, A, π, and s_o for both M_r and M_v since they are about the same CPS[2], whereas δ_r and δ_v are different since they represent how the corresponding environments react to the actions performed by the CPS. For a requirement ϕ, we aim to have δ_v that minimizes the difference between $\psi(M_r, \phi)$ and $\psi(M_v, \phi)$. Therefore, the problem of virtual environment model generation for CPS goal verification is to find δ_v such that $|\psi(M_r, \phi) - \psi(M_v, \phi)|$ is the minimum.

The virtual environment model generation problem has three major challenges. First, the number of possible states and actions is often very large, making it infeasible to build a virtual environment model (i.e., represented by a transition function $\delta_v : S \times A \rightarrow S$) by exhaustively analyzing individual states and actions. Based on the definition of $\delta : S \times A \rightarrow S$, denoting $n(S)$ and $n(A)$ as the number of elements in S and A respectively, $n(S) \times n(A)$ transitions must be defined to have a complete transition function. The more possible environmental states and CPS actions, the more difficult defining all transitions completely. In addition, δ_v should be close to δ_r, which is unknown to engineers.

Second, since the virtual environment model continuously interacts with the CPS under analysis in a closed-loop, even a small difference between the virtual and real environments can significantly differ in verification results as it accumulates over time, the so-called compounding error problem introduced in the introduction chapter of this dissertation. This means that simply having a transition function δ_v that mimics the behavior of δ_r in terms of individual input and output pairs, without considering the accumulation of errors for sequential inputs, is not enough.

Third, generating δ_v should not be as expensive as using many FOTs; otherwise, there is no point in using simulation-based CPS goal verification. Recall that manually crafting virtual environment models in a high-fidelity simulator requires a lot of expertise, which takes longer than doing FOTs many times for having statistically significant verification results. Therefore, a practical approach should generate an accurate virtual environment model efficiently and automatically.

To address the challenges mentioned above, we suggest leveraging IL to automatically generate virtual environment models from only a small amount of data. The data is the partial trajectory of M_r, which can be collected from a few FOTs for the CPS under test in its real application environment. Since IL can efficiently extract how experts make sequential actions for given states from a limited amount of demonstrations while minimizing the compounding errors, it is expected to be an excellent match to our problem. Therefore for our problem, IL will extract δ_v, instead of π (which is the original goal of IL), that can best reproduce given trajectories of M_r (i.e., FOT logs). Generated δ_v may differ depending on the amount of the trajectory, so we analyze it in the experiment.

The proposed data-driven environment model generation using IL will be introduced in the following chapter.

3.4.2 Extended Definition

The problem definition introduced above formally specifies the goal verification process of a CPS controller π and the role of a virtual environment model δ_v. In this section, we extend the original

[2]Note that S can be the same for M_r and M_v because it is a set of *observable* states from the perspective of the CPS under analysis.

problem definition based on an assumption that the controller π is a configurable software embracing some configurable variables influencing the controller behavior. The original problem definition is subsumed in the extended problem definition.

The controller π is a software module that controls the CPS action according to the observed environment. Software engineers can develop the software controller with various control mechanisms like rule-based approach, PID controller-based approach, or machine learning-based control. In many cases, the engineers implement the control software with some configurable parameters that change the behavior of the controller. The engineers then optimize the configurable parameters (i.e., configuration) to better achieve the CPS goals. In other words, the engineers analyze and verify many CPS controller variations discriminated by the configuration setting then search the best setting to develop an optimized CPS controller.

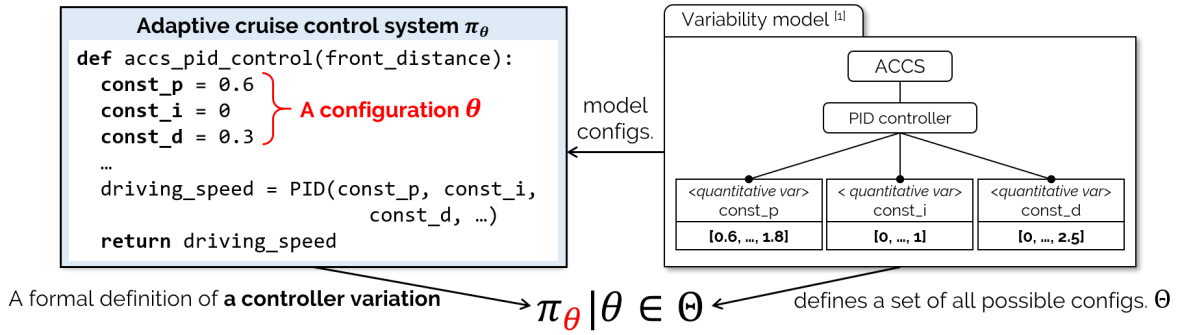


Figure 3.6: An example of configurable CPS controller

Figure 3.6 shows an example of a configurable adaptive cruise control system controller. The controller can be developed based on a PID controller which is one of the most widely used control strategy. The controller observes distance to the front vehicle and calculates driving speed to manipulate a vehicle. To calculate the driving speed, PID controller requires three constants that tune propositional, integral, and derivative controllers respectively. In this case, the sequence of three constants is a configuration, denoted by θ. All possible configurations of engineers interest can be described in a variability model shown in the right side of the Figure. Based on the model, a set of all possible configurations Θ is defined, and the controller configured by θ, an element of Θ is a specific controller variation, denoted by π_θ. Suppose the engineers verify all CPS controller variations of interest with the environment model-based simulation described in the original problem definition. For this purpose, the original problem definition is extended.

Specifically, $\pi_\theta : S \rightarrow A$ is a CPS software controller configured by a configuration θ that denotes the controller's variability point [44], where S is a set of environment states and A is a set of controller actions. is a sequence of quantitative, user-configurable variables [44] of the CPS software controller. (Note that the qualitative variables of θ are currently outside of our scope.) A quantitative variable can be either discrete or continuous [44], and the continuous variable can also be discretized by an unit interval. θ concisely specifies a certain version of π's decision-making logic, with the fixed possible input set S and output set A of π. Specifically, given two different θ_1 and θ_2, both $\pi_{\theta_1} : S \rightarrow A$ and $\pi_{\theta_2} : S \rightarrow A$, but the decided CPS actions $a_1 \leftarrow \pi_{\theta_1}(s)$ and $a_2 \leftarrow \pi_{\theta_2}(s)$ may be different for the same input $s \in S$.

We call controllers with different configurations (e.g., $\pi_{\theta_1}, \pi_{\theta_2}, \dots$) as variations of π. Engineers can change θ to make another variation π_θ under verification for specific engineering purposes, such as adaptation, optimization, repair, or evolution of π. We define a set of all possible θs Θ (i.e., $\theta \in \Theta$).

In many cases, Θ of a CPS controller is an infinite set because of the unlimited range or the continuity of a user-configurable variable. However, in practice, Θ under consideration is defined as a finite set by domain knowledge or constraints, so we will assume Θ is defined finitely in this study. Therefore, the engineer's goal is to verify all variations of π configured by $\theta \in \Theta$ as efficiently as possible.

Here, we investigate whether an environment model δ_v can be used to verify π's variations (i.e., $\{\pi_\theta | \theta \in \Theta\}$) that had not been used for the seed log collection of δ_v so that the engineers can *efficiently* verify all variations by collecting as small seed logs as possible by FOTs. To formally define the problem, we first extend the notations of our CPS-ENV interaction model M.

- $M_{r|\theta} = (S, A, \pi_\theta, \delta_r, \sigma_0)$ denotes CPS-ENV interaction between the real environment δ_r and the CPS controller π configured by θ.
- Let Θ_{tr} be the set of π's configurations used for generating a set of seed logs (training data) tr (i.e., tr is generated by monitoring π_θ for $\theta \in \Theta_{tr}$).
- $M_{tr|\theta} = (S, A, \pi_\theta, \delta_{tr}, \sigma_0)$ denotes CPS-ENV interaction between the virtual environment δ_{tr} generated using tr.

Using the extended notations, we define a problem of the virtual environment model generation for verification of CPS controller variations below.

- Suppose the finite set of CPS controller configurations under analysis Θ is given.
- A subset $\Theta_{tr} \subset \Theta$ indicates CPS controllers $\{\pi_\theta | \theta \in \Theta_{tr}\}$ used for seed log collection by FOTs.
- A method models or generates a virtual environment model δ_{tr} using the seed logs of $\{\pi_\theta | \theta \in \Theta_{tr}\}$.
- Given CPS goal under verification ϕ, we aims to minimize both (a) $\sum_{\theta \in \Theta} |\psi(|M_{r|\theta}, \phi) - \psi(|M_{tr|\theta}, \phi)|$ and (b) $|\Theta_{tr}|$.

Note that a controller variation π_θ whose $\theta \notin \Theta_{tr}$ is a controller whose behavior is never seen to δ_{tr}, so we call $\pi_\theta | \theta \notin \Theta_{tr}$ *unseen* CPS controller variation. (a) The point is to make the simulation-based CPS goal verification result as similar as possible to the FOT-based verification result for all (i.e., both seen and unseen) controller variations. (b) At the same time, the smaller number of the CPS controller variations under FOT for seed log collection $|\Theta_{tr}|$, the less laborious cost of CPS goal verification.

The extended problem definition subsumes the original problem definition. Specifically, if there is only one configuration under analysis or a controller is not a configurable controller so an engineer has only one unique controller under analysis, we can say $\Theta = \{\theta_1\}$ and $|\Theta| = 1$. In this case which there is only one θ under analysis, $\delta_{tr|\theta}$ of extended problem definition is always same with δ_v of the original problem definition. In summary, the original problem definition is extended for cases when there are more than two controllers (i.e., controller variations) under verification, so the original and extended problem definitions are the same when there is only one controller variation under verification.

3.5 Summary

In this chapter, we proposed *CPS-ENV interaction model* that formally abstracts the continuous interaction between a CPS controller and its environment. Using the model, we also proposed a formal framework of the CPS goal verification process. Based on the formal framework, we formally defined the problem of the environment model generation for CPS goal verification.

Chapter 4. Data-Driven Environment Model Generation

4.1 Introduction

To solve the difficulty of manually generating virtual environment models, we propose an automated data-driven environment model generation approach for CPS goal verification by recasting the problem of environment model generation as the problem of imitation learning. We call this novel approach *ENVironment Imitation* (*ENVI*). In machine learning, Imitation Learning (IL) has been widely studied to mimic complex human behaviors in a given task only from a limited amount of demonstrations [45]. Our approach leverages IL to mimic how the real environment interacts with the CPS under analysis from a small set of log data collected from FOTs. Since the log data records how the CPS and the real environment interacted, our approach can generate an environment model that mimics a state transition mechanism of the real environment according to the CPS action as closely as possible to that recorded in the log data. The generated environment model is then used to simulate the CPS software controller as many times as needed to statistically analyze the CPS goal achievement. Specifically, this chapter provides a systematic process and user-configurable parameters of *ENVI*.

In summary, this chapter introduces:

1) the detailed explanation of the process of our approach *ENVI*,

2) two environment model structures of *ENVI*,

3) three *ENVI* environment model generation algorithms extending three representative IL algorithms (BC, GAIL, and BCxGAIL),

4) and three quantitative best model selection criteria for validating the environment model made by *ENVI*.

The remainder of this chapter is organized as follows. Section 4.2 illustrates a motivating example. Section 4.3 provides background on IL. Section 4.4 shows the overview of ENVI. Section 4.5 proposes ENVI process in detail.

4.2 Motivating Example

In this section, we present a simple example of CPS goal verification to demonstrate a use case of our approach.

Consider a software engineer developing a lane-keeping system of an autonomous vehicle. The engineer aims to develop and test the vehicle's software controller (i.e., lane-keeping system) that continuously monitors the distance from the center of the lane and computes the steering angle that determines how much to turn to keep the distance as small as possible.

Once the software controller is developed, the engineer must ensure that the vehicle equipped with the controller continues to follow the center of the lane while driving. To do this, the engineer deploys the vehicle on a safe road and collect an FOT log, including the distance d_t and the steering angle a_t at time $t = 1, \ldots, T$ where T is a pre-defined FOT duration. Based on the collected data, the engineer can

quantitatively assess the quality of the lane-keeping system by calculating the sum of the distances the vehicle deviated from the center of the lane, i.e., $\Sigma_{t=1}^{T}|d_t|$. The quantitative assessment is used to verify precisely a goal of the system, i.e., whether $\Sigma_{t=1}^{T}|d_t| < \epsilon$ holds or not for a small threshold ϵ. Notice that, due to the uncertainties in FOT, such as non-uniform friction between the tires and the ground, the same FOT must be repeated multiple times, and statistical analysis should be applied to the results.

It takes a lot of time and resources to repeat the FOTs enough to obtain statistically significant results. To address this issue, the engineer may decide to rely on simulations. However, using high-fidelity and physics-based simulators, such as Webots [46] or Gazebo [47], is very challenging, especially for software engineers who do not have enough expertise in physics. It is not easy to accurately design the physical components of the system (e.g., the size of wheels and the wheelbase) and the road in the simulator so that the simulation results are almost identical to the FOT results.

Our approach, *ENVI*, enables the CPS goal verification without using such a high-fidelity simulator. The engineer can simply provide *ENVI* with the software controller (i.e., the lane-keeping system under analysis) and a small amount of FOT logs collected from the beginning, which is far less than the data required for statistically significant results using FOTs. Then *ENVI* automatically generates a virtual environment model that imitates the behavior of the real environment of the lane-keeping system; specifically, the virtual environment model can simulate d_{t+1} for given d_t and a_t for $t = 2, \ldots, T$ such that $\Sigma_{t=1}^{T}|d_t|$ calculated based on the virtual model is almost the same as the value calculated based on the FOTs. Therefore, by quickly re-running the simulation multiple times, the engineer can have statistically significant results about the quality of the software controller at little cost. Furthermore, if multiple software controller versions make different CPS behaviors, the virtual environment model generated by *ENVI* can be reused to verify the CPS goal achievements of new controller versions that have never been tested in the real environment.

The challenge for *ENVI* is automatically generating a virtual environment model that behaves as similar as possible to the real environment using a limited amount of data. To address this, we leverage *imitation learning* detailed in the following section.

4.3 Background: Imitation Learning

This section provides background knowledge of imitation learning that is required to understand the following sections.

Imitation Learning (IL) is a learning method that allows an agent to mimic expert behaviors for a specific task by observing demonstrations of the expert [45]. For example, an autonomous vehicle can learn to drive by observing how a human driver controls a vehicle. IL assumes that an expert decides an action depending on only the state that the expert encounters. Based on this assumption, an expert demonstration is a series of pairs of states and actions, and IL aims to extract the expert's internal decision-making function (i.e., a policy function that maps states into actions) from the demonstration [45]. We introduce two representative IL algorithms in the following subsections: Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL).

4.3.1 Behavior Cloning

Behavior Cloning (BC) infers the policy function of the expert using supervised learning [48, 49]. Training data can be organized by pairing states and corresponding actions in the expert's demonstration.

Then existing supervised learning algorithms can train the policy function that returns expert-like actions for given states. Due to the simplicity of the BC algorithm, BC can create a good policy function that mimics the expert quickly if there are sufficiently much demonstration data. However, if the training data (i.e., expert demonstration) does not fully cover the input state space or is biased, the policy function may not mimic the expert behavior correctly [49].

4.3.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [50] utilizes the idea of Generative Adversarial Networks [51] to evolve the policy function using iterative competitions with a *discriminator* that evaluates the policy function. Therefore, both the policy function and the discriminator are trained in parallel.

The policy function gets states in the expert demonstration and produces simulated actions. The discriminator then gets the policy function's input (i.e., states) and output (i.e., simulated actions) and evaluates how the policy function behaves like the real expert, as shown in the demonstration. The more similar the simulation is to the expert demonstration, the more rewarded the policy function is by the discriminator. The policy function is trained to maximize the reward from the discriminator.

On the other hand, the discriminator is trained using both the demonstration data and the simulation trace of the policy function. The state and action pairs, which is the input and output of the policy function, in the demonstration data are labeled as *real*, but the pairs in the simulation trace are labeled as *fake*. A supervised learning algorithm trains the discriminator to quantitatively evaluate whether a state and action pair is real (returning a high reward) or fake (returning a low reward).

After numerous learning iterations of the policy function and the discriminator, the policy function finally mimics the expert well to deceive the advanced discriminator. GAIL uses both the expert demonstration data and the simulation trace data of the policy function generated internally, so it works well even with small demonstration data [50]. However, because of the internal simulation of the policy function, its learning speed is relatively slow [52].

4.4 Environment Imitation Overview

This section provides ENVI, a novel approach to the problem of environment model generation for CPS goal verification, defined in Chapter 3. We solve the problem by using IL to automatically infer a virtual environment state transition function from the log recorded during the interaction between the CPS under test and its application environment. The original IL aims to generate (train) the system's policy function π in a given environment, but we leverage IL for ENVI with the aim of generating the environment δ_v of the CPS software controller under analysis. In this context, the real application environment is considered an "expert," and the FOT log demonstrates the expert.

Figure 4.1 shows the overview of the environment model generation and simulation-based CPS goal verification process using our approach. It is composed of five main stages: (1) collecting seed logs from FOTs, (2) defining an environment model structure based on environment characteristics, (3) training environment models from the seed logs using an IL algorithm, (4) selecting the best environment model, and (5) verifying the given CPS goals using the best environment model. Each of the stages is detailed in the following sections.

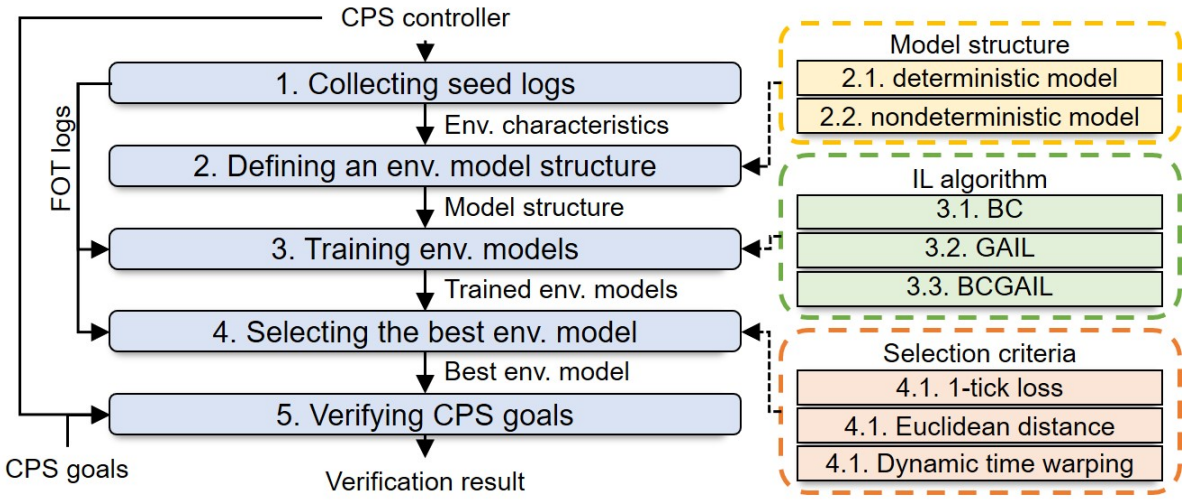


Figure 4.1: ENVI: Overall process and parameters

4.5 Environment Imitation Process

4.5.1 Stage 1: Collecting Seed FOT Logs

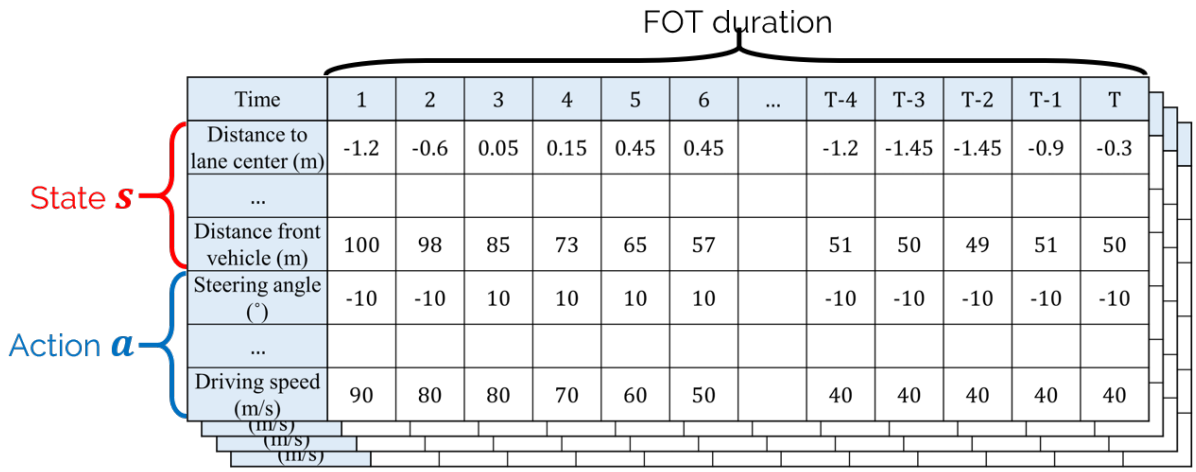


Figure 4.2: CPS FOT log example

The first stage of *ENVI* is to collect the interaction data between the CPS controller and its real environment, which will be used as the "demonstrations" of imitation learning to generate the virtual environment later. For a CPS-ENV interaction model $M_r = (S, A, \pi, \delta_r, s_0)$ defined in Chapter 3, the interaction data collected over time T can be represented as the trajectory of M_r over T steps, i.e., $\langle (s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T) \rangle$ where $s_{t+1} = \delta_r(s_t, a_t)$ and $a_t = \pi(s_t)$ for $t \in \{0, 1, \ldots, T-1\}$. The trajectory can be easily collected from an FOT, since it is common to record the interaction between the CPS controller and its real environment as an FOT log [53]. For example, the lane-keeping system records time-series data of the distances the vehicle deviated from the center of the lane d_t and the steering angles a_t over $t = 0, 1, \ldots, T$ during an FOT. The example of CPS FOT log is visualized in Figure 4.2.

In practice, the trajectory of the same M_r is not necessarily the same due to the uncertainty of the

real environment, such as the non-uniform surface friction. Therefore, it is recommended to collect a few FOT logs for the same M_r. Since the virtual environment model generated by imitation learning will mimic the given trajectories as much as possible, the uncertainty of the real environment recorded in the trajectories will also be imitated. Chapter 5 will investigate to what extent virtual environment models generated by *ENVI* can accurately mimic the real environment in terms of CPS goal verification when the size of the given FOT logs varies.

4.5.2 Stage 2: Defining Environment Model Structures

The second stage of *ENVI* is to generate a virtual environment model from the collected seed logs using an IL algorithm. To generate the environment model, the engineer should first define the environment model structure.

We implement an environment model as a neural network to leverage imitation learning. Before training the environment model, users define the neural network structure.

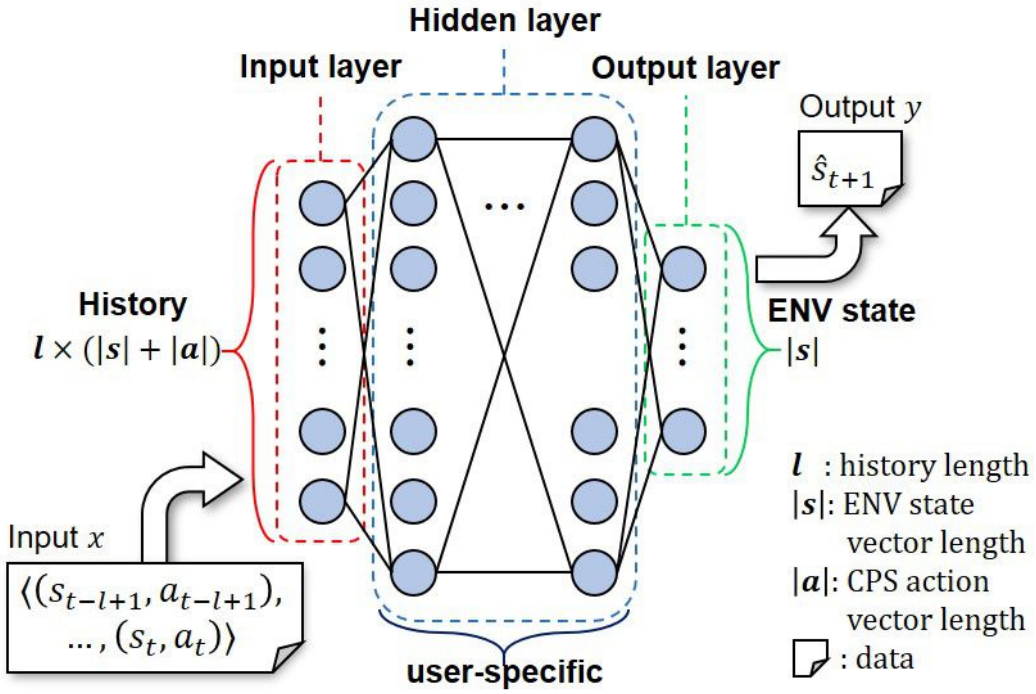


Figure 4.3: Deterministic environment model structure

The virtual environment model structure is based on the environmental state transition function $\delta : S \times A \to S$ defined in Chapter 3. It assumes that the ideal (real) environment generates the next state $s_{t+1} \in S$ by taking the current environment state $s_t \in S$ and the current CPS action $a_t \in A$ only, meaning that (s_t, a_t) is sufficient to determine s_{t+1} in the ideal environment at time t. However, in practice, s may not include sufficient information since it is observed by the sensors of the CPS under verification and the sensors have limited sensing capabilities. To solve this issue, we extend δ for virtual environment models as $\delta_v : (S \times A)^l \to S$ where l is the length of the state-action pairs required to predict the next state. This means that δ_v uses $\langle (s_{t-l+1}, a_{t-l+1}), \ldots, (s_t, a_t) \rangle$ to predict s_{t+1}. Notice that δ_v is equal to δ when $l = 1$. To account for the extension of δ, we also extend the CPS-ENV interaction model $M = (S, A, \pi, \delta, s_0)$ to $M_v = (S, A, \pi, \delta_v, \sigma_0)$ where $\sigma_0 = \langle (s_0, a_0), \ldots, (s_{l-1}, a_{l-1}) \rangle$ is a partial trajectory

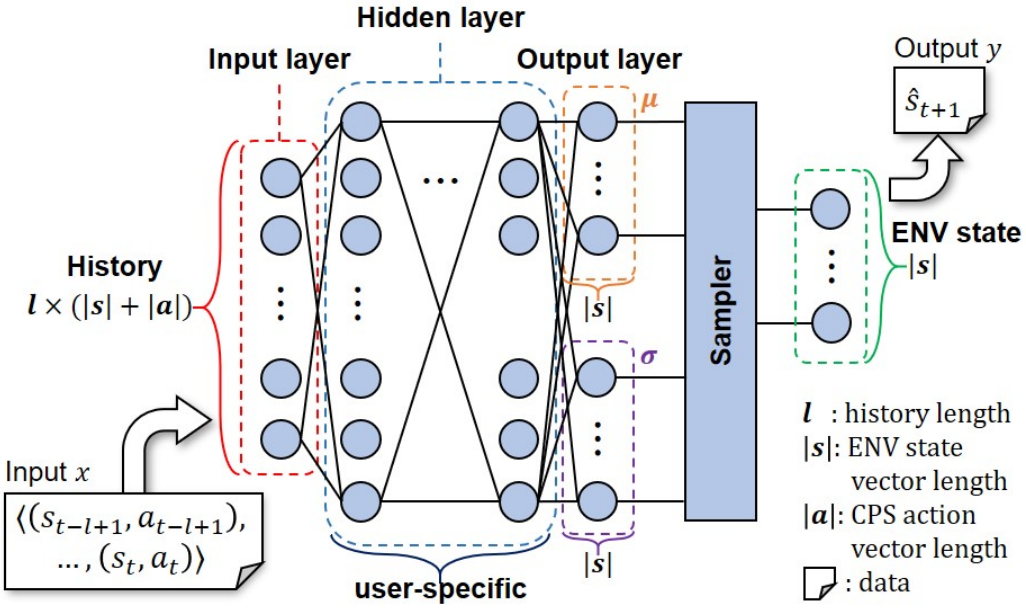


Figure 4.4: Nondeterministic environment model structure

of M_r over l steps starting from s_0. Intuitively speaking, σ_0 is the initial input for δ_v similar to s_0 (and $a_0 = \pi(s_0)$) for δ.

The history length l affects the information captured in environmental states; the larger l, the more information. However, having more information decreases the training and execution time of δ_v. To better balance between the amount of information and the computation cost, one can investigate the seed logs to obtain environment characteristics; for example, if there is a cyclic pattern in the seed logs, l can be the length of the cycle.

The design of hidden layers in δ_v specifies how the output variables of δ_v are calculated from the input variables of δ_v. It is specific to a domain, but general guidelines of the neural network design exist for practitioners [54, 55].

After the l and hidden layers are defined, the determinism of δ_v should be decided by users. The determinism is about the choice between simplicity and realism; a deterministic model, which returns the same output for a given input deterministically, is simpler than a nondeterministic model, which may return different outputs for the same input, whereas the latter is more realistic than the former considering the uncertainty of real environments. Specifically, Figure 4.3 and 4.4 show the structures of deterministic and nondeterministic models, respectively. As defined in Chapter 3, $\delta_v : S^l \times A^l \to S$ takes as input $\langle (s_{t-l+1}, a_{t-l+1}), \ldots, (s_t, a_t) \rangle$ and returns $\hat{s}_{t+1}$ in both structures. However, the neural network in the deterministic model shown in Figure 4.3 is trained to predict $\hat{s}_{t+1}$, whereas the neural network in the nondeterministic model shown in Figure 4.4 is trained to predict a probability distribution (i.e., a mean μ and a standard deviation σ) of $\hat{s}_{t+1}$ assuming that the randomness follows the normal distribution. The output $\hat{s}_{t+1}$ of the nondeterministic model is then calculated by sampling from the distribution using μ and σ. Since most of the uncertainties appearing in CPS log data follow the normal distribution, the experiments in this paper use the normal distribution sampler. However, an engineer can use other random distributions if needed for a specific domain.

Either deterministic and nondeterministic environment model structures are mapped with the collected FOT log data as Figure 4.5. The input neurons are mapped with the history data of the CPS

actions and environment states, and the final output neurons are mapped with the next environment state. The environment model defined here is trained using the seed logs collected in the previous stage. The training process is introduced in the next section.

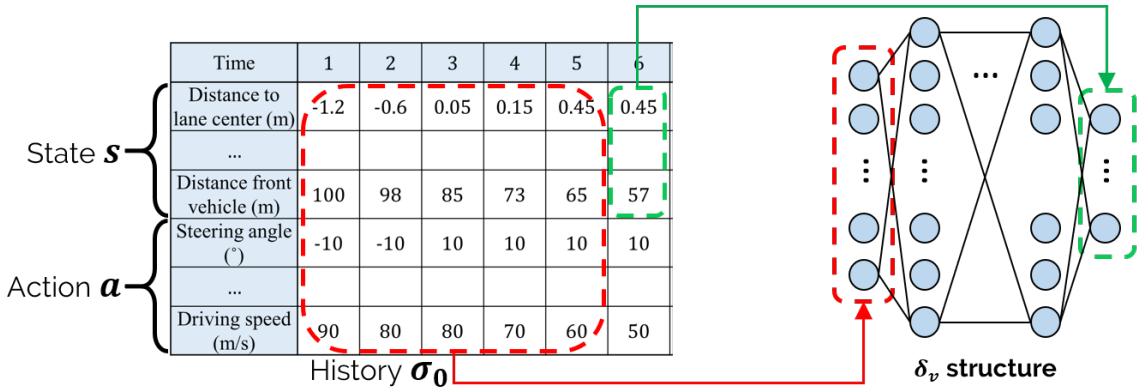


Figure 4.5: Mapping of the FOT log data and the environment model structure

4.5.3 Stage 3: Training Environment Models Using Imitation Learning

Once the structure of δ_v is determined, we can train δ_v using an IL algorithm with the training part of seed logs. The seed logs are used as a proper set of training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, where n is the number of FOT logs for training, X_i is the sequence of δ_v's inputs collected from i-th FOT log and Y_i is the corresponding sequence of outputs (i.e., the expected value of $\delta_v(x_j)$ is y_j for all $j \in \{1, \ldots, |X_i|\}$ and $|X_i| = |Y_i|$ for $i \in \{1, \ldots, n\}$). Since $x \in X$ is an l-length sequence of state-action pairs, we can generate D from an FOT log using a sliding window of length l. Specifically, for an FOT log $\langle (s_0, a_0), \ldots, (s_T, a_T) \rangle$, $x_j = \langle (s_j, a_j), \ldots, (s_{l-j+1}, a_{l-j+1}) \rangle$ for $j \in \{0, \ldots, T - l + 1\}$.

We leverage specific IL algorithms for the environment model generation problem and run the algorithm to train δ_v using D. In the following subsections, we explain how each of the representative IL algorithms, i.e., BC, GAIL, and the combination of BC and GAIL (BCGAIL), can be used for training δ_v.

Note that we only present how BC, GAIL, and BCGAIL can be extended for ENVI as representative examples since they are the most widely used IL algorithms. Nevertheless, all IL algorithms can be extended for ENVI in general, as long as an IL algorithm is modified for training the environmental state transition function $\delta : S \times A \to S$ from training policy function $\pi : S \to A$, as described in the following.

ENVI BC algorithm

As described in Section 4.3.1, BC trains an environment model δ_v using supervised learning. Pairs of the input and output of the real environment recorded in FOT logs are given to δ_v as training data, and δ_v is trained to learn the real environment state transition shown in the training data.

Specifically, the BC algorithm (whose pseudocode is shown in Algorithm 1) takes as input a randomly initialized environment model δ_v and a training dataset D; it returns a set of trained environment models M.

The algorithm initializes a set of trained environment models M (line 1). The algorithm then iteratively trains δ_v using D until a stopping condition (e.g., a fixed number of iterations) is met (lines 2–9). For each $(X, Y) \in D$, the algorithm repeats the following (lines 3–7): (1) executing δ_v on X to predict

Algorithm 1: ENVI BC algorithm

Input : ENV model (randomly initialized) δ_v,
Training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$

Output: Set of trained ENV models M

1 Set of trained ENV models $M \leftarrow \emptyset$
2 **while** *not*(*stoping_condition*) **do**
3 **foreach** $(X, Y) \in D$ **do**
4 Sequence of model outputs $Y' \leftarrow \delta_v(X)$
5 Float $loss_{BC} \leftarrow getLoss(Y, Y')$
6 $\delta_v \leftarrow update(\delta_v, loss_{BC})$
7 **end**
8 $M \leftarrow append(M, \delta_v)$
9 **end**
10 **return** M

a sequence of outputs Y' (line 4), (2) calculating the training loss $loss_{BC}$ based on the difference between Y' and Y (line 5), and (3) updating δ_v to minimize $loss_{BC}$ using optimization algorithms such as well-known Adam [56] (line 6). For every iteration of the training the copy of current δ_v is saved in M (line 8). The algorithm ends by returning the trained δ_vs collected in M (line 10).

Algorithm 1 is intuitive and easy to implement. In addition, the model's loss converges fast because it is a supervised learning approach. However, if the training data does not fully cover the input space or is biased, the model may not accurately imitate the real environment.

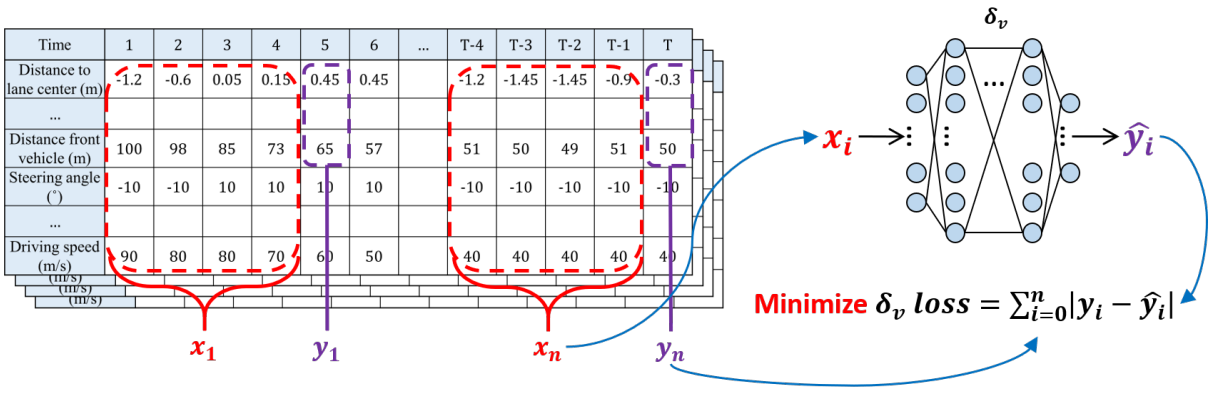


Figure 4.6: ENVI BC algorithm summary

In summary, the ENVI BC algorithm trains the environment model δ_v as shown in Figure 4.6

ENVI GAIL algorithm

As described in Section 4.3.2, GAIL iteratively trains not only δ_v but also the discriminator ζ that evaluates δ_v in terms of the CPS controller π. Specifically, for a state s, ζ evaluates δ_v with respect to δ_r (captured by D) by comparing $\delta_v(s, \pi(s))$ and $\delta_r(s, \pi(s))$. To do this, ζ is trained using D by supervised learning[1], and δ_v is trained using the evaluation results of ζ.

[1]The structure of ζ is similar to δ_v, but the input of ζ is $(s, \delta_v(s, \pi(s))$ and the output of ζ is a reward value r.

Recall that ζ is another neural network shown in Figure 4.7, whose input is a pair of environment model input and output data, and output is a reward to the environment model. ζ quantitatively evaluates how similar the simulated environmental state transition, expressed in the pair of model input and output, is to the real environment state transition. It returns the evaluation result as a reward (e.g., a high reward for the real environment state transition, but a low reward for the simulated or fake environment state transition). Therefore, the number of the input neurons of the discriminator is $(l \times (|s| + |a|)) + |a|$, and the number of the output neurons is 1. The hidden layers of the discriminator also should be defined by users like the environment model.

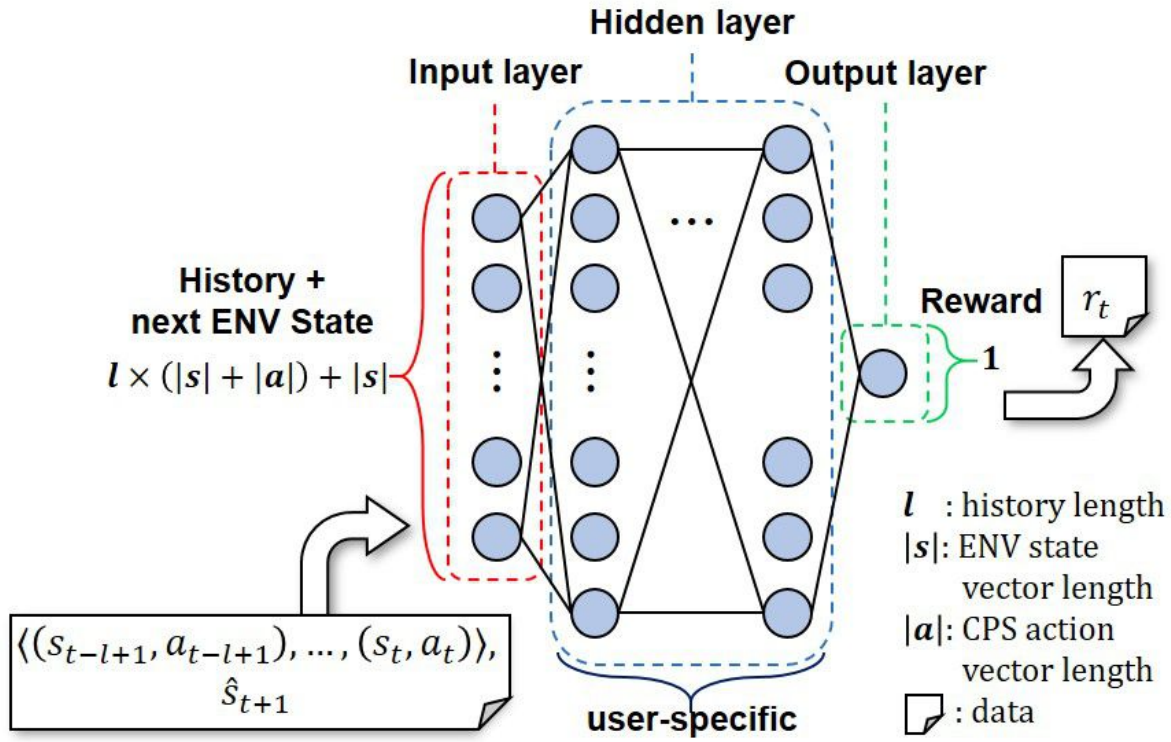


Figure 4.7: The discriminator structure for GAIL

Algorithm 2 shows the pseudocode of GAIL. Similar to Algorithm 1, it takes as input a randomly initialized environment model δ_v and a training dataset $D = (X, Y)$; however, it additionally takes as input a randomly initialized discriminator ζ and the CPS controller under analysis π. It returns a set of trained virtual environment models M.

A set of trained environment models M is first initialized (line 1). The algorithm then iteratively trains both δ_v and ζ using D and π until a stopping condition is met (lines 2–20). To train ζ, for each $(X, Y) \in D$ (lines 3–18), the algorithm executes δ_v on X to predict a sequence of outputs Y' (line 4), calculates the discriminator loss $loss_d$ indicating how well ζ can distinguish Y and Y' for X (line 5), and updates ζ using $loss_d$ (line 6). Once ζ is updated, the algorithm trains δ_v using ζ and π (lines 7–17). Specifically, the algorithm initializes a sequence of rewards R (line 7) and a model input x' (line 8), collects $r \in R$ for each x' using δ_v, π, and ζ (lines 9–15), calculates the environment model loss $loss_{GAIL}$ by aggregating R (line 16), and updates δ_v using $loss_{GAIL}$ using optimization algorithms in reinforcement learning [57, 58] (line 17). To collect $r \in R$ for each x' (lines 9–15), the algorithm executes δ_v on x' to predict an output y' (line 10), executes ζ on x' and y' to get a reward r (line 11),

Algorithm 2: ENVI GAIL algorithm

Input : ENV model (randomly initialized) δ_v,
Discriminator (randomly initialized) ζ,
Function of CPS decision-making logic π,
Training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$

Output: Set of trained ENV models M

Set of trained ENV models $M \leftarrow \emptyset$
while $not(stoping\_condition)$ **do**
  foreach $(X, Y) \in D$ **do**
    `// Discriminator training`
    Sequence of model outputs $Y' \leftarrow \delta_v(X)$
    Float $loss_d \leftarrow getDisLoss(\zeta, X, Y, Y')$
    $\zeta \leftarrow update(\zeta, loss_d)$
    `// Environment model training`
    Sequence of model rewards $R \leftarrow \emptyset$
    Model input $x' \leftarrow X[0]$
    for $|X| - 1$ **do**
      Model output $y' \leftarrow \delta_v(x')$
      Reward $r \leftarrow \zeta(x', y')$
      $R \leftarrow append(R, r)$
      CPS action $a \leftarrow \pi(y')$
      $x' \leftarrow updateInput(x', y', a)$
    end
    Float $loss_{GAIL} \leftarrow aggregate(R)$
    $\delta_v \leftarrow update(\delta_v, loss_{GAIL})$
  end
  $M \leftarrow append(M, \delta_v)$
end
return M

appends r at the end of R (line 12), executes π on y' to decide a CPS action a (line 13), and updates $x' = \langle (s_1, a_1), (s_2, a_2) \ldots, (s_l, a_l) \rangle$ as $x' = \langle (s_2, a_2) \ldots, (s_l, a_l), (y', a) \rangle$ by removing (s_1, a_1) and appending (y', a) (line 14). A copy of δ_v is temporarily saved in M for each iteration (line 19) and the algorithm ends by returning M, the set of trained δ_vs (line 21).

Notice that, to train δ_v, GAIL uses the input-output pair (x', y') simulated by π and ζ, in addition to the real input-output pair (x, y) in D. This is why it is known to work well even with a small amount of training data [50, 52]. However, the algorithm is more complex to implement than BC, and the environment model converges slowly or sometimes fails to converge depending on hyperparameter values.

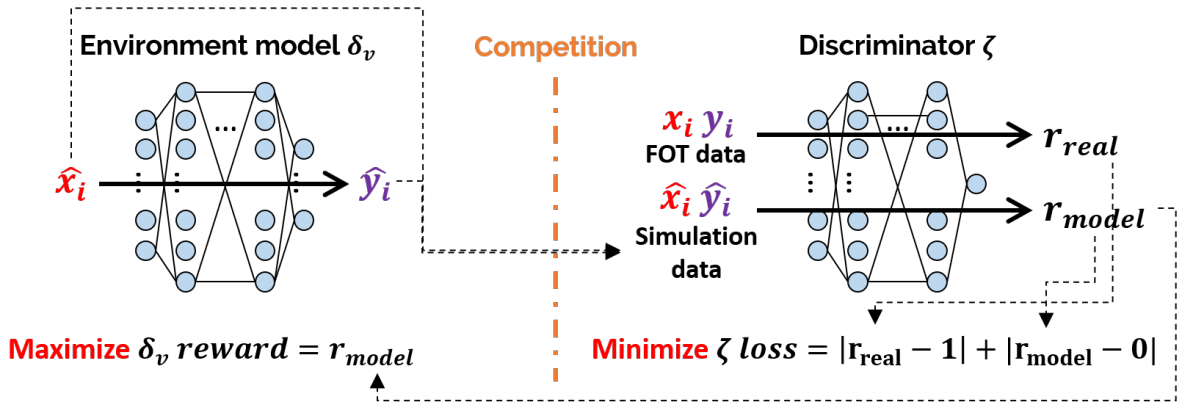


Figure 4.8: ENVI GAIL algorithm summary

In summary, the ENVI GAIL algorithm trains the environment model δ_v as shown in Figure 4.8

ENVI BCGAIL algorithm

Notice that BC trains δ_v using the training data only, but GAIL trains δ_v using the simulated data as well; BC and GAIL can be combined to use both training and simulated data without algorithmic conflict. This idea is suggested by [50] to improve learning performance, and [52] later implemented the idea as an algorithm BCGAIL.

The BCGAIL algorithm is the same as GAIL in terms of its input and output, and it also trains both δ_v and ζ similar to GAIL. In particular, ζ is updated as the same as in GAIL. However, δ_v is updated using both $loss_{BC}$ (line 4 in Algorithm 1) and $loss_{GAIL}$ (line 15 in Algorithm 2). By doing so, BCGAIL can converge fast (similar to BC) with a small amount of training data (similar to GAIL).

Specifically, we can implement the BCGAIL algorithm for *ENVI* by mixing Algorithms 1 and 2. The input of the BCGAIL algorithm is the same with Algorithm 2. The algorithm iteratively trains both δ_v and ζ until a stopping condition is met. For each training data, the algorithm repeats the following: (1) training ζ by Algorithm 2, (2) calculating the BC's environment model loss by Algorithm 1, (3) calculating the GAIL's environment model loss by Algorithm 2, and (4) updating δ_v using the sum of the losses calculated in the step (2) and (3) to minimize both. The algorithm ends by returning δ_v.

In summary, the ENVI BCGAIL algorithm trains the environment model δ_v as shown in Figure 4.9

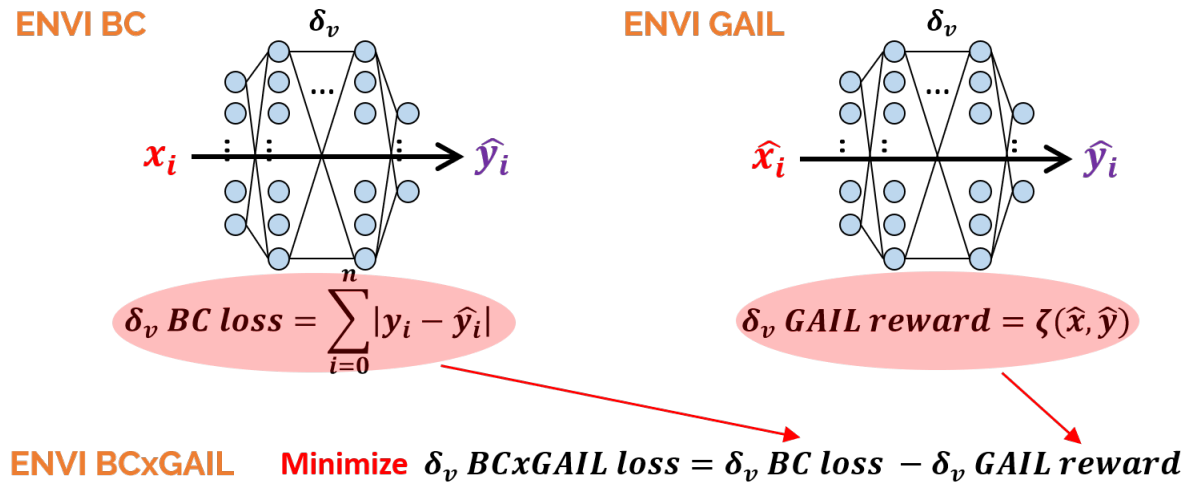


Figure 4.9: ENVI BCGAIL algorithm summary

4.5.4 Stage 4: Selecting the Best Environment Model

The IL algorithms return a set of trained environment models, and the best environment model that mimics the actual environmental state transition well is selected in this stage. This is because many IL algorithms, especially that based on GAIL, suffer from the convergence difficulty problem; the model's loss slowly converges or fails to converge [59]. Thus, we cannot guarantee the latest model to be the best model, and a *validation* process is required to select the best model from candidate models stored during the training procedure. In original IL, human experts usually observe the simulation traces of the trained model to evaluate whether the model behaves like themselves [60]. However, the physical environment is the target of imitation of ENVI, so it is challenging to validate environment models manually. To address this and automatically evaluate trained models, we suggest using three metrics: (1) 1-tick loss, (2) Euclidean distance, and (3) Dynamic Time Warping (DTW). The idea behind the metrics is to assess the similarity between the virtual and real environments using the validation part (i.e., not used for training) of the seed logs. Using the metrics, the best model can be automatically selected from the candidate models generated by the IL algorithm from the previous stage. The following paragraphs detail the three metrics.

1-tick loss (exact matching of the 1-step execution) The first metric evaluates the 1-step execution of δ_v. This expects that if a single environmental state transition mimics the real environment well, the simulation result, which is the sum of accumulated state transitions, will also be realistic [48]. This rationale is the same as that of the BC algorithm. Therefore, the same loss function is also used here. Specifically, all possible model inputs collected from the validation FOT logs are given to δ_v, and δ_v's outputs are compared to the expected outputs collected from the validation dataset to calculate the environment model's validation loss.

Euclidean Distance (exact matching of the T-step executions) The second metric verifies that the model's T-step simulation results exactly match the FOT logs. It expects that given the same starting point, FOT and simulation will proceed the same. Specifically, the model is simulated from the initial states extracted from validation FOT logs, as described in the GAIL algorithm. The simulation logs are compared to the validation FOT logs by the Euclidean distance. Euclidean distance compares ith point

of simulation log to the ith point of FOT log (so-called lock-step alignment) [61], so it captures whether the δ_v's simulations are precisely the same with FOTs well.

Dynamic Time Warping (pattern matching of the T-step executions) The third metric quantifies the similarity of the patterns of T-step simulations and FOTs. This assumes that it is almost impossible for the simulation to be exactly the same as the FOT in the multi-step simulation, so it at least seeks to find an environmental model whose simulation pattern is similar to the FOT. Specifically, it compares the simulation logs and FOT logs by Dynamic Time Warping (DTW). DTW is a time-series distance metric that compares a point in a source series to many points in a target series (so-called elastic alignment) and finally quantifies the similarity of the patterns of two time-series [61]. Therefore, it measures how similar the behavior pattern of the virtual environment is to the real environment.

4.5.5 Stage 5: Verifying CPS Goals

The last stage of ENVI is to verify the CPS controller under analysis using the simulation with the virtual environment model δ_v generated from the previous stages. This is decoupled from the previous stages that leverage IL, so engineers can use any simulation-based methods with δ_v to get the CPS goal verification result $\psi(M_v, \phi)$ for a given goal ϕ. Specifically, an engineer can test the controller π based on δ_v that provides realistic inputs (i.e., observable states) to π by simulating the virtual CPS-ENV interaction model $M_v = (S, A, \pi, \delta_v, \sigma_0)$ to collect many execution trajectories instead of FOTs.

Algorithm 3: *ENVI* CPS controller simulation

Input : ENV model (trained) δ_v,
CPS controller π,
Simulation duration T,
init model input history σ_0

Output: Simulation trajectory τ

Sequence of state and action $\tau \leftarrow \emptyset$
History $\sigma \leftarrow \sigma_0$
for T **do**
  state $s \leftarrow \delta_v(\sigma)$
  action $a \leftarrow \pi(s)$
  $\tau \leftarrow append(\tau, (s, a))$
  $\sigma \leftarrow update(\sigma, (s, a))$
end
return τ

Specifically, CPS simulation using the generated environment model δ_v can be executed following the Algorithm 3. To simulate M_v, the initialization data σ_0 should be given. Since σ_0 is the partial trajectory of M_r over l steps, the engineer should conduct partial FOTs over l steps to get σ_0. Notice that acquiring σ_0 is much cheaper than having full FOTs for FOT-based CPS goal verification since l is much shorter than T (i.e., the full FOT duration).

From the given initial input σ_0, the CPS controller π and the environment model δ_v are sequentially executed for simulation duration T steps. First, δ_v produces an environmental state. π observes the environmental state and decides an action. The new state and action is recorded. Based on the new

state and action, the environment model input σ is updated by moving the history window one step forward. After T-steps iteration, the collected simulation log τ is returned.

This simulation can be repeated to accumulate as many assessments of ϕ as needed by statistical verification methods such as statistical model checking (SMC) [62]. Indeed, an SMC algorithm (e.g., Sequential Probability Ratio Test [62]) may require thousands of trajectories to verify the CPS goal depending on the given confidence interval. Therefore, the simulation using generated δ_v allows engineers to perform the CPS goal verification with little cost in such cases. Although the initial input σ_0 is required for simulating M_v, having σ_0 is much cheaper than having full FOT logs for FOT-based CPS goal verification since l (i.e., the length of σ_0) is much shorter than T (i.e., the entire FOT duration). Furthermore, only one σ_0 would be enough for a nondeterministic δ_v since it returns different simulation results for the same σ_0.

4.6 Summary

In this chapter, we introduced our novel data-driven environment model generation approach using imitation learning, ENVironment Imitation (ENVI). Specifically, we introduced the process of ENVI and user-configurable parameters (e.g., model structures, IL algorithms, and model selection criteria).

Chapter 5. Empirical Evaluation

5.1 Introduction

This chapter empirically evaluates our novel approach ENVI based on case studies of autonomous driving systems and their goal verification. Specifically, it evaluates the accuracy and efficiency of *ENVI-based verification* applied to the real CPS software controllers. We will call the simulation-based CPS goal verification using ENVI-generated models *ENVI-based verification* in this chapter.

This chapter is organized as follows: Chapter 5.2 presents the research questions of the empirical evaluation. Chapter 5.3 shows the CPS experimental environment and its implementation manuals. Chapter 5.4 shows the experimental data collection process of this evaluation. Chapter 5.5 describes detailed experimental setup for environment model generation. Chapter 5.6 provides the evaluation results for each research question. Chapter 5.7 reveals the threats of the evaluation. Chapter 5.8 summarizes this chapter.

5.2 Research Questions

This section defines research questions of our empirical evaluation. We first investigate the impact of using different ENVI parameters (i.e., model determinism, IL algorithms, and model selection criteria) on CPS goal verification and obtain a guide for setting optimal ENVI. We then analyze how similar the environment models generated by the optimized ENVI are to the real environment and how accurate the goal verification results of the seen controllers (i.e., controllers used for seed log collection) are. We then analyze ENVI's environment model generation efficiency for efficient CPS goal verification in terms of the cost of collecting FOT logs for the model generation. In addition, we analyze the goal verification accuracy of the unseen controllers. We finally empirically search seed log collection strategies to make ENVI effective for unseen controller variation verification. To summarize, we answer the following five research questions:

RQ1: What is the impact of ENVI parameters on the simulation-based CPS goal verification accuracy?

RQ2: How accurate is the simulation-based goal verification of seen controller using ENVI?

RQ3: Can ENVI efficiently generate environment models with a small amount of FOTs?

RQ4: How accurate is the simulation-based goal verification of unseen controller using ENVI?

RQ5: What are the effective seed log collection strategies for accurate unseen controller verification using ENVI?

5.3 Experiment Environment: Platooning LEGOs

In software engineering, it is challenging to validate an approach in the real CPS because development of a CPS experimental environment requires huge cost. Therefore, to conduct our case study on the real CPS and also to provide a public CPS experimental environment, we designed and developed an open

physical CPS experimental environment named *Platooning LEGOs*. This experimental environment is public experimental environment, so anyone can implement this cheaply and easily and utilize it for their research. In this section, we first introduce our experimental environment in detail.

5.3.1 Introduction to Open CPS Experiment Environment

Cyber-physical systems (CPS), such as autonomous vehicles, play an increasingly important role in modern society, attracting considerable interest in CPS engineering [63]. Because not only virtual information but also physical conditions or people must be considered, it is difficult to fully anticipate uncertainties in the environment of a CPS at the time of designing. Therefore, a CPS essentially requires adaptation functionality that can consistently achieve system goals in uncertain environments.

Another characteristic of some modern systems is that they form systems-of-systems (SoS) in which multiple independent systems cooperate to achieve higher-level goals that cannot be achieved by a single system [16]. Examples of SoS are clusters of vehicles or drones, smart factories where many robotic systems work together, and complex defense systems with multiple weapon systems. As the size and influence of SoS increase, an important objective of SoS engineering is to ensure that SoS goals are achieved stably regardless of uncertainty.

In this context, cyber-physical systems-of-systems (CPSoS) require engineering for collaborative adaptations in the uncertain physical world [64]. To promote active research and share common adaptation problems, the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) research community has accumulated several exemplars[1] [65, 66, 67, 68, 69, 70, 71, 72, 73, 74]. However, there are few exemplars for adaptation engineering of CPSoS. Moreover, while most exemplars have provided simulators, studying CPS only in simulations without physical environments has limitations in reflecting reality. Furthermore, building a physical experimental environment often requires specialized domain knowledge and entails high costs.

To meet the need for a CPSoS exemplar to consider the physical environment realistically for adaptation engineering, we propose an open physical exemplar called *Platooning LEGOs*. As a representative example of CPSoS, we selected a platooning technology for autonomous vehicles [75]. Platooning is an industrial technology that is actively being developed by vehicle manufacturers. Vehicles with the same destination form a platoon through communication, drive in a line to reduce air resistance, thus reducing fuel consumption, and adjust the distances between them to reduce road occupancy. Platooning is self-adaptive to uncertain situations in a driving environment. Our exemplar implements platooning using programmable LEGO robots. Unlike the cases where platooning robots have been implemented and used in experiments privately [76, 77, 78], we propose *Platooning LEGOs* as a reproducible and expandable exemplar that allows anyone to build the same physical experimental environment for CPSoS engineering. In summary, our *Platooning LEGOs* exemplar contributes to the field as:

- a CPSoS exemplar: an industrial adaptation model problem (platooning) representing both CPS and SoS,
- a physical exemplar: a physical experimental environment producing real data from physical sensors and actuators,
- an open exemplar: an exemplar that allows anyone to build the same physical experimental environment with a limited budget using LEGOs and expand its physical and software elements.

[1] SEAMS exemplar repository:
https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

5.3.2 Platooning LEGOs Overview

SoS-level overview

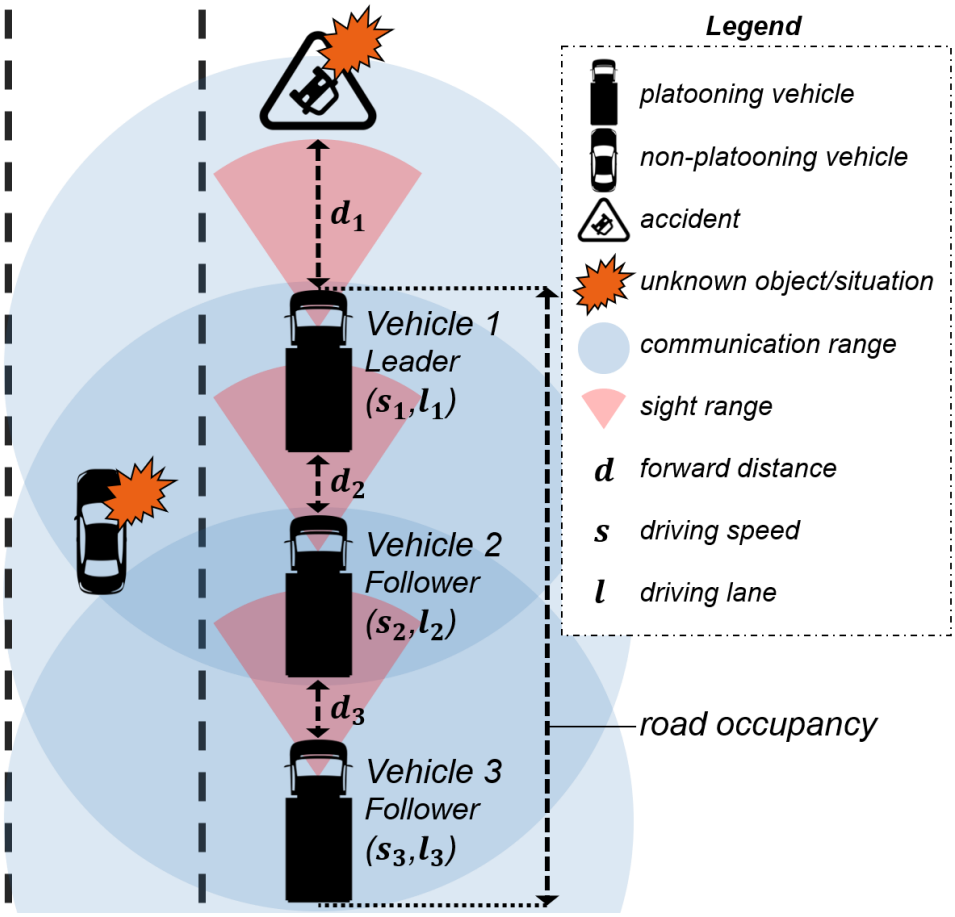


Figure 5.1: Overview of *Platooning LEGOs*

Platooning is a technology currently being developed by real industries; therefore, levels and functions vary according to the manufacturer. We refered to demonstrations of various platooning techniques and simplified core features that can be implemented using LEGOs. Figure 5.1 shows an overview of our *Platooning LEGOs*. Each platooning vehicle is a programmable LEGO robot. A vehicle can independently drive in a lane, control its driving speed, change lanes, and detect obstacles ahead. Each vehicle transmits a set of raw data that include its current driving speed, lane, and forward distance. The platoon comprises three vehicles. The first vehicle is the leader, which knows the conditions ahead. The second vehicle is a follower driving along the path of the leader and relaying the leader's state to a third vehicle, another follower, which follows the second vehicle and is outside the leader's communication range. As unknown objects or situations exist in a road environment, each autonomous vehicle makes decisions on its speed and lane so that the platoon adaptively achieves its SoS-level goals in an uncertain environment.

The SoS-level adaptation goals of the platoon are summarized in Table 5.1. There are two "hard" goals (with clear satisfaction criteria) and two "soft" goals (without clear-cut criteria). If the two hard goals are achieved, the platoon tries to achieve the soft goals. The first hard goal is to prevent a collision with another vehicle or other object. The second hard goal is to drive in a row in the same lane to minimize air resistance and thus fuel consumption. The first soft goal is to minimize road occupancy

Table 5.1: Robot vehicle goals of the *Platooning LEGOs*

Goal	**Description**	**Evaluation metric**
(Hard) Collision prevention	All vehicles of the platoon shall not collide with any vehicle or object.	$(d_1 > 0) \wedge (d_2 > 0) \wedge (d_3 > 0)$
(Hard) Driving in a row	Except when changing lanes, all vehicles of the platoon must drive in a row on the same lane.	$(l_1 = l_2) \wedge (l_2 = l_3) \wedge (l_3 = l_1)$
(Soft) Road occupancy minimization	In order to reduce road occupancy, the distance between vehicles in the platoon should be minimized as much as possible.	$d_2 + d_3 + (3 * vehicleLength)$
(Soft) Travel speed maximization	In order to shorten the travel time, the average driving speed of the platooning vehicles should be maximized within road limits.	$\frac{s_1 + s_2 + s_3}{3}$

for smooth traffic flow. This is achieved by minimizing the distances between the platooning vehicles by speed adjustments. The second soft goal is to maximize the platoon's driving speed to minimize travel time. The data-based logical or numeric evaluation metrics of the goals are displayed in Table 5.1. In cases in which the experimental data cannot accurately show whether the goals are achieved in a physical environment (e.g., a side collision of a vehicle), users can observe the experiment itself in addition to the data to determine whether the goals are achieved.

Self-adaptive constituent vehicles

In this section, we describe the behavior of each constituent vehicle. Each vehicle is autonomous and stand-alone. The vehicles' activities are summarized in Table 5.2. The activities are organized as a MAPE (Monitor, Analyze, Plan, and Execute) loop of SAS [79]. Although in this work we implemented an independent MAPE loop in each vehicle without a coordinator, different types of MAPE patterns for SoS can be used in our exemplar [80]. Our LEGO vehicles are equipped with three kinds of sensors: a color sensor for line following, an ultrasonic sensor for obstacle detection, and a maximum of two buttons for reception of the driver's manual commands. We assume the color sensor for line following to be a basic function for autonomous driving, and it is thus not described in Table 5.2. Only driving lane and speed adaptations are described.

The first vehicle (leader) senses forward distances to detect accidents or other obstacles. It can also sense the driver's commands, such as manual control of speed or lane, through buttons. In some real examples of platooning, a leader vehicle allows manual driving. However, even if there is no manual command, it can automatically adapt the driving speed and lane. Moreover, it sends its current driving speed, lane, and forward distance to a follower vehicle.

The second vehicle (follower) senses forward distances and receives the leader's current state. However, unlike the leader, when it is part of a platoon, its steering wheel and accelerator cannot be used, and only adaptive cruise control is allowed. In real cases, a follower can leave or join the platoon at the driver's command, but the exemplar described herein only covers the joined state of the platooning protocol. If a user wishes to allow manual command of a follower vehicle, it can be expanded using buttons. Based on the monitored and received situation of the platoon, the second vehicle follows the

Table 5.2: Activities of vehicles in Platooning LEGOs

Vehicle	**Activity**
Vehicle 1: Leader	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Driver's manual command (Button)
	Analyze & Plan: • Driving speed decision • Driving lane decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • *msg1* (to vehicle 2) – Current driving speed – Current driving lane – Current forward distance
Vehicle 2: Follower	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Peers' situation (*msg1*) (Bluetooth)
	Analyze & Plan: • Driving speed decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • *msg2* (to vehicle 3) – Current driving speed – Current driving lane – Current forward distance
Vehicle 3: Follower	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Peers' situation (*msg2*) (Bluetooth)
	Analyze & Plan: • Driving speed decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • None

leader's lane and decides the speed.

The third vehicle (follower) follows the first and second vehicles. Because the leader's communication range may be limited, the third vehicle only receives messages from the second vehicle. This communication topology is called "predecessor following" [81]. Although all activities of the third vehicle are subsumed by the second one, it is still an independent constituent system, and it adapts its speed to contribute to the platoon's goals.

Environmental uncertainties of the platoon

The environmental uncertainties that can be addressed by *Platooning LEGOs* are summarized in Table 5.3. The platoon has limited knowledge of the peers' situation, the physical road environment, such as other vehicles or accidents, and human drivers' behavior. It is almost impossible to enumerate all possible situations of the platoon at the time of designing. Therefore, the platoon should be adaptive to the uncertainty of the environment. Because *Platooning LEGOs* provide a physical experimental environment, realistic physical events can be simulated in experiments. Moreover, the platoon interacts with the environment through sensors and actuators. Actual interactions may differ from the expected interactions due to sensing/actuating noise or failure. Such incomplete interactions can also produce uncertain platoon operation results. To simulate diverse settings and address various uncertainties, sensor/actuator noise or failure rates can be introduced to experiments.

5.3.3 Implementation Manuals

Physical implementation

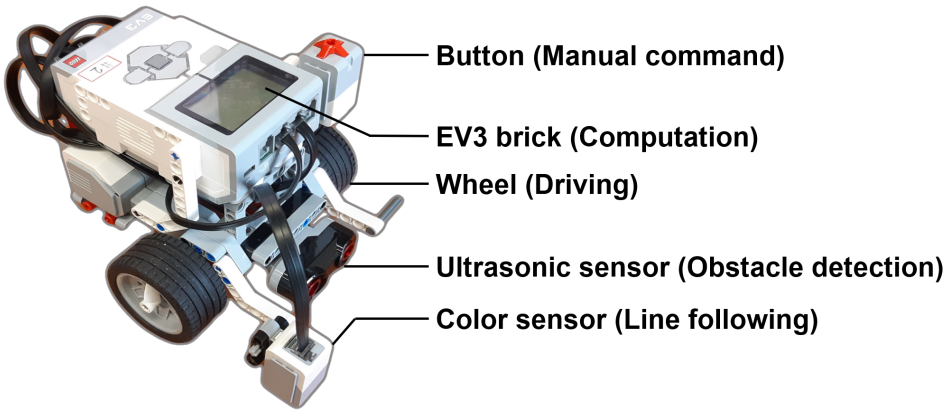


Figure 5.2: A LEGO Mindstorms EV3 vehicle

Vehicle implementation: Each vehicle is an independent LEGO MINDSTORMS EV3 robot (The LEGO Group, Denmark), as shown in Figure 5.2. Each robot is equipped with two main wheels connected to motors and one auxiliary wheel. It is also equipped with a color sensor and an ultrasonic sensor to sense the road and the situation in front of it. Messages from other vehicles are received via Bluetooth, which is embedded in the EV3. A button can also be attached for commands from a human driver, such as manual lane change or acceleration. All physical implementations are very simple and follow the building instructions provided by the manufacturer. The instructions and manuals of the *Platooning*

Table 5.3: Environmental uncertainties addressed by the *Platooning LEGOs*

Environmental uncertainty	**Subcategory**	**Description**
Uncertainty due to the limited knowledge of the environment	Unpredictable peers' situations	Each vehicle determines its behavior based on the peers' situation received through a network connection, but the possible peers' situations cannot be fully anticipated in advance.
	Unpredictable physical environment	Unexpected physical conditions, such as an interruption by another vehicle or an accident on the road, may occur, which cannot be predicted.
	Unpredictable human behavior	A human driver can give a direct command to a vehicle through buttons, and the behavior may not be accurately predictable and may interfere with the achievement of the platoon's goals.
Uncertainty due to the incomplete interaction with the environment	Inaccurate sensing	The sensed or received information may not accurately reflect the actual environmental situation because of sensor noise or message transmission/reception delays.
	Sensing failure	The vehicles may fail to obtain information on the environment because of the broken physical sensors or message loss.
	Inaccurate effecting	The planned adaptation behavior may not be ideally applied to the physical world because of the motor's limited precision or physical constraints such as frictional force of the road.
	Effecting failure	With a very low probability, the planned action may not be executed due to a connection error with the motor or other unexpected causes.

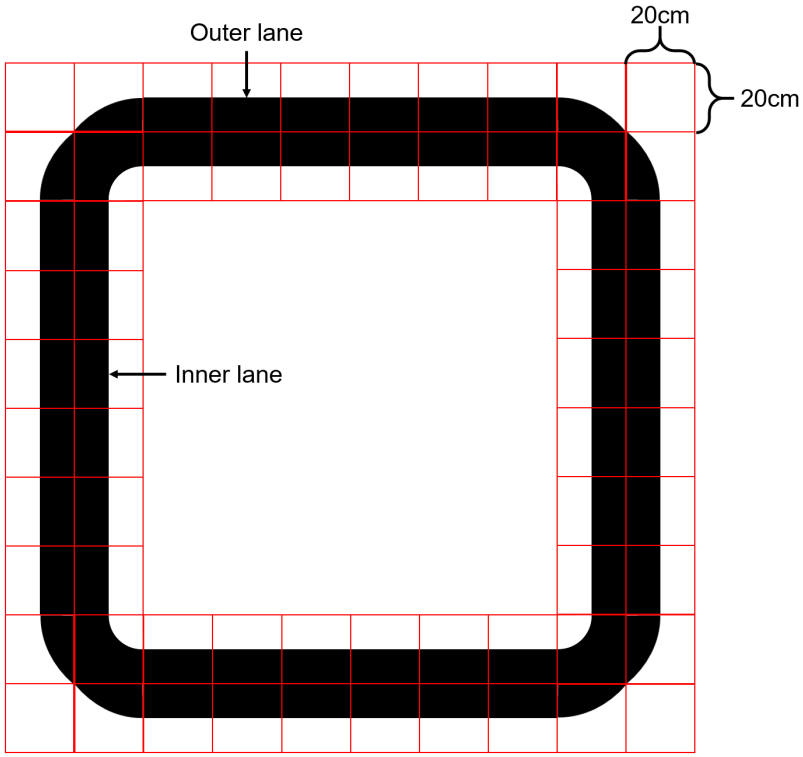


Figure 5.3: Road environment

LEGOs have been uploaded to our GitHub repository[2].

Road implementation: To simulate platooning on an endless highway, engineers print circular roads, as shown in Figure 5.3, and attach them to a floor. There are an outer and an inner lane. Vehicles drive clockwise following the boundaries of the black line. When a vehicle decides to change lanes, it turns in the direction of the new lane and drives until it finds a white area. The road implementation material has also been uploaded to our repository[2]. The simulated road environment can be easily replicated using sheets of paper and a space of only about 2m×2m. Figure 5.4 shows the vehicles and simulated road environment. The three vehicles are in the outer lane.

As *Platooning LEGOs* provides a physical experimental environment with very simple physical implementations, they allow software engineers to focus on the vehicles' software. The software implementation guide is provided in the next subsection.on.

Software implementation

The vehicles' software is implemented using a Python API[3]. Each vehicle iteratively performs monitoring, analysis, planning, and execution. A detailed description of each step is presented in the code skeleton shown in Algorithm 4.

Monitor & Receive: The lane and speed adaptations are based on recognition of the road environment and the peer vehicles' state. The road environment is monitored by sensors. A color sensor

[2] *Platooning LEGOs* repository: https://doi.org/10.5281/zenodo.4604167 (https://github.com/yongjunshin/Platooning-LEGOs)

[3] Mindstorms EV3 API: https://pybricks.github.io/ev3-micropython

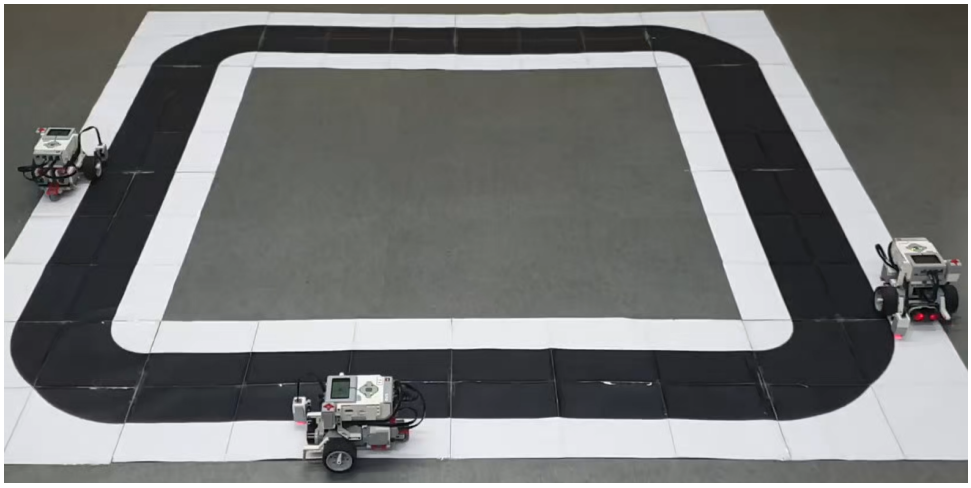


Figure 5.4: A physical implementation of *Platooning LEGOs*

Algorithm 4: A vehicle code skeleton

```
1:  Configuration of sensors, motors, and vehicle
2:  data = DataLog('time','lane','speed')
3:  watch = StopWatch()
4:  while True do
5:      time = watch.time()
6:      color = colorSensor.reflection()
7:      dist = ultrasonicSensor.distance()
8:      peerLane = laneMailBoxFrontVehicle.read()
9:      Adaptation of lane and speed
10:     vehicle.drive(speed, turnRateFor(lane))
11:     laneMailBoxBackVehicle.send(lane)
12:     data.log(time, color, dist, peerLane, lane, speed)
13: end while
```

monitors the floor and returns a color value of the road (Algorithm 4, line 6). An ultrasonic sensor measures the distance to objects in front of the vehicle in millimeters (Algorithm 4, line 7). The peer vehicles' state is received via Bluetooth. A vehicle shares a mailbox with another vehicle and can receive data using the *read*() function (Algorithm 4, line 8). For more complex scenarios, additional sensors, such as touch, gyroscope, and infrared sensors, and mailboxes for information reception from more peers can be used following the API[3] and our manual and sample codes[2].

Analyze & Plan: Each vehicle adapts its driving lane and speed by analyzing the monitored environment, the peers' state, and its own state (Algorithm 4, line 9). Engineers can use their own adaptation approaches and analyze their effectiveness. To guide their implementation, we provide a code skeleton and the sample code used in our experiment through our repository[2].

Execute: A *vehicle* is an instance of a *DriveBase* object in the API, and the adaptation decision on speed and lane is executed by the *drive*() function of the instance (Algorithm 4, line 10). The function receives the speed (mm/s) and rotation angle (deg/s) as inputs. The driving lane is a specific concept in

our exemplar, so a lane change decision must be converted to a rotation angle. Changing the lane can be realized by turning clockwise or counterclockwise. A reusable code for following and changing lanes can be found in our repository[2].

Send: To achieve SoS goals, the state and adaptation decisions of a vehicle should be known to a follower. The vehicle uses the *send*() function of a mailbox shared with the follower so that the follower can read the data. This communication allows the vehicles to be integrated into an SoS.

Data logging: Logging data is an important feature of an experimental environment. The LEGO API[3] provides a simple logging function implemented in two lines of code (Algorithm 4, lines 2 and 12). The data are saved as a CSV file. A timestamp for each iteration of the adaptation loop can also be extracted (Algorithm 4, lines 3 and 5).

5.3.4 Sample Experiment of Platooning LEGOs

Sample scenario

To demonstrate the feasibility of the *Platooning LEGOs* as a physical experimental environment for CPSoS engineering, we conducted a sample experiment. The platooning vehicles were programmed to achieve the adaptation goals described in Table 5.1. The implementation code can be found in our repository[2]. To check whether our platoon implementation is sufficiently adaptive to environmental uncertainties, we introduced two events that could interfere with the goal achievement while the platoon is driving. The first event was an interruption by a *moving obstacle*, such as a non-platooning vehicle on a highway. The second event was a blockage of a lane due to a *fixed obstacle*, such as a traffic accident.

Sample experiment result

The experimental code and result data have been uploaded to our repository[2]. The experimental results are visualized in Figure 5.5. A video of the experiment has also been released[4]. Figure 5.5 (a–d) shows the achievement of the platoon's adaptation goals. Figure 5.5 (e–f) shows the adaptations of each vehicle. The unexpected events (obstacles) are also shown. The two hard goals (collision prevention and driving in a row) were achieved in all scenarios. On the other hand, the achievement of the two soft goals varied depending on the situation.

When moving obstacles interfered with the platoon's driving (moving obstacles 1 and 2 in Figure 5.5), road occupancy increased and travel speed decreased. However, after the moving obstacles disappeared, the vehicles adapted their driving speeds to reduce road occupancy and increase the platoon's travel speed. When stationary obstacles blocked a lane (fixed obstacles 1 and 2 in Figure 5.5), the travel speed decreased. The leader decided to change lanes, and the followers also changed lanes to bypass the obstacles. The vehicles then adapted their speed to maximize the platoon's travel speed. The experiment confirmed that the *Platooning LEGOs* can be used as a case of industrial self-adaptive CPSoS and a physical experimental environment for CPSoS engineering.

[4]Experiment demonstration video - https://youtu.be/tRSoTPq5EEI

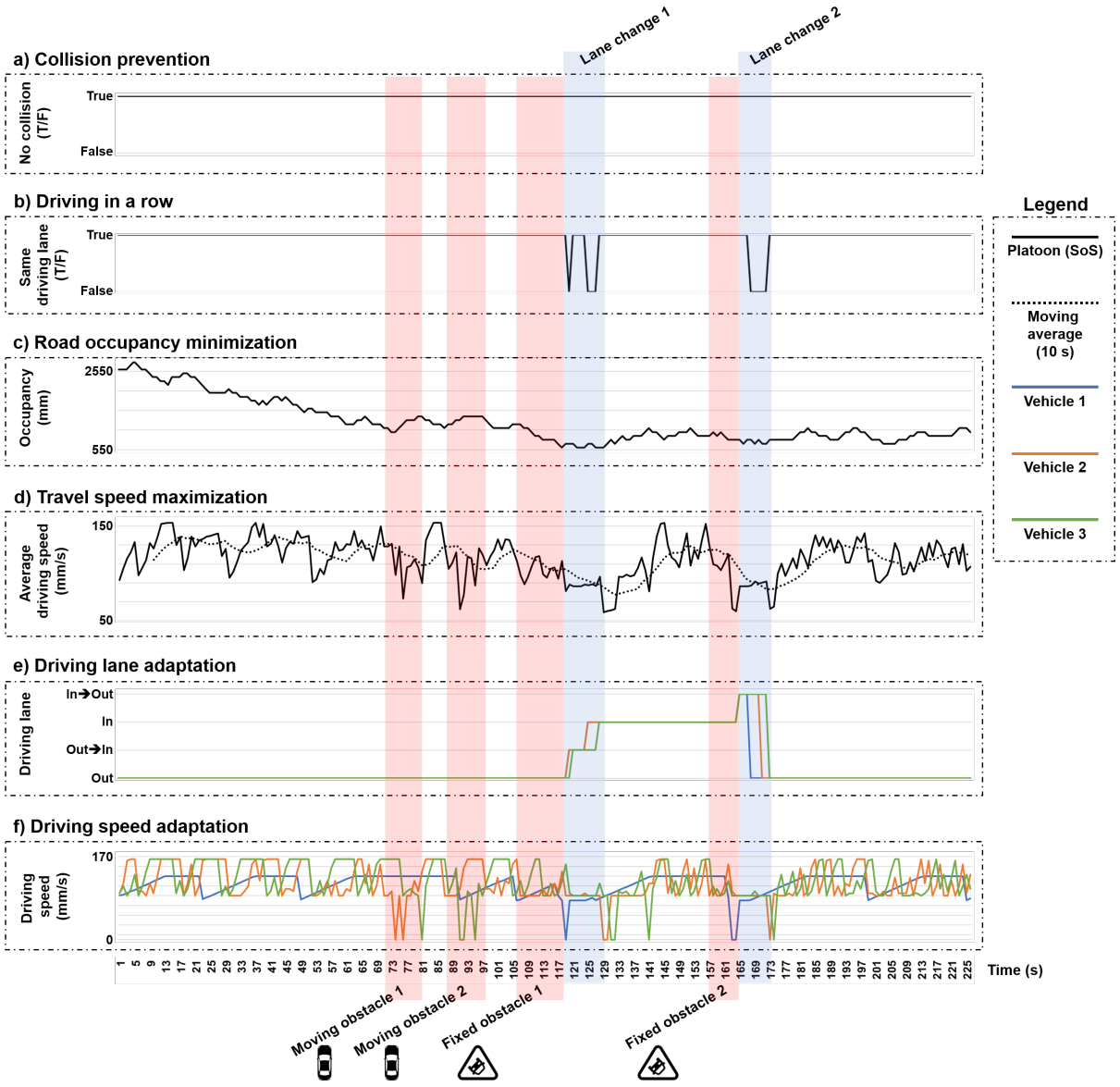


Figure 5.5: Sample experiment results

5.4 Experimental Data Collection

We collected experiment data for this case study using the Platooning LEGOs. We collected the data not for only our case study also for other related research. Therefore, we systematically collect the FOT log data and open the dataset as a research data benchmark. This section describes both how the experiment dataset is collected and how to use the dataset.

5.4.1 Introduction to Open CPS FOT Dataset

Cyber-physical systems (CPSs) continuously adapt their actions to satisfy goals in physical environments [4]. A CPS has a feedback loop consisting of a controller that checks the goal achievement and manipulates physical components based on its decision-making strategy [3].

Developing a decision-making strategy is one of the primary purposes of CPS development. When

there are many goals that a CPS is required to achieve simultaneously, it becomes more challenging to develop an effective strategy. One popular approach is to create dedicated controllers for each goal to divide the concern [82, 83, 84]. It views complex CPSs through the lens of system-of-systems (SoS) [21, 20, 85]. For example, both a lane-keeping system and an adaptive cruise control system operate together within an autonomous vehicle.

Engineers can conduct field operational tests (FOTs) [86] of a CPS under development to evaluate to what extent the CPS can achieve the given goals in the actual operational environment and optimize the configurations of the CPS controllers. However, conducting the CPS FOT has several engineering challenges. FOT results are stochastic because of uncertainty in the physical environment (e.g., sensor noise). It requires engineers to repeat many FOTs to obtain statistically significant results. In a multi-controller CPS, one controller may affect the performance of another controller during an FOT. A specific combination of controllers may trigger an emergent behavior that developers may not expect. Additionally, in many cases, the configuration space of the controllers under analysis is extensive and continuous, making the optimization of the controllers more exhaustive.

To realize these challenges, we have hands-on experience in developing a multi-controller CPS and conducting FOTs. We designed and modeled an autonomous robot vehicle consisting of a lane-keeping system and adaptive cruise control system. We then performed FOTs of 125 possible controller configurations each 50 times, and analyzed the results. This paper provides all materials and datasets related to this case study for future research and shares the lessons learned from our hands-on experience.

In summary, this paper contributes to the research on multi-controller CPS development by providing the following:

- A re-implementable case study of a multi-controller CPS, including its model, software, and hardware implementation manuals,
- An autonomous driving FOT log dataset of 125 controller configurations, each with 50 test results, obtained from about 100 hours of driving,
- Lessons learned from hands-on experience exposing research challenges emerging in the multi-controller CPS FOT,
- Possible applications of the FOT log dataset for future research.

5.4.2 Background of CPS Controller Feedback Loop Design

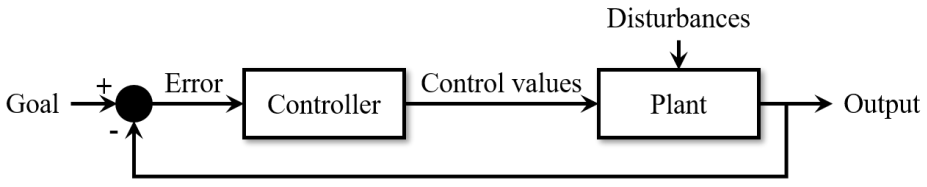


Figure 5.6: A feedback loop from the control perspective [3]

Many CPSs have feedback loops that observe the uncertain and changing environments and make adaptive actions [87]. A popular approach to modeling the feedback loop is based on control theory [88, 83, 89]. Figure 5.6 shows the feedback loop from a control perspective [15, 3]. The feedback loop consists of a controller and a plant. Control values generated by the controller manipulate the plant,

and the plant's behavior depends on the control values. The behavior of the plant is measurable for the system goals. Using the measured behavior of the plant, the controller calculates the error associated with each goal and determines the control values of the plant to minimize the error. In addition to the control values, factors that affect the plant's behavior but are not under the direct control of the system are called disturbances or, sometimes, uncertainties in software engineering. The disturbances can make the behavior of the plant different from what the controller intended, so the controller should mitigate the effect of the disturbances. Many studies expect that the deep foundation of control theory will boost feedback loop design [90, 91, 92, 93]. Therefore, this study also models and develops a multi-controller CPS using feedback loops based on this control perspective.

5.4.3 CPS FOT Data Collection Scenario

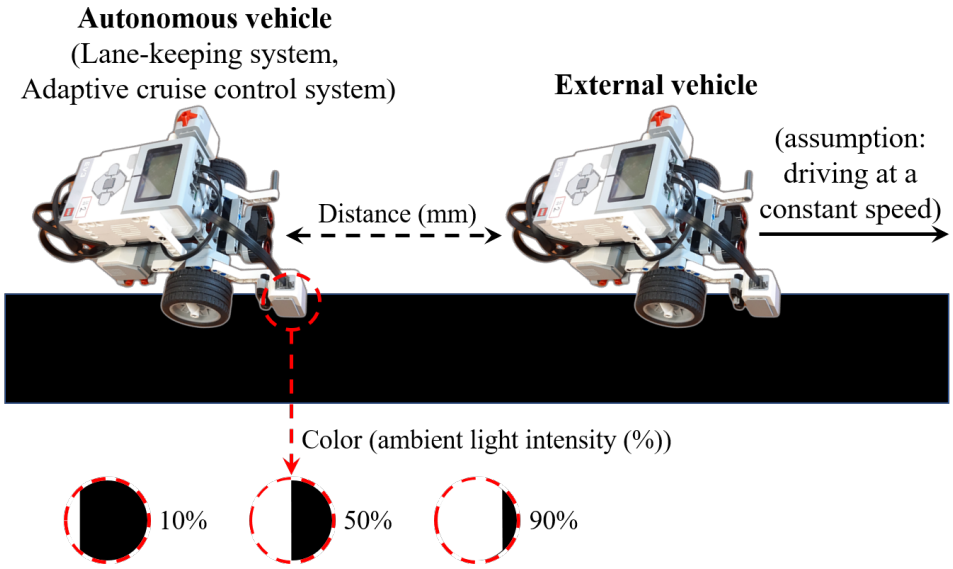


Figure 5.7: An autonomous robot vehicle case study design

This section introduces the design of our case study to develop and analyze a multi-controller CPS. We developed an autonomous vehicle to provide a representative example of a multi-controller CPS. We utilized an open physical experiment environment *Platooning LEGOs* [23] that provides a programmable LEGO robot vehicle and an experimental track design[5]. Leveraging the physical experiment environment, we implemented our case study in Figure 5.7. We developed an autonomous vehicle equipped with a lane-keeping system and an adaptive cruise control system. The vehicle observes its operational environment using a color sensor facing down (i.e., lane) and a distance sensor facing the front. The color sensor gives the light intensity value of the lane under the sensor. The value provides information about the vehicle's relative position from the lane center (i.e., the border between the white and black areas). In addition, there is an external vehicle in front of the autonomous vehicle, so the distance sensor gives the distance between the two vehicles. We assume that the external vehicle drives at a constant speed in this case study.

The autonomous vehicle has two explicit control systems and goals, as shown in Figure 5.8. The control systems are modeled as decoupled feedback loops from the control perspective, and they operate

[5]Hardware implementation manuals of the robot vehicle and the FOT environment: https://github.com/KAIST-SE-Lab/Platooning-LEGOs

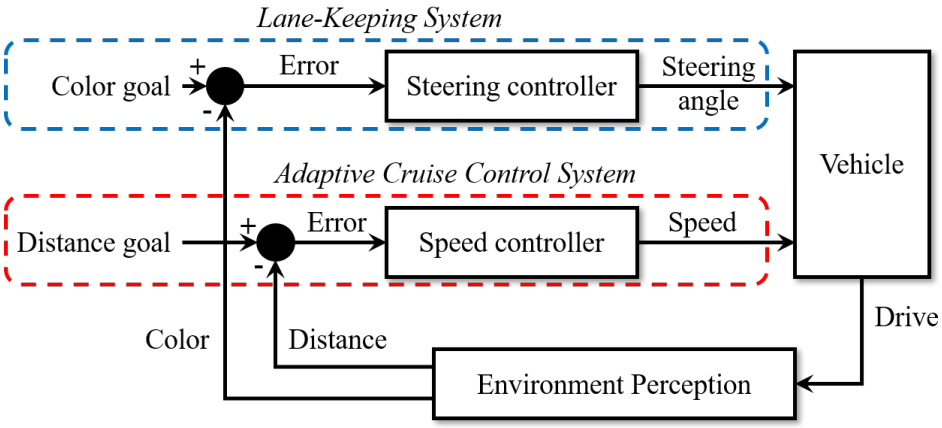


Figure 5.8: Autonomous vehicle controllers

together to achieve the two goals simultaneously. The first goal is to drive as smooth as possible following the center of the lane. The value of the lane center recognized by the color sensor accurately specifies this goal. The lane-keeping system observes the lane color of the current position. It calculates the error between the observed color value and the goal, and a steering controller decides the steering angle to keep the vehicle at the lane center. The second goal is to maintain the distance between the autonomous and the external vehicles to a set distance configured by the user. The adaptive cruise control system observes the distance and calculates the error from the goal. The speed controller sets the speed to minimize the error. Therefore, the steering angle and speed pair specify the vehicle's instant driving state.

We implemented the controllers as PID controllers [94]. A PID controller gets an observation value $o(t)$ (e.g., color or distance) from a sensor and calculates the error $e(t)$ for a goal. It returns a control value $y(t)$ (e.g., steering angle or speed) from the error $e(t)$ as follows: $y(t) = K_p e(t) + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}$. In discrete system whose $t = 0, 1, 2, \ldots$, $y(t) = K_p e(t) + K_i \sum_{\tau=0}^{t} e(\tau) + K_d \Delta e(t)$, where $\Delta e(t) = e(t) - e(t-1)$. The three non-negative coefficients K_p, K_i, and K_d, each determines the degree of activation of different control mechanisms [94], configure a PID controller. We implemented the discrete PID controllers of the lane-keeping system and the adaptive cruise control system in Python embedded in the robot vehicle. The software iteratively calculates the steering angle and speed every 50 ms. It records $o(t)$, $e(t)$, $\sum_{\tau=0}^{t} e(\tau)$, $\Delta e(t)$, and $y(t)$ of both the steering and speed controllers. We released the controller software used in this case study.[6]

5.4.4 Data Collection Strategy

To analyze autonomous driving, we conducted FOTs of the vehicle with numerous possible steering and speed controller configurations. We ran the vehicle FOT with varying independent variables that affect autonomous driving performance, as shown in Figure 5.9. The configuration of the coefficients of the steering and speed PID controllers primarily affects driving performance. However, to limit the orthogonal configuration axes, we fix the K_i and K_d but only vary K_p of the controllers (x- and y-axes). In addition to the controller configurations, the environment is another factor that affects CPS goal achievement but is not under the direct control of the CPS. An external vehicle is a dynamic environment of an autonomous vehicle. Therefore, we also varied the constant speed of the external

[6] Controller software and FOT log data collected from this case study: https://github.com/est-cho/AV-FOT

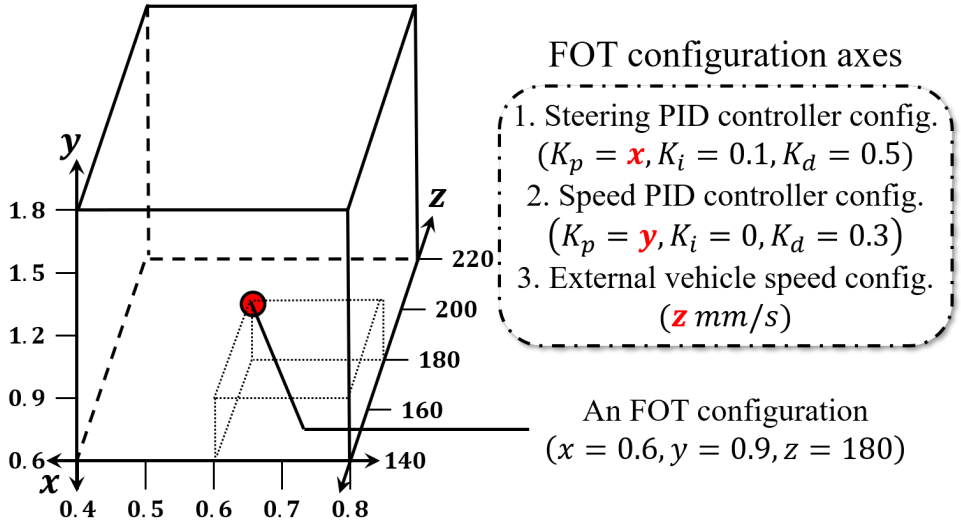


Figure 5.9: Autonomous vehicle FOT configuration space

vehicle (z-axis) during FOTs. Based on a pre-experiment, we set our case study's configuration range and fixed coefficients, as shown in Figure 5.9. Note that the configuration axes are continuous, so it is impossible to experiment with all possible configurations. We discretize the configuration range to five for each axis, so there are 125 ($= 5 \times 5 \times 5$) possible configurations of the autonomous vehicle FOT.

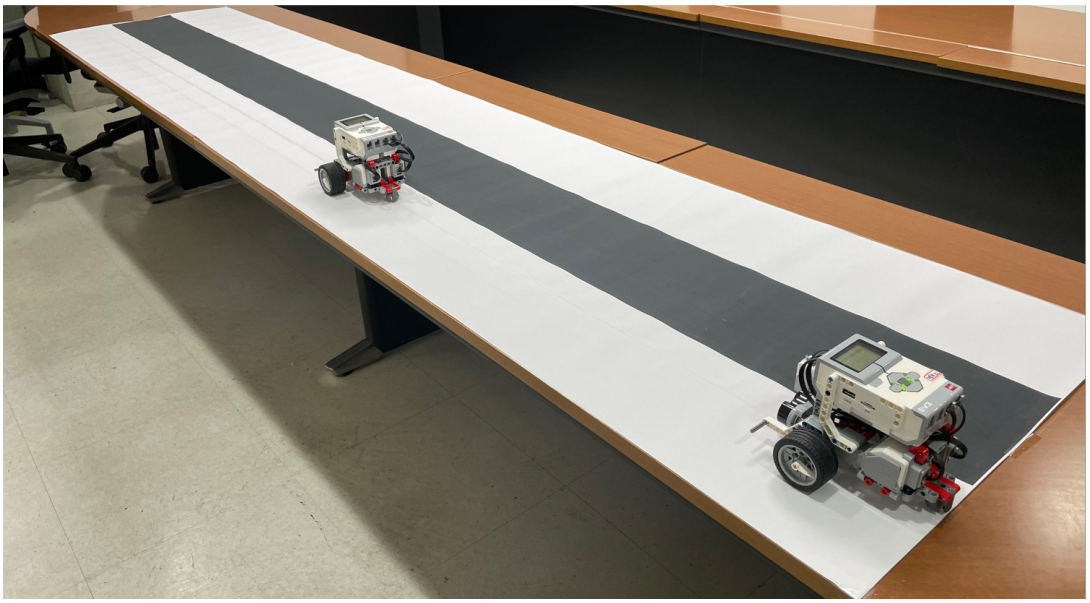


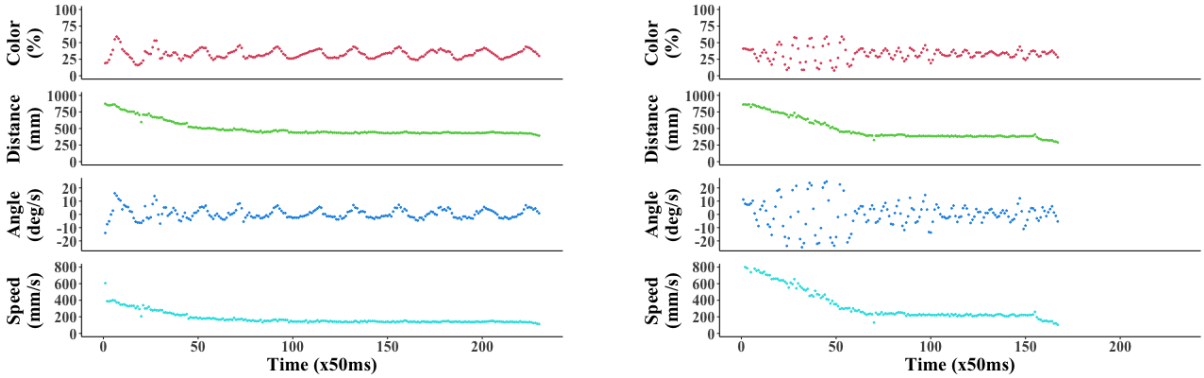
Figure 5.10: Implemented robot vehicles and the FOT environment

Figure 5.10 shows the implemented robot vehicles and the FOT environment. The color goal was 33%, which is the value obtained when sensing the center of the lane in our experimental setting, and the safe distance goal between the two vehicles was 200 mm, which is longer than the length of a robot vehicle. The length of the lane is 3 m, and the distance between the tails of the two vehicles at the start of driving is 1 m. The two vehicles start driving simultaneously, and the experiment ends when the front vehicle arrives at the end of the lane. We keep the rest of the elements as consistent as possible, except for the independent variables under analysis (i.e., FOT configurations). However, since uncertainties may

exist in the physical environment (e.g., sensor noise or non-uniform friction of the lane), we repeated the FOTs of all possible configurations in Figure 5.9 50 times to obtain statistically significant results. The FOT dataset is available in our repository.[6]

5.4.5 FOT Data Analysis

We conducted 50 FOTs for each configuration, taking about 100 hours, and collected 6,250 (= 125 × 50) FOT logs. The volume of the dataset was about 80 MB. The raw data were released on our repository[6]. By analyzing the FOT logs collected by varying independent variables (i.e., configurations), engineers can understand the controllers of the CPS. This section describes the collected FOT logs by analyzing them from three viewpoints.



(a) Config. (x=0.4, y=0.6, z=140)

(b) Config. (x=0.8, y=1.2, z=220)

Figure 5.11: Autonomous vehicle driving trace visualization

Viewpoint 1: Analyzing a single FOT result The driving trace of an FOT is the time-series data of the variables described in Section 5.4.3. Engineers may evaluate a vehicle's driving performance with a specific configuration by analyzing the time-series data. Figure 5.11 visualizes two arbitrary FOT logs. It only visualizes the color and distance observation values, steering angle, and speed control values.

Since the FOT ended when the front vehicle arrived at the end of the lane, the lengths of the FOTs in Figure 5.11 (a) and (b) differ depending on the external vehicle speed z. We also observed that the vehicle controllers continuously adapt the steering angle and speed during driving. Consequently, the observed values of lane color and front distance changed. In the log, we observed that the vehicle moves left and right to keep itself on the lane center as much as possible. In addition, after the autonomous vehicle caught up with the external vehicle, it drove while maintaining a safe distance from the external vehicle. We can observe that the change in configuration results in different shapes of time-series data. In addition, engineers can quantify the driving characteristics of a specific configuration, such as the time to catch up with the front vehicle and the amplitude of the fluctuation of the lane color [95].

Viewpoint 2: Analyzing the FOT results of a configuration There are many FOT logs of the same configuration, so engineers can statistically evaluate the goal achievement of the configuration. Figure 5.12 shows the distribution of the two autonomous driving goal achievements (i.e., lane-keeping and adaptive cruise control) of three arbitrary configurations in terms of the mean squared error (MSE) of lane color and distance time-series data from the goals. A small lane-keeping MSE means driving close

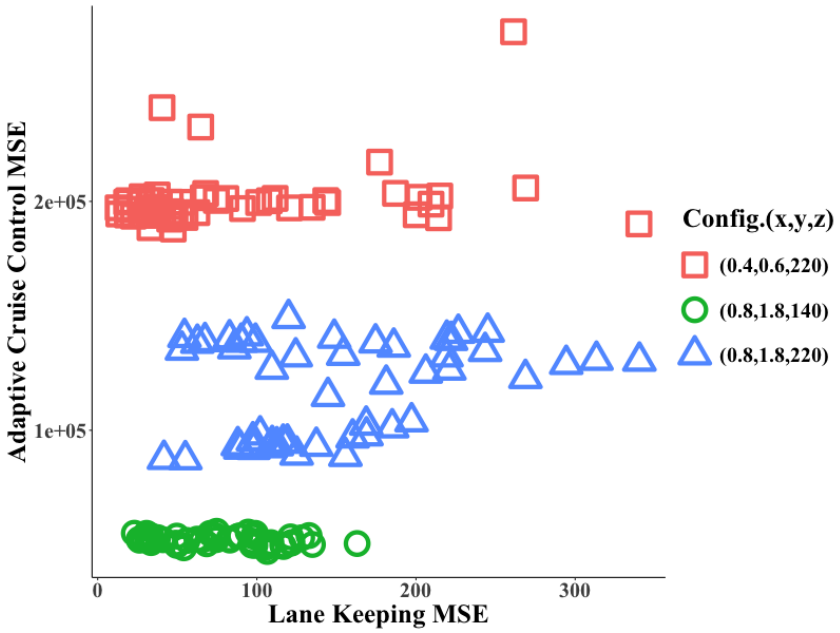


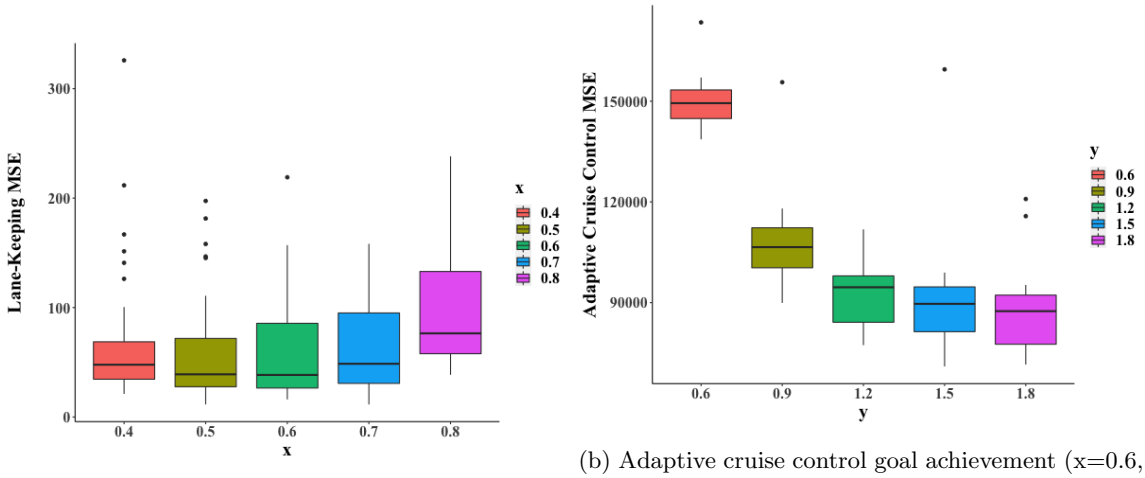
Figure 5.12: Distribution of achievement of two autonomous driving goals obtained through repetitive FOTs

to the center of the lane, and a small adaptive cruise control MSE means catching up with the external vehicle quickly.

In Figure 5.12, we can see that the FOT results were not always the same, even though engineers tested the same configuration. We tried to control other variables as much as possible, except configuration. However, uncertainties (e.g., sensor noise, the direction in which the vehicle was placed manually, or the remaining battery) still affected goal achievement. By analyzing this distribution, engineers could evaluate the consistency of many FOTs of the same configuration, thereby quantifying the degree of the uncertainties that affect the controllers. For example, configuration (0.8, 1.8, 140) appears to be less affected by such uncertainties than the other configurations shown in Figure 5.12. In particular, we can see that the goal achievements are further dispersed by simply increasing the external vehicle speed while remaining in the other configurations. It shows that the degree of uncertainty of the FOT varies depending on not only the CPS's internal configurations but also the environmental configurations.

Viewpoint 3: Analyzing the FOT results of many configurations Engineers can also explore changes in goal achievement by varying configurations to optimize the controllers of the autonomous vehicle. This allows engineers to understand how each configuration axis affects CPS's goal achievement. Figure 5.13 shows how the goal achievements of the lane-keeping system and the adaptive cruise control system change with steering and speed controller configurations, respectively. Configuration axes that were not analyzed were arbitrarily fixed for visualization. Although many FOTs were not deterministic, we could statistically compare different configurations. In Figure 5.13 (a), the steering controller whose K_p (x) was 0.6 performed the best on average when y was 1.5 and z was 200. In Figure 5.13 (b), the adaptive cruise control system achieved its goal better as it increased the K_p of the speed controller (y) when x was 0.6 and z was 180.

Figure 5.14 analyzes the errors of the two autonomous driving goals by simultaneously changing the two configuration axes. Subgraphs (a) and (b) show the MSEs for lane-keeping and adaptive cruise



(a) Lane-keeping goal achievement (y=1.5, z=200)

(b) Adaptive cruise control goal achievement (x=0.6, z=180)

Figure 5.13: Changes in the achievement of autonomous driving goals affected by configurations (one independent variable)

control, respectively. Additionally, the subgraphs also show when the speed of the external vehicle, which is an environmental factor, is 140, 180, and 220. A point in a 3D graph is the MSE average of 50 FOTs.

In Figure 5.14 (a), as both x and y increase, the lane-keeping MSE generally increases. This means that the larger the K_ps of the steering and speed controllers are, the more the vehicle shakes left and right on the lane. Although y was a configuration variable of the adaptive cruise control system, it also affected the performance of the lane-keeping system. In addition, shaking increased as the speed of the external vehicle increased. As shown in Figure 5.14 (b), the adaptive cruise control MSE was primarily affected by the y configuration axis. The larger the y, the smaller the MSE. This finding shows that the autonomous vehicle could catch up to the external vehicle quickly and maintain the distance because the K_p of the speed controller was large. In addition, the faster the external vehicle, the harder it is to maintain the safe distance.

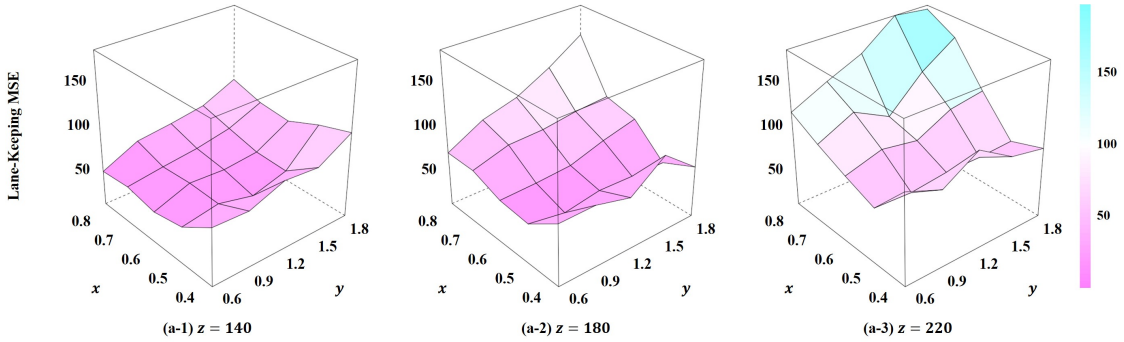
Based on this viewpoint, engineers can understand the trade-off between the goals of the autonomous vehicle and the goal achievements of each configuration. Finally, the controllers can be optimized based on this analysis and knowledge.

Engineers can analyze CPS's behavior and the controller configurations' impact on CPS goals with these various viewpoints. Through the analysis, the engineers obtain knowledge to understand CPS controllers. In addition, based on statistical analysis, many FOT results can provide statistically significant information to engineers.

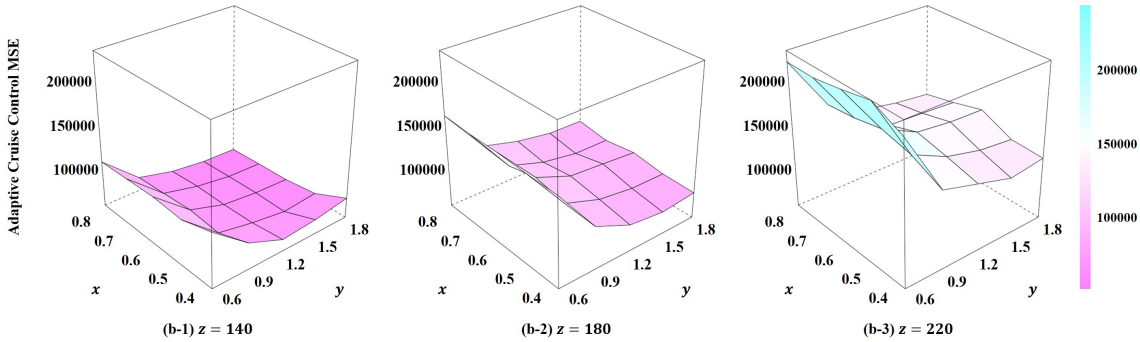
5.4.6 Possible Applications of the Open FOT Dataset

We released the FOT logs collected from our case study[6] for future research on engineering for multi-controller CPS development and FOT engineering. This section introduces some possible applications of the FOT dataset.

Data-driven modeling of the CPS-environment interaction Due to the interaction of the CPS and the environment, both CPS states and environmental states change over time. Accurate modeling



(a) Lane-keeping goal achievement



(b) Adaptive cruise control goal achievement

Figure 5.14: Changes in the achievement of autonomous driving goals affected by configurations (three independent variables)

of the interaction and its effect is the first step for an accurate CPS simulation to reduce the FOT cost [27, 39]. We can automatically extract valuable interaction models from our FOT log dataset [28, 24]. The FOT log shows sequential transitions of the CPS's internal data used for decision-making (e.g., speed and angle control values) and the environmental data observed by sensors (e.g., distance and color sensor values) every 50 ms. In addition, our dataset contains many FOT results of different CPS configurations, so it could also reveal the effect of the configurations on the interaction.

Quantifying uncertainties of multi-controller CPS The uncertainties mentioned earlier stemming from CPS operation in a physical environment and multiple controllers' interdependence (emergent behavior) may cause the CPS to behave contrary to the engineers' expectations. To mitigate the uncertainties, the execution data of CPS may be analyzed further by quantifying uncertainties or extracting causes of variations in goal achievement within a configuration [96, 97]. To quantify uncertainty, enough sample data are needed to obtain statistically significant results from the analysis. Our FOT dataset presents 50 test results per 125 configurations, which provides expansive configuration space and ample test data.

CPS optimization based on data analysis Although the CPS is expected to achieve its goals reliably, we have experienced that goal achievement significantly varies by the configurations of the internal controllers and the external environment. Unfortunately, engineers cannot accurately predict CPS behavior in the real world before runtime. Therefore, the runtime data can optimize the CPS for the operational environment [98], and related studies can use our dataset for this purpose. In particular,

machine learning techniques for optimizing CPS configurations may use our dataset for training [99]. Real-time CPS monitoring and adaptation can also use our dataset by streaming the FOT logs [100, 29]. In addition, the FOT logs are actual multivariate time-series data, so there could be many possible applications [101].

Design of domain-specific FOT methodologies Not limited to the specific applications described above, our dataset and hands-on experience in this paper can guide a domain-specific FOT methodology design [86]. In this paper, we focused on a scenario of repeating many FOTs over the configuration space of a multi-controller CPS. Although this scenario does not represent all situations, we believe that practitioners can design their own FOT methods based on the data and experiences described here.

5.5 Experiment Settings for Environment Model Generation

5.5.1 Overall Experimental Process

Using the experimental environment and FOT logs introduced in the previous sections, this empirical evaluation simulate the whole process of CPS controller verification for each subject systems. Lane-keeping system and adaptive cruise control systems are verified their safety and passenger comfort. The two goals are measured from the operational logs collected by FOTs or simulations using environment models, respectively. After the FOT-based and simulation-based verification results are obtained, those two results are compared to quantify how accurate the simulation-based verification results are. The similarity metric will be introduced in the next section. The point is that the more similar the two verification results, the more accurate the environment model and the model-based verification are.

To answer RQ1, we run ENVI with all possible ENVI parameter settings. There are 18 different configurations of ENVI, so we run ENVI 18 times with different configurations and compare the results in terms of the verification accuracy. Finally, we analyze effective ENVI parameter settings for simulation-based verification.

To answer RQ2, ENVI optimized by RQ1 results is compared with the other environment model generation methods (baselines). They are compared in terms of the verification accuracy. Here the CPS controllers used to collect small seed logs are verified by many simulations.

To answer RQ3, we iteratively run ENVI and the baselines with different volumes of seed logs to investigate the data efficiency for environment model generation.

To answer RQ4, we generate environment models using ENVI and baselines and use the environment models to simulate and verify unseen CPS controllers that have not been used for the seed log collection. We run environment model generation methods including ENVI many times for all possible combinations of seen controller subset and unseen controller under verification, based on the FOT logs collected in Section 5.4. Finally, the verification accuracy of ENVI and baselines are compared.

TO answer RQ5, the all possible combinations of seen controller subset and unseen controller under verification, used in RQ4, are analyzed to search effective seed log collection strategies for unseen controller verification using ENVI. Finally, an effective seed log collection guideline is given for ENVI users.

5.5.2 CPS Goal Verification Accuracy Metric

It is essential to assess the accuracy of the ENVI-based verification for all RQs. To do this, we measure the (dis)similarity between the FOT-based verification and ENVI-based verification results. The more similar the ENVI-based and FOT-based verification results, the better ENVI accurately mimics the real environment, when the same CPS controller is verified.

Specifically, we define an *imitation score* (i.e., *the smaller the better*) of an environment model δ_v as

$$ImitationScore(\delta_v) = D_{KL}(\psi(M_v, \phi) \| \psi(M_r, \phi))$$

where $\psi(M_v, \phi)$ is a simulation-based verification result of CPS controller goals ϕ using δ_v generated by the environment model generation method under analysis (e.g., ENVI), and $\psi(M_r, \phi)$ is an FOT-based verification result on the same goals as a reference. Note that executions of M_v and M_r are nondeterministic as discussed in Chapter 3, so we define $\psi(M_v, \phi)$ and $\psi(M_r, \phi)$ as joint distributions of the passenger comfort and safety assessments obtained from *multiple* simulation and FOT logs, respectively. Though the distributions of goal assessment results can be further analyzed to get a boolean or numeric verification result by statistical verification methods (e.g., SMC) as described in Section 4.5.5, the distributions of the goal assessment results are directly compared to evaluate ENVI more rigorously at a lower level in our experiments. The dissimilarity of $\psi(M_v, \phi)$ and $\psi(M_r, \phi)$ is quantified by Kullback–Leibler divergence (KL divergence, D_{KL}) [102]. $D_{KL}(P\|Q)$ is a measure of divergence of a probability distribution P from a reference distribution Q, widely used in imitation learning [103, 104]. If P is identical to Q, $D_{KL}(P\|Q)$ is zero; the divergence increases as their dissimilarity increases. Thus, the better ENVI mimics the real environment so that $\psi(M_v, \phi)$ is identical to $\psi(M_r, \phi)$, the smaller the KL divergence $D_{KL}(\psi(M_v, \phi)\|\psi(M_r, \phi))$, which is the imitation score.

We interpret the experiment results based on the imitation score to answer the three research questions. In RQ1, we compare different ENVI configurations based on the imitation score. In RQ2, we evaluate the accuracy of ENVI-based verification of seen controllers based on the imitation score. In RQ3, we also analyze the change of the imitation score according to the number of training FOT logs to evaluate how efficient the data-driven model generation is. In RQ4, we evaluate the accuracy of ENVI-based verification of unseen controllers based on the imitation score. In RQ5, we search effective seed log collection strategies based on the imitation score.

5.5.3 Comparison Baselines

From RQ2 to RQ4, we compare ENVI with two alternative data-driven environment model generation approaches using Machine Learning (ML) techniques other than IL. In terms of ML, the environment model generation problem defined in this paper can be seen as a regression problem that infers the future based on the past data. Therefore, engineers can generate the environment model δ_v using regression models without IL. We consider two well-known regression models, i.e., Polynomial Regression (PR) [105] and Random Forest regression (RF) [106]. We used pre-defined PR and RF APIs in Scikit-learn library [107]. All experimental settings except for the parts related to the learning method (e.g., the volume of training data) are the same as ENVI.

In addition to PR and RF, we make a random environment model. The random environment model does not require data or domain knowledge for modeling but changes the environmental state randomly regardless of the previous CPS actions. Injecting random environmental state observation to CPS controllers is often used to verify the possibility of unknown malfunctions of the controllers [108, 109].

However, it does not represent the continuous interaction of the CPS and its operational environment, making the verification result imprecise or rarely reproduced in reality [109]. Therefore, we use the random environment model as another baseline ignoring the CPS-ENV interaction defined in Chapter 3.

5.5.4 Environment Imitation Settings

As described in Chapter 4, the CPS goal verification using ENVI follows five main stages. Remind the second, third, and fourth stages have user-configurable parameters, so we make total 18 ENVI versions for all possible combinations (2 model structures, 3 IL algorithms, and 3 selection criteria) for empirical analysis. In the following subsections, we explain our experimental setup for each stage in detail.

Collecting Seed Logs

For case study 1, we run a robot vehicle with an LKS on a straight road for about 5 seconds for an FOT. At 20 Hz, the following information is recorded in the logs: (1) a lane color value c_t as an environmental state observed by the vehicle's color sensor and (2) a turning rate r_t as a CPS action decided by the vehicle's controller. Therefore, an FOT log is a sequence of state-action pairs $\langle (c_0, r_0), \ldots, (c_T, r_T) \rangle$ where T is the FOT duration.

For case study 2, we run an ego vehicle equipped with an ACCS and another moving front vehicle, one meter apart at the beginning, on a three-meter straight road until the front vehicle reaches the end of the road. One FOT takes about 10 seconds. The ego vehicle records (1) a distance to the front vehicle d_t observed by the distance sensor and (2) a driving speed s_t as a CPS action at 20 Hz. Therefore, an FOT log is a sequence $\langle (d_0, s_0), \ldots, (d_T, s_T) \rangle$ where T is the FOT duration.

For each of the three versions of the two systems, we conduct 50 FOTs, so we collect a total of 150 logs (3 software versions and 50 FOTs) for each subject system. Among the 50 FOT logs of each version of the controllers, 40 logs are used as seed logs, and 10 logs are used for testing (i.e., evaluating) the model. In the seed logs, 20 logs are used for model training, and the remaining 20 are used for model selection (i.e., validation). The dataset of the three versions is used together by ENVI to generate an environment model commonly used to verify the three controller versions. In addition, we perform 5-fold cross-validation, repeating the experiments five times with different splits of the datasets.

Defining Environment Model Structure

As for the model structure, we set the length of history l as 10, meaning that the input of a virtual environment model is a 20-dimensional vector (i.e., a sequence of 10 state-action pairs). Tables 5.4 and 5.5 summarizes structures of the deterministic and nondeterministic environment model structures, respectively. We tried to design the hidden layers straightforward. We decided to use a 1D convolution layer for reducing noise and selecting important features of the time series data and fully connected layers for forward propagation. The deterministic model directly calculates the next environmental state from the historical data. On the other hand, the nondeterministic model calculates a mean and standard deviation of the next state and randomly selects a state based on the normal distribution.

Training Environment Model

We implement IL algorithms using PyTorch library [110]. BC uses the ADAM optimizer [56] to update environment models. Since GAIL needs a policy gradient algorithm to update models, we use a state-of-the-art Proximal Policy Optimization (PPO) algorithm [58]. As for the hyperparameters of the

Table 5.4: Deterministic environment model structure

id	input id	layer	output shape
0		input	(2, 10)
1	0	1D convolution layer	(16, 8)
2	1	Max pooling layer	(16, 4)
3	2	Flatten layer	(64)
4	3	Fully connected layer	(512)
5	4	Fully connected layer	(512)
6	5	Fully connected layer	(1)

Table 5.5: Nondeterministic environment model structure

id	input id	layer	output shape
0		input	(2, 10)
1	0	1D convolution layer	(16, 8)
2	1	Max pooling layer	(16, 4)
3	2	Flatten layer	(64)
4	3	Fully connected layer	(512)
5	4	Fully connected layer	(512)
6	5	Fully connected layer (μ)	(1)
7	5	Fully connected layer (σ)	(1)
8	6, 7	Normal distribution sampler	(1)

Table 5.6: Hyperparameter values for IL algorithms

Algorithm	Hyperparameter	Value
BC	Number of training iteration	500
	Learning rate	0.00005
GAIL	Number of training iteration	500
	Model & discriminator learning rate	0.00005
	PPO num. policy iteration	10
	PPO num. discriminator iteration	10
	PPO reward discount γ	0.99
	PPO GAE parameter λ	0.95
	PPO clipping ϵ	0.2

IL algorithms, we use default values from the original papers [58, 50]. Table 5.6 shows the hyperparameter values used in our evaluation.

Selecting the Best Environment Model

When the IL algorithm is finished, trained δ_vs are evaluated based on the validation logs to select the best one. Based on the three selection criteria under analysis, three δ_vs are chosen.

Verifying CPS Goals

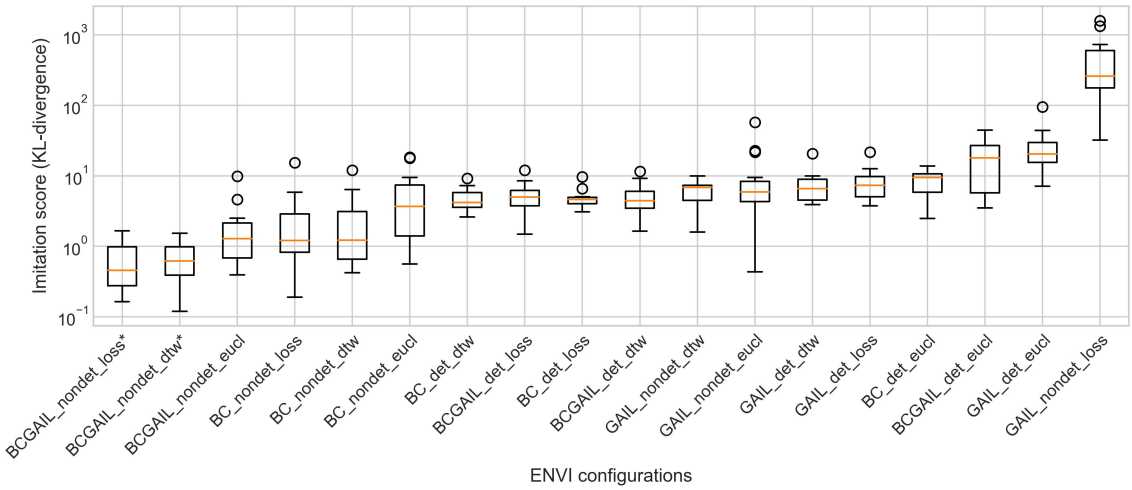
Each version of LKS and ACCS is simulated multiple times with the environment models. For the simulation, the initial inputs of the environment models are given from the testing dataset described in Section 5.5.4. Each simulation log is then used to assess both passenger comfort and safety. As described in Section 5.5.2, the joint distribution of the passenger comfort and safety assessment results based on the multiple simulation logs is the verification result $\psi(M_v, \phi)$. In contrast, the verification result based on the full testing FOT logs is $\psi(M_r, \phi)$.

5.6 Evaluation Results

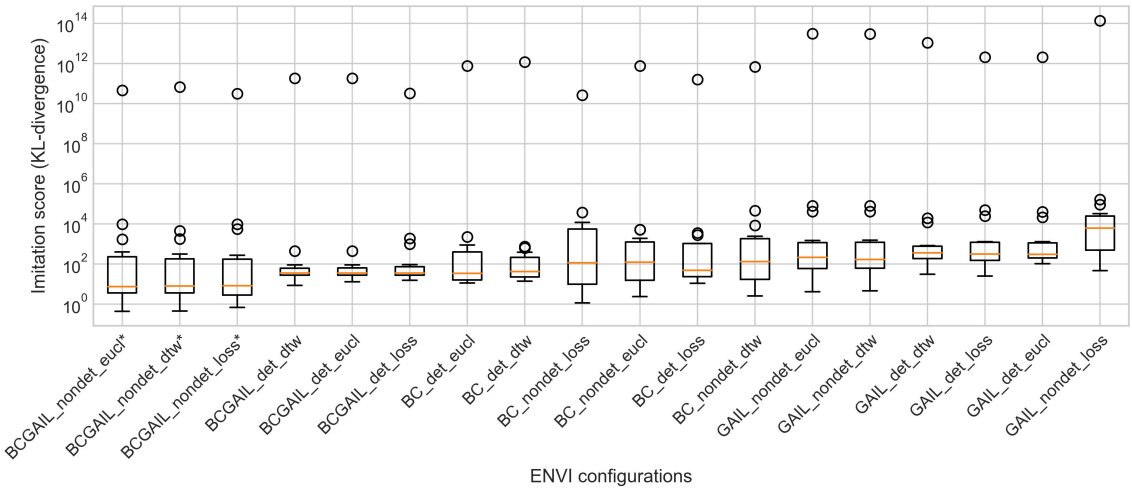
5.6.1 RQ1: User Parameter Analysis

RQ1 aims to investigate the effect of ENVI parameters on the imitation score and suggest optimal configurations of ENVI. To answer RQ1, we make all possible configurations of ENVI parameter settings and compare them statistically in the imitation score.

Figure 5.15 shows the comparison of 18 ENVI configurations in terms of the imitation score in the verification of LKSs (Figure 5.15a) and ACCSs (Figure 5.15b). ENVI is configured by an IL algorithm (*BC*, *GAIL*, or *BCGAIL*), a model structure (deterministic (*det*) or nondeterministic (*nondet*)), and a model selection criterion (1-tick loss (*loss*), Euclidean distance (*eucl*), or *DTW*). For each case study, an ENVI configuration is evaluated 15 times (3 different controller versions and 5-fold cross-validation). The spread of imitation scores of the ENVI configurations is shown on the boxplot in the log scale. For each case study, the ENVI configurations are sorted by the average imitation score. A configuration achieving the smallest imitation score is the optimal.



(a) Case study 1



(b) Case study 2

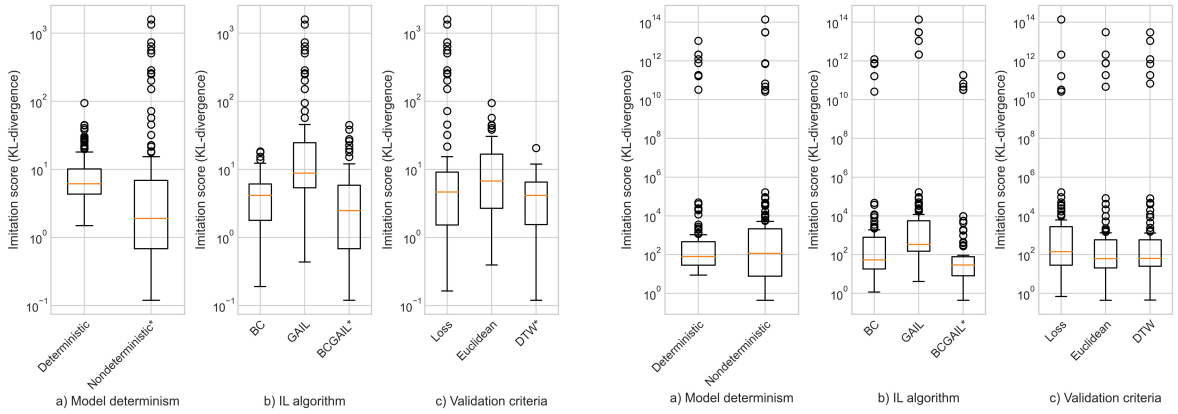
Figure 5.15: Imitation scores of all possible configurations of ENVI. The asterisk (*) highlights a set of optimal configurations with no statistically significant differences.

Figure 5.15 highlights optimal configurations with asterisks (*) that outperform the others in each case study. The Kruskal test selects a set of best configurations that do not have a statistically significant difference in terms of the imitation score. *BCGAIL_nondet_loss* and *BCGAIL_nondet_dtw* are optimal configurations in case study 1. In case study 2, three configurations using *BCGAIL* and *nondeterministic* model structure are optimal regardless of model selection criteria (*loss*, *eucl*, and *dtw*). *BCGAIL_nondet_loss* and *BCGAIL_nondet_dtw* configurations are common optimal in both case studies. It means that ENVI with these configurations could generate the most accurate environment models in both cases. Interesting is that ENVI versions that train *nondeterministic* environment model structure using *BCGAIL* algorithm are the common top-3 configurations in both case studies. It seems that the environment model structures and the IL algorithms greatly influence accuracy of the model, but the

Table 5.7: Confidence level (p-value) of effect of the ENVI parameters on the imitation score and rank of the influence. p-value is highlighted in bold when it is smaller than 0.05.

ENVI parameter	Case study 1	Case study 2	Rank of the influence on the imitation score
Model determinism	**3.201*e*-10**	4.661*e*-01	2
IL algorithm	**3.770*e*-16**	**5.192*e*-12**	1
Model selection criteria	**3.823*e*-04**	4.688*e*-01	3

model selection criteria have less impact on the score.



(a) Case study 1

(b) Case study 2

Figure 5.16: Comparison of imitation scores achieved by different ENVI parameter settings. The asterisk (*) highlights parameter settings that are statistically significantly better than the others.

To further analyze the effect of each ENVI parameter on the model generation, we analyzed the variance of each parameter's imitation score. Figure 5.16 shows the spreads of the imitation score achieved by each ENVI parameter setting, and Table 5.7 summarizes the statistical test (Kruskal test) results of the ENVI parameters' effects on the imitation score. The test's null hypothesis is that the distributions of the imitation score achieved by different ENVI parameter settings are identical. Suppose the p-value of the test is smaller than 0.05. In that case, we can reject the null hypothesis, so the imitation score is highly affected by different ENVI parameter settings. It means that users should carefully configure the ENVI parameter.

Both Figure 5.16 and Table 5.7 show magnitude of the effect of ENVI parameters on the imitation score and the optimal parameter settings. First, the IL algorithm affects the imitation score most significantly in both case studies. Especially, the BCGAIL algorithm generates the environment model most accurately. Following the IL algorithm, the model structure influences the imitation score. In particular, it is generally better to train the nondeterministic environment model in case study 1. In case study 2, the two model structures do not have a significant difference in the average imitation score, but the nondeterministic environment model could achieve a much lower minimum imitation score than the deterministic model. The model selection criteria have the smallest effect on the imitation score in both case studies. All three criteria do not make a difference in the ENVI's performance in case study 2, while using DTW is slightly better in case study 1. This seems to be because DTW can most flexibly compare the noisy FOT and the simulation logs [61].

The RQ1 results first show that the IL algorithm significantly impacts the performance of ENVI. We found BCGAIL algorithm outperforms the other algorithms to generate accurate environment models. Indeed, the BCGAIL algorithm is known to mimic expert behavior better than BC and GAIL [52]. We also confirm that it effectively solves the virtual environment model generation problem. In addition, the nondeterministic environment model structure is more suitable for mimicking the real FOT environment. The real environment state transition is also nondeterministic because CPS FOTs suffer from uncertainties such as sensor noise or non-static road friction. Therefore, the nondeterministic environment model seems appropriate to mimic the uncertain environment. Finally, we can say that the three model selection criteria introduced in Section 4.5.4 have little impact on ENVI's performance, while DTW is recommended in case study 1. However, it also implies that the rationales behind the three criteria are all reasonable.

In RQ1, we empirically suggest a guide to optimize ENVI regarding the imitation score based on two case studies. Although the guide is based on our limited case studies and all the parameters introduced in Section 4.5 are still meaningful in IL, we provide a starting point for the novel use of IL in the environment model generation for CPS goal verification. The following subsections examine how accurate and efficient CPS goal verification using ENVI optimized by *BCGAIL_nondet_dtw* is.

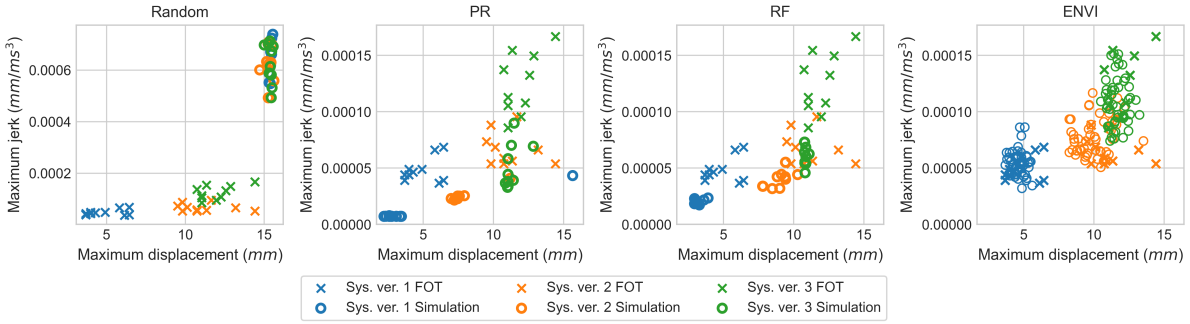
> The answer to RQ1 is that ENVI's imitation score is most influenced by the IL algorithms, followed by the model structures and selection criteria in both case studies. Statistically, *BCGAIL* algorithm, the *nondeterministic* model structure, and *DTW* model selection criterion are first recommended, based on our empirical evaluation.

5.6.2 RQ2: Verification Accuracy for Seen CPS Controllers

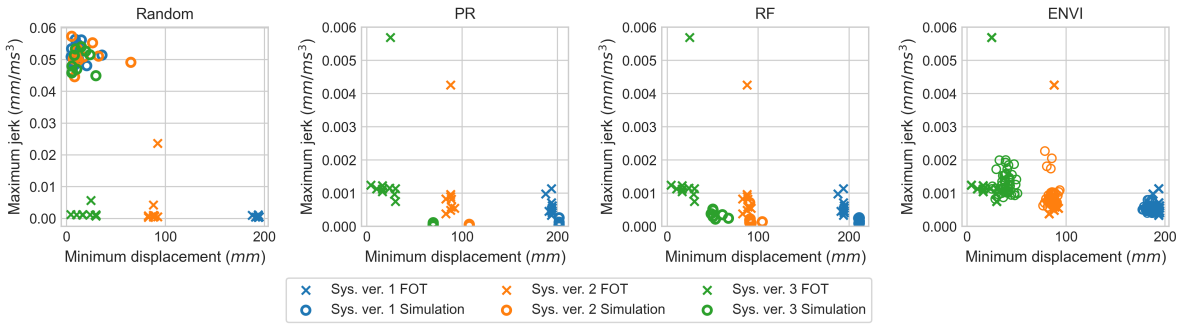
RQ2 aims to investigate how well environment models generated by ENVI mimic the real environments and how accurate the ENVI-based verification is. To answer RQ2, ENVI is configured by an optimal setting found in RQ1 (*BCGAIL_nondet_dtw*). We analyze the ENVI-based verification compared to the baselines.

Figure 5.17 visualizes multiple passenger comfort and safety assessment results obtained from simulations and FOTs. The x-axis of a scatter diagram is the safety measure (i.e., maximum displacement (mm) from the lane center for the LKS and minimum displacement (mm) from the front safety distance for the ACCS), and the y-axis is the passenger comfort measure (i.e., maximum jerk (mm/ms^3) for both case studies). A FOT/simulation-based verification result is visualized as X/O-shaped dots, respectively. As already described in Section 5.5.4, 10 FOT logs for each controller version are used for testing, so there are 10 X-shaped dots of the same color distinguishing the controller version in each scatter diagram. Based on the testing FOT logs, the environment models under comparison simulate the CPS controllers, marked as 10 O-shaped dots of each color. However, ENVI, used here, trains the nondeterministic environment model, so we repeat the simulation five times using the same testing FOT logs to show the results mitigating the nondeterminism, 50 O-shaped dots are shown in the ENVI diagrams. Remind the virtual environment model generation goal is to make the simulation-based verification result similar to the FOT-based results. Therefore, the closer the distributions of the O-shaped dots and the X-shaped dots, the more accurate and realistic the simulation-based verification.

In Figure 5.17, we can see that the distribution of ENVI's verification results more overlapped with the distribution of FOT verification results than the baselines. This shows that ENVI-made environment



(a) Case study 1



(b) Case study 2

Figure 5.17: Comparison of FOT-based and simulation-based passenger comfort and safety verification results

models mimics the real environment well, so the verification results based on the simulations using the model are also realistic compared to the baselines. On the other hand, the vehicle was evaluated as unrealistically unsafe and uncomfortable by the random model, since the environmental state oscillates randomly regardless of the CPS actions. PR and RF models change the verification results depending on the controller versions. However, the distribution of verification results of the two models rarely overlap with the FOT-based results. In particular, the PR and RF models do not properly imitate the uncertainty that emerges in the real world, so even if the test is repeated, many simulation results are mostly the same, which is not the case in reality.

Table 5.8 shows the error of the simulation-based verification to interpret how accurate the simulation-based verification is quantitatively. Simulation-based comfort and safety assessment results, O-shaped dots in Figure 5.17, are compared with corresponding FOT-based assessment results, X-shaped dots in Figure 5.17, and their mean differences are the verification error. As already visually confirmed in Figure 5.17, Table 5.8 shows that ENVI's verification errors are smaller than the baselines for all CPS goals and case studies; it means that the ENVI-based verification is the most accurate. For example, when the real minimum distance between the front vehicle and the adaptive-cruise vehicle under verification is $200mm$, which is the safe distance goal in our experiment, the simulated distance could be about $210mm$ or $190mm$ with the error of $10mm$. Although the influence of the error varies by domain, the error of ENVI-based verification in our case studies is not significant for analyzing the controllers under verification.

In addition to the verification error, the accuracy of the models can be also compared in terms of the imitation score considering the two verification goals together. Figure 5.18 shows the log scale spreads

Table 5.8: Comparison of verification errors of ENVI and baselines. The lowest error for each case study and verification goal is highlighted in bold.

		Verification error	
		safety (mm)	passenger comfort (mm/ms^3)
Case 1	Random	6.04010	0.00055
	PR	2.36947	0.00005
	RF	1.66293	0.00003
	ENVI	**1.18543**	**0.00002**
Case 2	Random	86.06980	0.04900
	PR	28.33626	0.00233
	RF	18.76562	0.00159
	ENVI	**10.49110**	**0.00121**

of the imitation score of ENVI and the baselines. The theoretically lowest (best) imitation score is zero. As already confirmed in the previous results, ENVI achieves the best imitation scores in overall in both case studies. PR and RF are better than the random model but not as effective as ENVI to mimic the real environment well.

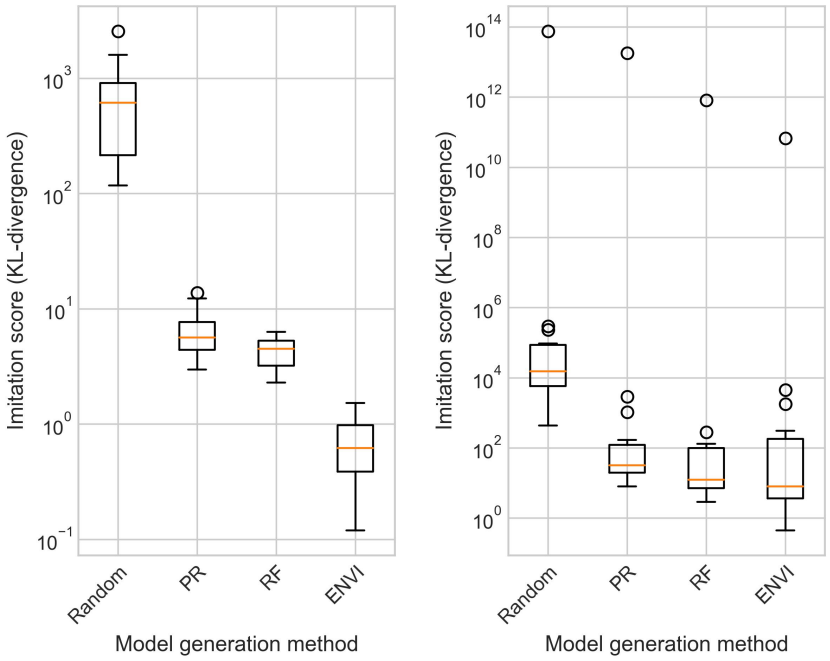
RQ2 results show ENVI can generate virtual environment models that can perform accurate simulation-based verification. When verifying the passenger comfort and safety of the two ADAS using the ENVI-made environment model, the simulation-based verification results were similar to the FOT-based results visually and quantitatively compared to the baselines. Therefore, engineers can accurately verify CPS controllers at a low cost with simulation using ENVI instead of FOT.

The answer to RQ2 is that ENVI can generate accurate environment models from the seed logs. Specifically, the ENVI-based verification results achieves smaller verification error and lower imitation scores than the baselines for all case studies and verification goals. Thus, ENVI makes CPS goal verification efficient by replacing the real environment with the virtual environment model while keeping the verification result similar to reality.

5.6.3 RQ3: Model Generation Efficiency

RQ3 aims to investigate the efficiency of ENVI in terms of the number of FOTs required for collecting the training data for environment model generation. Collecting seed logs is a bottleneck in the ENVI process, which is laborious and challenging to accelerate. Therefore ENVI aims to generate an environment model with as small seed logs as possible. To answer RQ3, we reduce the number of FOT logs given to ENVI as training data from 20 to 1 and analyze the change of imitation score compared to the baselines. Remind 20 training and validation FOT logs of each controller version were given in RQ1 and RQ2 as described in Section 5.5.4. However, the number of FOT logs for training the environment models varies in RQ3, while the testing logs are the same. ENVI in RQ3 also uses the optimal configuration found in RQ1.

Figure 5.19 shows the change in the average imitation score according to the number of FOT logs for training and validation in two case studies. Since the random environment model does not require



(a) Case study 1 (b) Case study 2

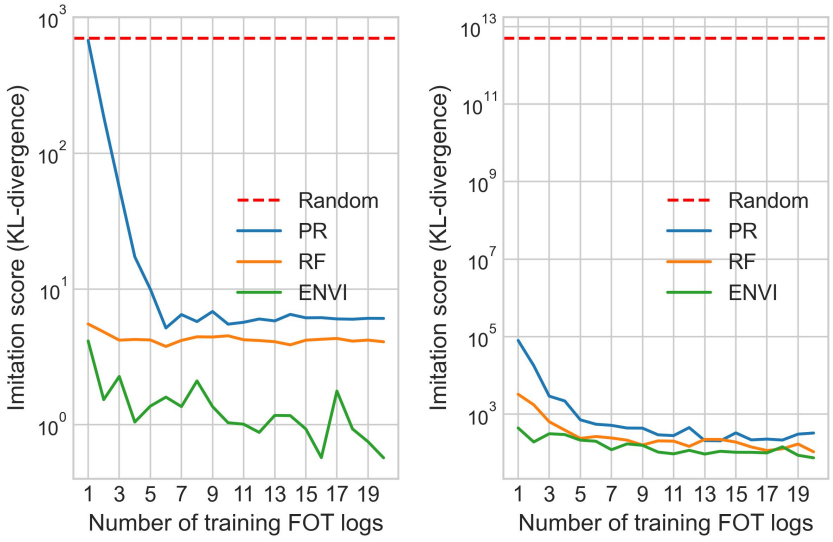
Figure 5.18: Comparison of ENVI and baselines in terms of imitation score of seen controller verification

training data, the score does not vary depending on the number of training FOT logs.

In both case studies, ENVI achieves smaller imitation scores than baselines even when the number of training FOT logs is small overall. In addition, even if ENVI uses only one FOT log for the model generation, it achieves the imitation scores similar to the baselines with 20 FOT logs. It means that the verification accuracy of ENVI with only one FOT log could be similar to the accuracy of PR and RF shown in RQ2. It implies that IL is very efficient in inferring the real environmental behavior from the small data. From this, we can see that even when small FOT logs are available, ENVI can mimic the real environment well. However, using PR- or RF-made environment models is still better than using random models for verifying CPS controllers.

RQ3 results show that ENVI can generate more accurate environment models with a small amount of seed log data than the baselines. Therefore, ENVI is promising to perform accurate simulation-based CPS goal verification using only a small amount of data when the FOT is costly. It will significantly reduce the cost of CPS goal verification. In addition, the baselines are not as efficient as ENVI, even in our simplified case studies, so ENVI is more applicable in practice. However, applying ENVI to the verification of more complex CPS is still one significant future work.

The answer to RQ3 is that ENVI can generate environment models with a small number of seed logs compared to the alternative data-driven environment model generation techniques. Therefore, engineers can reduce the cost of FOTs for collecting training data for the environment model by using ENVI.



(a) Case study 1 (b) Case study 2

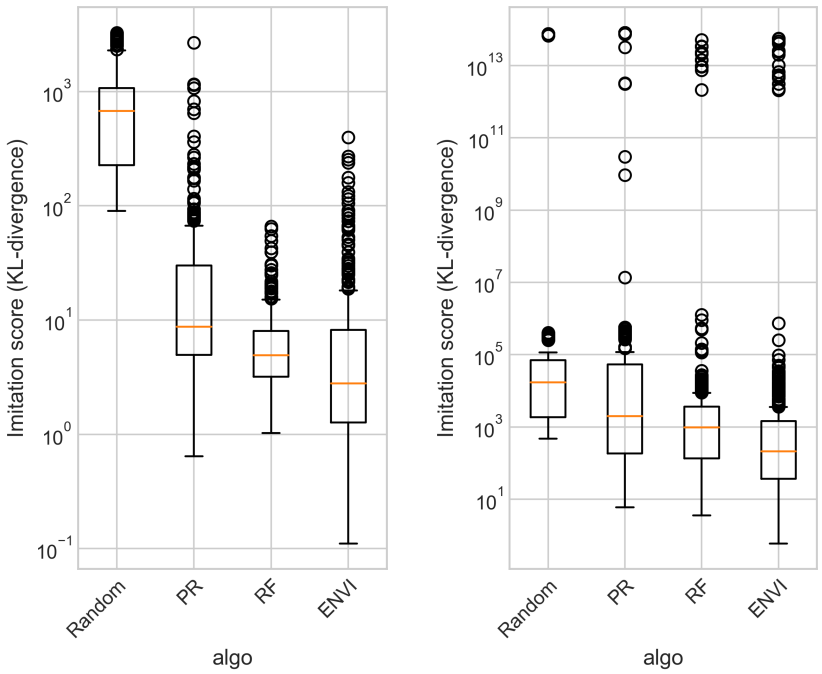
Figure 5.19: Comparison of training data efficiency of ENVI and baselines

5.6.4 RQ4: Verification Accuracy for Unseen CPS Controllers

RQ4 aims to investigate the verification accuracy of unseen controllers using ENVI. The 'unseen controllers' means that CPS controller variations configured by configurations that have not not been used for the seed log collection, as discussed in the extended problem definition of ENVI in Section 3.4.2. Therefore the environment model have not interacted with the unseen controllers during training, but they interact to verify CPS goal achievement of the unseen controllers. To answer RQ4, we select all possible configurations subsets that can be selected from a set of five configurations of interest, and use them to collect seed logs and generate environment models. The generated environment model is used to verify CPS controller variations using the remaining configurations that are not selected in the configuration for seed log collection. The verification results obtained are analyzed and their accuracy is evaluated by imitation score.

Figure 5.20 shows the imitation score of ENVI and baselines for each case study. In both case studies, the imitation scores of ENVI are lower than that of other environmental model generation methods, in general. This shows that ENVI-made environment models are more accurate than the baselines in the verification of unseen controller variations that have never been used for training environment models. This is because IL is more appropriate to infer a general environmental behavior mechanism for whole configuration space than the other methods.

The RQ4 results show that environment models generated using ENVI can also be reused for verification of new CPS controllers that have not previously been FOTed. ENVI achieves the extended problem definition discussed in Section 3.4.2 better than other methods. Recap the environment model should perform accurate verification for all CPS controller configurations of interest. The ENVI-made environment model can be reused for the verification of all configurations of interest. Therefore, ENVI can significantly reduce the CPS goal verification by replacing the FOTs to the simulations using the virtual environment models.



(a) Case study 1 (b) Case study 2

Figure 5.20: Comparison of ENVI and baselines in terms of imitation score of unseen controller verification

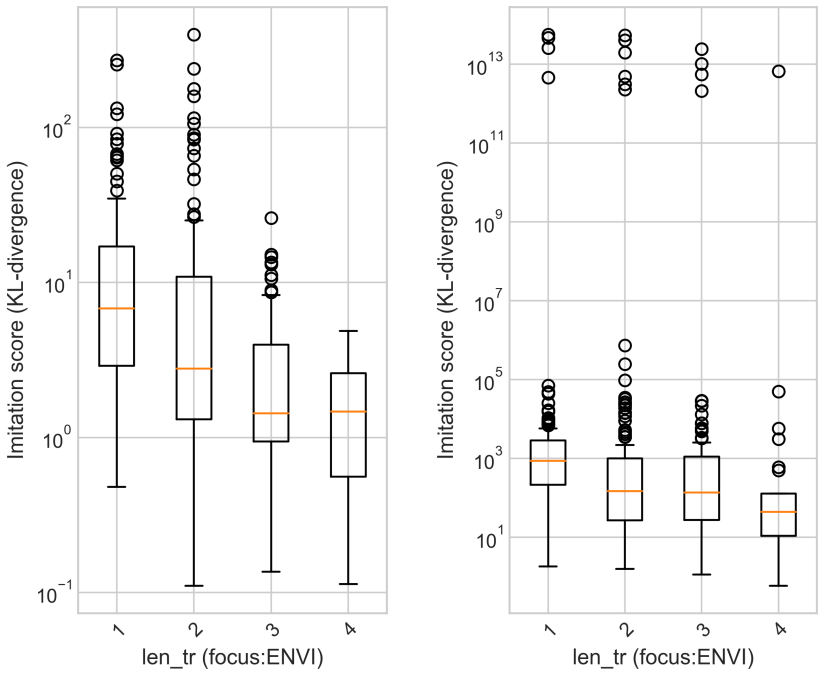
The answer to RQ4 is that ENVI can generate accurate environment models from the seed logs, as shown in RQ2, and the models can be reused to verify unseen CPS controller variations accurately. Specifically, the ENVI-based verification results achieve lower imitation scores than the baselines for all case studies. In other words, ENVI better achieves the extended problem definition of environment modeling than the baselines. Thus, reusing the ENVI-made environment models can significantly reduce the cost of CPS goal verification of all CPS controller configurations of interest.

5.6.5 RQ5: Seed Log Collection Strategy

RQ5 aims to investigate effective seed log collection strategies for unseen controller verification using ENVI. In RQ4, given the same seed logs, we see that the ENVI-made environment models can be reusable for the verification of unseen controllers not used in seed log collection, and the verification results of ENVI were more accurate than other methods. However, the accuracy of the environment model for unseen controller verification is still affected by the configurations used in the seed log collection. For example, when there are CPS controller configurations of interest from No. 1 to 5, an environment model generated from seed logs of No. 1 and No. 3 controller variations and another environment model generated from seed logs of No. 1 and No. 5 variations show different accuracy in the verification of No. 2 contorller variation. Therefore, RQ5 tries to find effective seed log collection strategies for unseen controller verification to guide ENVI users.

To answer RQ5, we generate many environment models from the seed log collection using all possible configurations subsets that can be selected from a set of five configurations of interest, like RQ4. Then

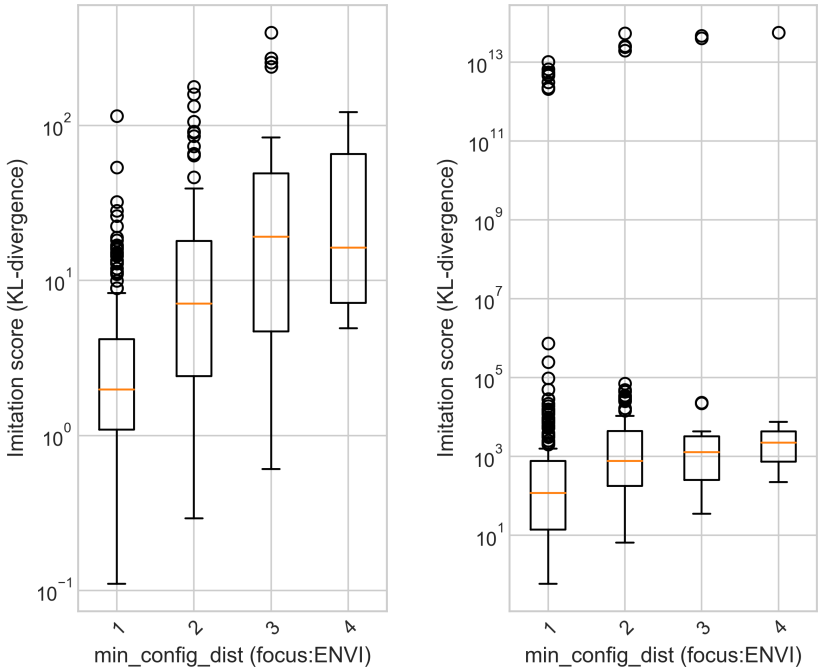
to find effective seed log collection strategies, we label tags indicating a specific situation (i.e., strategy) of seed log collection on each configuration subset. Specifically, we put three different kinds of labels indicating seed log collection strategies. The first label is the number of configurations. From five configurations under experiment for each case system, one to four configurations can be used for seed log collection except at least one configuration under verification. The second label the distance between seen and unseen configurations. How far is the configuration under verification from the configurations used for seed log collection can affects the accuracy of the environment model. The last label is 'interpolation' or 'extrapolation'. Interpolation indicates a configuration under verification is between two or more seen configurations in a continuous configuration space. On the other hand, extrapolation indicates a configuration under verification is outside of the range covered by the seen configurations in the configuration space. For example, if configuration No. 2 and No. 4 were used for seed log collection, verifying No. 3 configuration is the interpolation but verifying No. 1 and No. 5 is the extrapolation. We mark each configuration subset used for seed log collection by these three labels to specify certain seed log collection strategies. We then analyze the change of imitation score according to the labels of seed log collection strategies.



(a) Case study 1 (b) Case study 2

Figure 5.21: Comparison of ENVI imitation score according to number of CPS controllers used for seed log collection

First, figure 5.21 shows the change of ENVI's imitation score according to the number of CPS controller variations used for seed log collection among total five variations under analysis. ENVI generally received lower imitation scores as more controller variations were used to collect seed logs. This means that the more controllers are seen to the environment models during training, the more likely the environment model can accurately interact with unseen controllers. Engineers can accurately verify the unseen controller with the ENVI-made environment model as shown in RQ4, but still can expect to generate more accurate environment models as more controller's FOT logs are collected during training.



(a) Case study 1

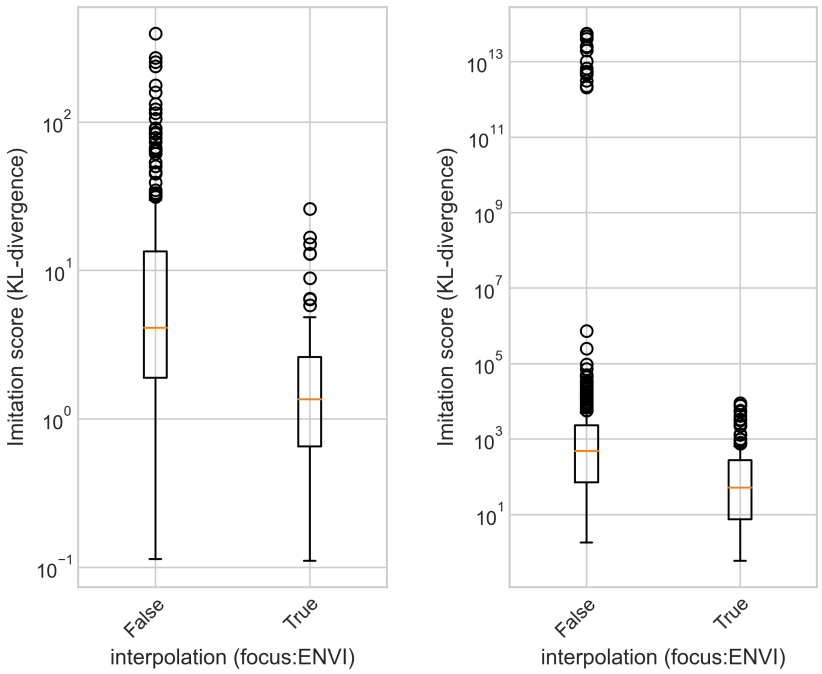
(b) Case study 2

Figure 5.22: Comparison of ENVI imitation score according to the distance between seen configurations and an unseen configuration under verification

Second, figure 5.22 shows the change of ENVI's imitation score according to the distance between the seen and unseen configurations. There can be more than one seen configurations, so the minimum distance between the seen and unseen configurations were used. ENVI obtained a lower imitation score as the distance between the seen and the unseen configurations on the configuration space was closer. This means that engineers can expect higher verification accuracy if they verify the an unseen CPS controller configuration close to the configuration used in the seed log collection.

Last but not least, figure 5.23 compare the ENVI's imitation scores of interpolation and extrapolation verification. The results show that interpolation verification achieves significantly lower imitation scores than extrapolation verification. This means that more accurate verification can be expected when the unseen controller to be verified is within the range of controllers used in the seed log collection. This also reveals a limitation that ENVI's environment model is reusable within the configuration range seen by the seed log within the configuration space of interest but may be less accurate outside the range.

RQ5 provides effective seed log collection strategies in which unseen verification can be accurate from three perspectives. First, a more accurate environment model is generated when more configurations are used for seed log collection. Second, the closer the configurations used for seed log collection and the configuration under verification, the more accurate the ENVI-based verification results. Finally, in the case of interpolation verification, where the controller configuration to be verified is within the range of the seen configuration, a more accurate environmental model is generated. For effective unseen controller verification using ENVI, users are recommended to collect seed logs considering these three guidelines.



(a) Case study 1

(b) Case study 2

Figure 5.23: Comparison of ENVI imitation score of interpolation verification and extrapolation verification

5.7 Threats to Validity

In terms of external validity, our LEGO-lized autonomous vehicle and two driving assistance systems under verification are simplified for representing the real CPS and software controllers. Although they may differ from the real CPS (e.g., autonomous vehicle), it represents CPS software controllers in practice in terms of continuous interaction with the environment. Applying ENVI to more complex CPS could show different results, but the applicability of ENVI for the simulation-based verification shown in this paper is still valid for CPSs with such software controllers. However, additional case studies with more complex CPS are required to improve our results' generalizability.

In terms of internal validity, the goal verification results based on specific autonomous driving goals (e.g., passenger comfort and safety) could be a potential threat since the evaluation of the driving assistance controller's goal could be biased to a specific aspect of driving. To mitigate this threat, in our evaluation, we chose two popular and important goals motivated by industrial standards such as ISO 11270 for LKS [111] and ISO 15622 for ACCS [112] specifying acceptable safety and comfort. We then aggregated the results on both goals to comprehensively understand whether the subject controllers work well or not. Hyperparameter value settings for IL (e.g., number of iterations, learning rates, Etc.) could be another potential threat to the internal validity since the performance of machine learning can largely depend on hyperparameter values. We used the values recommended in the original studies [58, 50]. Nevertheless, hyperparameter tuning is an important research field, so it remains an interesting future work.

5.8 Summary

In this chapter, we evaluated our approach ENVI with two case studies of real CPS goal verification. The case studies verified LEGO-lized autonomous vehicles equipped with a lane-keeping system and an adaptive cruise control system. The case studies evaluated ENVI in five perspectives, 1) impact of ENVI user parameter, 2) verification accuracy of seen controllers, 3) model generation efficiency, 4) verification accuracy of unseen controllers, and 5) effective seed log collection strategies. In summary, the results show that the CPS goal verification using ENVI-made virtual environment models is more accurate than the baselines, even when only a few FOT logs are used for training the models. Therefore, ENVI can reduce the cost of CPS goal verification but remain the verification accuracy.

Chapter 6. Conclusion

6.1 Summary of Achievements

First, we conducted a systematic literature review in environment modeling and investigated how recent studies have described the concepts of the environment. In addition, we explored how the studies represented the environment as models. Following a systematic review protocol, we selected and analyzed 128 primary studies. We provided five common characteristics of the environment, two common sources of environmental uncertainty, and 14 reference environment models. We also identified four common perspectives of the environment specification. Finally, we compared the related environment modeling approach to our approach.

Second, we proposed a formal model of the interaction between the CPS and its environment, a CPS-Environment interaction model, and a formal framework of the CPS goal verification process. We formally defined the environment model generation problem based on the formal framework. The original environment modeling problem definition was extended for a case when the CPS controller under verification is user-configurable software.

Third, we proposed ENVI (ENVironment Imitation), a novel data-driven environment imitation approach that efficiently generates accurate virtual environment models for CPS goal verification. We specifically presented the ENVI process and its user parameters (e.g., model determinism, IL algorithms, and validation criteria). ENVI requires only a few FOTs for training a virtual environment model instead of conducting expensive FOTs many times. An accurate virtual environment model can be automatically generated from the collected FOT logs by leveraging IL algorithms (i.e., BC, GAIL, and BCGAIL).

Finally, we conducted real CPS development and verification case studies to evaluate our novel environment model generation approach ENVI empirically. We examined 1) the impact of ENVI user parameters, 2) verification accuracy for seen CPS controllers, 3) model generation efficiency, 4) verification accuracy for unseen CPS controllers, and 5) the effective seed log collection strategies of ENVI. Based on the evaluation results, we validated that ENVI can efficiently generate accurate environment models for CPS goal verification. Consequently, the cost of CPS goal verification could be reduced.

In addition to evaluating our approach, we tried to contribute to academia by providing a reusable CPS experimental environment and an open CPS FOT dataset. We presented a physical experiment environment called *Platooning LEGOs*, a model problem of platooning technology implemented using LEGOs. In addition, using *Platooning LEGOs*, we designed a reproducible case study to develop a multi-controller CPS and performed its FOTs. The experiment environment and the collected FOT log dataset were released on an open-source repository.

6.2 Discussion

This section discusses some open challenges and future research directions of the virtual environment model generation for CPS goal verification.

First, sample efficiency is essential. This is because conducting FOTs to collect logs is the most expensive task in the data-driven approach. In our experiments, the BCGAIL algorithm was the most

efficient in most cases. Using state-of-the-art techniques for increasing sample efficiency [52, 113] could further help.

Second, it should be robust to noise in FOT logs. Many IL studies assume the correctness of the expert demonstration [114, 60]. However, the *expert* in our problem is the real environment, so some noise is inevitable in the demonstration data (e.g., due to sensor noise). Though we used noisy data collected by the real CPSs in the experiments, systematically investigating the impact of noise was not in the scope of our work. Nevertheless, as many studies have already considered the noise issue in machine learning [115, 116], they could better guide how to address noisy FOT logs in ENVI.

Third, finding a proper level of abstraction for the complex environment is essential. We abstracted the environment as a state-transition function in a closed-loop simulation and recast the model generation problem as the IL problem (see chapter 3). This is a typical level of abstraction for the environment modeling [27, 38]. However, this simple representation may not be sufficient for some domains. Therefore, an extension of the environment model is an interesting future work. We can also refer to some IL studies that imitate complex expert behaviors (e.g., multi-task or concurrent behavior) [117, 118].

Finally, a hybrid of data-driven and knowledge-based environment modeling can make the model further effective. When a high-fidelity simulation engine is based on well-known principles in the CPS domain, engineers can manually create an accurate virtual environment in the simulator. In contrast to such knowledge-based environment modeling, ENVI is a data-driven approach in which only a few seed logs are required to automatically generate an accurate virtual environment model. This is a massive advantage in inferring complex environmental behavior from data. Therefore, ENVI can complement the knowledge-based approach depending on the application domain.

Bibliography

[1] S. Levine, Lecture note of supervised learning of behaviors. cs 285, uc berkeley (May 2018).

[2] D. Weyns, Software engineering of self-adaptive systems: an organised tour and future challenges, Chapter in Handbook of Software Engineering (2017).

[3] A. Filieri, M. Maggio, K. Angelopoulos, N. d'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, et al., Software engineering meets control theory, in: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2015, pp. 71–82.

[4] R. Baheti, H. Gill, Cyber-physical systems, The impact of control technology 12 (1) (2011) 161–166.

[5] D. An, J. Liu, M. Zhang, X. Chen, M. Chen, H. Sun, Uncertainty modeling and runtime verification for autonomous vehicles driving control: A machine learning-based approach, Journal of Systems and Software 167 (2020) 110617. doi:https://doi.org/10.1016/j.jss.2020.110617.
URL https://www.sciencedirect.com/science/article/pii/S0164121220300959

[6] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, S. K. Gupta, Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles, Journal of Systems and Software 137 (2018) 197–215. doi:https://doi.org/10.1016/j.jss.2017.10.031.
URL https://www.sciencedirect.com/science/article/pii/S0164121217302546

[7] D. Bozhinoski, D. Di Ruscio, I. Malavolta, P. Pelliccione, I. Crnkovic, Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective, Journal of Systems and Software 151 (2019) 150–179. doi:https://doi.org/10.1016/j.jss.2019.02.021.
URL https://www.sciencedirect.com/science/article/pii/S0164121219300317

[8] A. Ahmad, M. A. Babar, Software architectures for robotic systems: A systematic mapping study, Journal of Systems and Software 122 (2016) 16–39. doi:https://doi.org/10.1016/j.jss.2016.08.039.
URL https://www.sciencedirect.com/science/article/pii/S0164121216301479

[9] Y.-R. Shiue, K.-C. Lee, C.-T. Su, Real-time scheduling for a smart factory using a reinforcement learning approach, Computers & Industrial Engineering 125 (2018) 604–614. doi:https://doi.org/10.1016/j.cie.2018.03.039.
URL https://www.sciencedirect.com/science/article/pii/S036083521830130X

[10] W. Wang, Y. Zhang, J. Gu, J. Wang, A proactive manufacturing resources assignment method based on production performance prediction for the smart factory, IEEE Transactions on Industrial Informatics 18 (1) (2022) 46–55. doi:10.1109/TII.2021.3073404.

[11] M. Zema, S. Rosati, V. Gioia, M. Knaflitz, G. Balestra, Developing medical device software in compliance with regulations, in: 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015, pp. 1331–1334. doi:10.1109/EMBC.2015.7318614.

[12] K. Fu, Trustworthy medical device software, Public Health Effectiveness of the FDA 510 (2011) 102.

[13] D. Weyns, An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective, John Wiley & Sons, 2020.

[14] A. J. Ramirez, A. C. Jensen, B. H. Cheng, A taxonomy of uncertainty for dynamically adaptive systems, in: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 99–108.

[15] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al., Software engineering for self-adaptive systems: A second research roadmap, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 1–32.

[16] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, J. Peleska, Systems of systems engineering: basic concepts, model-based techniques, and research directions, ACM Computing Surveys (CSUR) 48 (2) (2015) 18.

[17] Y.-M. Baek, J. Song, Y.-J. Shin, S. Park, D.-H. Bae, A meta-model for representing system-of-systems ontologies, in: 2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS), IEEE, 2018, pp. 1–7.

[18] D. Seo, D. Shin, Y.-M. Baek, J. Song, W. Yun, J. Kim, E. Jee, D.-H. Bae, Modeling and verification for different types of system of systems using prism, in: Proceedings of the 4th International Workshop on Software Engineering for Systems-of-Systems, ACM, 2016, pp. 12–18.

[19] Y.-J. Shin, S. Hyun, Y.-M. Baek, D.-H. Bae, Spectrum-based fault localization on a collaboration graph of a system-of-systems, in: 2019 14th Annual Conference System of Systems Engineering (SoSE), IEEE, 2019, pp. 358–363.

[20] M. J. de C Henshaw, Systems of systems, cyber-physical systems, the internet-of-things... whatever next?, Insight 19 (3) (2016) 51–54.

[21] C. Guariniello, A. K. Raz, Z. Fang, D. DeLaurentis, System-of-systems tools and techniques for the analysis of cyber-physical systems, Systems Engineering 23 (4) (2020) 480–491.

[22] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, J.-M. Bruel, Relax: a language to address uncertainty in self-adaptive systems requirement, Requirements Engineering 15 (2) (2010) 177–196.

[23] Y.-J. Shin, L. Liu, S. Hyun, D.-H. Bae, Platooning legos: An open physical exemplar for engineering self-adaptive cyber-physical systems-of-systems, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2021, pp. 231–237.

[24] Z. Ding, Y. Zhou, M. Zhou, Modeling self-adaptive software systems with learning petri nets, IEEE Transactions on Systems, Man, and Cybernetics: Systems 46 (4) (2015) 483–498.

[25] Y.-J. Shin, Y.-M. Baek, E. Jee, D.-H. Bae, Data-driven environment modeling for adaptive system-of-systems, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 2044–2047.

[26] E. M. Fredericks, B. DeVries, B. H. Cheng, Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty, in: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2014, pp. 17–26.

[27] Y. Qin, C. Xu, P. Yu, J. Lu, Sit: Sampling-based interactive testing for self-adaptive apps, Journal of Systems and Software 120 (2016) 70–88.

[28] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, K. Inoue, Learning revised models for planning in adaptive systems, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 63–71.

[29] Y.-J. Shin, E. Cho, D.-H. Bae, Pasta: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems, Fundamental Approaches to Software Engineering 12649 (2021) 292.

[30] D. Budgen, P. Brereton, Performing systematic literature reviews in software engineering, in: Proceedings of the 28th international conference on Software engineering, 2006, pp. 1051–1052.

[31] Z. Stapic, E. G. López, A. G. Cabot, L. de Marcos Ortega, V. Strahonja, Performing systematic literature review in software engineering, in: Central European Conference on Information and Intelligent Systems, Faculty of Organization and Informatics Varazdin, 2012, p. 441.

[32] S. Keele, et al., Guidelines for performing systematic literature reviews in software engineering, Tech. rep., Citeseer (2007).

[33] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering, 2014, pp. 1–10.

[34] M. Salehie, L. Tahvildari, Self-adaptive software: Landscape and research challenges, ACM transactions on autonomous and adaptive systems (TAAS) 4 (2) (2009) 1–42.

[35] S. Monpratarnchai, T. Tetsuo, Applying adaptive role-based model to self-adaptive system constructing problems: a case study, in: 2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, IEEE, 2011, pp. 69–78.

[36] J. Van Der Donckt, D. Weyns, M. U. Iftikhar, S. S. Buttar, Effective decision making in self-adaptive systems using cost-benefit analysis at runtime and online learning of adaptation spaces, in: International Conference on Evaluation of Novel Approaches to Software Engineering, Springer, 2018, pp. 373–403.

[37] G. F. Solano, R. D. Caldas, G. N. Rodrigues, T. Vogel, P. Pelliccione, Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2019, pp. 89–99.

[38] A. Reichstaller, A. Knapp, Risk-based testing of self-adaptive systems using run-time predictions, in: 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2018, pp. 80–89. doi:10.1109/SASO.2018.00019.

[39] W. Yang, C. Xu, Y. Liu, C. Cao, X. Ma, J. Lu, Verifying self-adaptive applications suffering uncertainty, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 199–210. doi:10.1145/2642937.2642999.
URL https://doi.org/10.1145/2642937.2642999

[40] M. Tanabe, K. Tei, Y. Fukazawa, S. Honiden, Learning environment model at runtime for self-adaptive systems, in: Proceedings of the Symposium on Applied Computing, 2017, pp. 1198–1204.

[41] J. Cámara, W. Peng, D. Garlan, B. Schmerl, Reasoning about sensing uncertainty in decision-making for self-adaptation, in: A. Cerone, M. Roveri (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham, 2018, pp. 523–540.

[42] G. A. Moreno, J. Cámara, D. Garlan, B. Schmerl, Flexible and efficient decision-making for proactive latency-aware self-adaptation, ACM Trans. Auton. Adapt. Syst. 13 (1) (apr 2018). doi:10.1145/3149180.
URL https://doi.org/10.1145/3149180

[43] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning, Vol. 135, MIT press Cambridge, 1998.

[44] S. A. Safdar, T. Yue, S. Ali, H. Lu, Evaluating variability modeling techniques for supporting cyber-physical system product line engineering, in: International Conference on System Analysis and Modeling, Springer, 2016, pp. 1–19.

[45] A. Hussein, M. M. Gaber, E. Elyan, C. Jayne, Imitation learning: A survey of learning methods, ACM Comput. Surv. 50 (2) (apr 2017). doi:10.1145/3054912.
URL https://doi.org/10.1145/3054912

[46] O. Michel, Cyberbotics ltd. webots™: Professional mobile robot simulation, International Journal of Advanced Robotic Systems 1 (1) (2004) 5. doi:10.5772/5618.

[47] N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Vol. 3, 2004, pp. 2149–2154 vol.3. doi:10.1109/IROS.2004.1389727.

[48] S. Schaal, Learning from demonstration, in: Advances in Neural Information Processing Systems, 1996, pp. 1040–1046.
URL http://papers.nips.cc/paper/1224-learning-from-demonstration

[49] B. D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robotics and Autonomous Systems 57 (5) (2009) 469–483. doi:https://doi.org/10.1016/j.robot.2008.10.024.
URL https://www.sciencedirect.com/science/article/pii/S0921889008001772

[50] J. Ho, S. Ermon, Generative adversarial imitation learning, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, 2016, p. 4572–4580.

[51] J. Ho, S. Ermon, Generative adversarial imitation learning, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, 2016, p. 4572–4580.

[52] R. Jena, C. Liu, K. Sycara, Augmenting gail with bc for sample efficient imitation learning, arXiv (2020). arXiv:2001.07798.

[53] L. D. Xu, L. Duan, Big data for cyber physical systems in industry 4.0: a survey, Enterprise Information Systems 13 (2) (2019) 148–169. doi:10.1080/17517575.2018.1442934.

[54] M. Rafiq, G. Bugmann, D. Easterbrook, Neural network design for engineering applications, Computers & Structures 79 (17) (2001) 1541–1552. doi:https://doi.org/10.1016/S0045-7949(01)00039-6.
URL https://www.sciencedirect.com/science/article/pii/S0045794901000396

[55] A. Schilling, C. Metzner, J. Rietsch, R. Gerum, H. Schulze, P. Krauss, How deep is deep enough? – quantifying class separability in the hidden layers of deep neural networks, arXiv (2019). arXiv:1811.01753.

[56] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv (2017). arXiv:1412.6980.

[57] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, Advances in neural information processing systems 12 (1999).

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv (2017). arXiv:1707.06347.

[59] Z. Guan, T. Xu, Y. Liang, When will generative adversarial imitation learning algorithms attain global convergence, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 1117–1125.

[60] X. B. Peng, P. Abbeel, S. Levine, M. van de Panne, Deepmimic: Example-guided deep reinforcement learning of physics-based character skills, ACM Trans. Graph. 37 (4) (jul 2018). doi:10.1145/3197517.3201311.
URL https://doi.org/10.1145/3197517.3201311

[61] A. Abanda, U. Mori, J. A. Lozano, A review on distance based time series classification, Data Mining and Knowledge Discovery 33 (2) (2019) 378–412.

[62] A. Legay, B. Delahaye, S. Bensalem, Statistical model checking: An overview, in: International conference on runtime verification, Springer, 2010, pp. 122–135.

[63] T. Bures, D. Weyns, B. Schmer, E. Tovar, E. Boden, T. Gabor, I. Gerostathopoulos, P. Gupta, E. Kang, A. Knauss, et al., Software engineering for smart cyber-physical systems: Challenges and promising solutions, ACM SIGSOFT Software Engineering Notes 42 (2) (2017) 19–24.

[64] S. Engell, R. Paulen, M. A. Reniers, C. Sonntag, H. Thompson, Core research and innovation areas in cyber-physical systems of systems, in: International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, Springer, 2015, pp. 40–55.

[65] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, F. Plasil, An architecture framework for experimentations with self-adaptive cyber-physical systems, in: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2015, pp. 93–96.

[66] S. Gerasimou, R. Calinescu, S. Shevtsov, D. Weyns, Undersea: an exemplar for engineering self-adaptive unmanned underwater vehicles, in: 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2017, pp. 83–89.

[67] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, D. Hughes, Deltaiot: A self-adaptive internet of things exemplar, in: 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2017, pp. 76–82.

[68] F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka, I. Gerostathopoulos, Intelligent ensembles-a declarative group description language and java framework, in: 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2017, pp. 116–122.

[69] M. Provoost, D. Weyns, Dingnet: a self-adaptive internet-of-things exemplar, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2019, pp. 195–201.

[70] J. Wuttke, Y. Brun, A. Gorla, J. Ramaswamy, Traffic routing for evaluating self-adaptation, in: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 27–32.

[71] I. Gerostathopoulos, E. Pournaras, Trapped in traffic? a self-adaptive framework for decentralized traffic optimization, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2019, pp. 32–38.

[72] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, B. Nuseibeh, Dragonfly: a tool for simulating self-adaptive drone behaviours, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2019, pp. 107–113.

[73] G. Moreno, C. Kinneer, A. Pandey, D. Garlan, Dartsim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2019, pp. 181–187.

[74] A. Bennaceur, C. McCormick, J. García-Galán, C. Perera, A. Smith, A. Zisman, B. Nuseibeh, Feed me, feed me: an exemplar for engineering adaptive software, in: 2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2016, pp. 89–95.

[75] C. Bergenhem, S. Shladover, E. Coelingh, C. Englund, S. Tsugawa, Overview of platooning systems, in: Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012), 2012.

[76] D. Martinec, Z. Hurák, Vehicular platooning experiments with lego mindstorms nxt, in: 2011 IEEE International Conference on Control Applications (CCA), IEEE, 2011, pp. 927–932.

[77] E. Kita, H. Sakamoto, H. Takaue, M. Yamada, Robot vehicle platoon experiment based on multi-leader vehicle following model, in: 2014 Second International Symposium on Computing and Networking, IEEE, 2014, pp. 491–494.

[78] E. Kita, M. Yamada, Vehicle velocity control in case of vehicle platoon merging, in: 2019 4th International Conference on Intelligent Transportation Engineering (ICITE), IEEE, 2019, pp. 340–344.

[79] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50.

[80] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, K. M. Göschka, On patterns for decentralized control in self-adaptive systems, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 76–107.

[81] A. Abunei, C. R. Comşa, C. F. Caruntu, I. Bogdan, Redundancy based v2v communication platform for vehicle platooning, in: 2019 International Symposium on Signals, Circuits and Systems (ISSCS), IEEE, 2019, pp. 1–4.

[82] S. Shevtsov, D. Weyns, M. Maggio, Simca* a control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 13 (4) (2019) 1–34.

[83] R. D. Caldas, A. Rodrigues, E. B. Gil, G. N. Rodrigues, T. Vogel, P. Pelliccione, A hybrid approach combining control theory and ai for engineering self-adaptive systems, in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2020, pp. 9–19.

[84] M. Lee, K. Lee, C. Kim, J. Lee, Analytical design of multiloop pid controllers for desired closed-loop responses, AIChE Journal 50 (7) (2004) 1631–1635.

[85] L. Zhang, Applying system of systems engineering approach to build complex cyber physical systems, in: Progress in Systems Engineering, Springer, 2015, pp. 621–628.

[86] Y. Barnard, O. Carsten, Field operational tests: challenges and methods, in: Proceedings of European Conference on Human Centred Design for Intelligent Transport Systems, Eds edn. HUMANIST publications, Lyon, 2010, pp. 323–332.

[87] R. Alur, Principles of cyber-physical systems, MIT press, 2015.

[88] T. Patikirikorala, A. Colman, J. Han, L. Wang, A systematic survey on the design of self-adaptive software systems using control engineering approaches, in: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 33–42.

[89] S. Shevtsov, D. Weyns, Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 229–241.

[90] A. Filieri, M. Maggio, K. Angelopoulos, N. D'ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, et al., Control strategies for self-adaptive software systems, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 11 (4) (2017) 1–31.

[91] R. De Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, et al., Software engineering for self-adaptive systems: Research challenges in the provision of assurances, in: Software Engineering for Self-Adaptive Systems III. Assurances, Springer, 2017, pp. 3–30.

[92] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. A. Müller, H. Giese, R. Rouvoy, E. Rutten, What can control theory teach us about assurances in self-adaptive software systems?, in: Software Engineering for Self-Adaptive Systems III. Assurances, Springer, 2017, pp. 90–134.

[93] J. Cámara, A. V. Papadopoulos, T. Vogel, D. Weyns, D. Garlan, S. Huang, K. Tei, Towards bridging the gap between control and self-adaptive system properties, in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2020, pp. 78–84.

[94] J. C. Doyle, B. A. Francis, A. R. Tannenbaum, Feedback control theory, Courier Corporation, 2013.

[95] T. Cherrett, D. Pitfield, Extracting driving characteristics from heavy goods vehicle tachograph charts, Transportation Planning and Technology 24 (4) (2001) 349–363.

[96] N. Bencomo, Quantun: Quantification of uncertainty for the reassessment of requirements, in: 2015 IEEE 23rd International Requirements Engineering Conference (RE), IEEE, 2015, pp. 236–240.

[97] N. Bencomo, A. Belaggoun, A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty, in: Companion Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 460–463.

[98] R. Al-Ali, L. Bulej, J. Kofroň, T. Bureš, A guide to design uncertainty-aware self-adaptive components in cyber–physical systems, Future Generation Computer Systems 128 (2022) 466–489.

[99] S. Bandaru, A. H. Ng, K. Deb, Data mining methods for knowledge discovery in multi-objective optimization: Part a-survey, Expert Systems with Applications 70 (2017) 139–159.

[100] X. Fei, N. Shah, N. Verba, K.-M. Chao, V. Sanchez-Anguix, J. Lewandowski, A. James, Z. Usman, Cps data streams analytics based on machine learning for cloud and fog computing: A survey, Future generation computer systems 90 (2019) 435–450.

[101] W. W. Wei, Multivariate time series analysis and applications, John Wiley & Sons, 2018.

[102] J. M. Joyce, Kullback-leibler divergence, in: International encyclopedia of statistical science, Springer, 2011, pp. 720–722.

[103] Y. Tsurumine, Y. Cui, K. Yamazaki, T. Matsubara, Generative adversarial imitation learning with deep p-network for robotic cloth manipulation, in: 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), IEEE, 2019, pp. 274–280.

[104] X. Zhang, Y. Li, X. Zhou, J. Luo, Cgail: Conditional generative adversarial imitation learning—an application in taxi drivers' strategy learning, IEEE Transactions on Big Data (2020).

[105] E. Ostertagová, Modelling using polynomial regression, Procedia Engineering 48 (2012) 500–506.

[106] M. R. Segal, Machine learning benchmarks and random forest regression (2004).

[107] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[108] E. M. Fredericks, Automatically hardening a self-adaptive system against uncertainty, in: 2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2016, pp. 16–27.

[109] W. Yang, C. Xu, M. Pan, C. Cao, X. Ma, J. Lu, Journal of Systems and Software 138 (2018) 82–99. doi:https://doi.org/10.1016/j.jss.2017.12.009, [link].
URL https://www.sciencedirect.com/science/article/pii/S0164121217303023

[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, Curran Associates Inc., Red Hook, NY, USA, 2019, pp. 8026–8037.

[111] Intelligent transport systems — lane keeping assistance systems (lkas) — performance requirements and test procedures, Standard, International Organization for Standardization (May 2014).

[112] Intelligent transport systems — adaptive cruise control systems — performance requirements and test procedures, Standard, International Organization for Standardization (2018).

[113] X. Zhang, Y. Li, Z. Zhang, Z.-L. Zhang, f-gail: Learning f-divergence for generative adversarial imitation learning, in: H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 12805–12815.
URL https://proceedings.neurips.cc/paper/2020/file/967990de5b3eac7b87d49a13c6834978-Paper.pdf

[114] M. Abdou, H. Kamal, S. El-Tantawy, A. Abdelkhalek, O. Adel, K. Hamdy, M. Abaas, End-to-end deep conditional imitation learning for autonomous driving, in: 2019 31st International Conference on Microelectronics (ICM), 2019, pp. 346–350. doi:10.1109/ICM48031.2019.9021288.

[115] S. Gupta, A. Gupta, Dealing with noise problem in machine learning data-sets: A systematic review, Procedia Computer Science 161 (2019) 466–474, the Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia. doi:https://doi.org/10.1016/j.procs.2019.11.146.
URL https://www.sciencedirect.com/science/article/pii/S1877050919318575

[116] Z. Zeng, Y. Liu, W. Tang, F. Chen, Noise is useful: Exploiting data diversity for edge intelligence, IEEE Wireless Communications Letters 10 (5) (2021) 957–961. doi:10.1109/LWC.2021.3051688.

[117] S. Agrawal, M. van de Panne, Task-based locomotion, ACM Trans. Graph. 35 (4) (jul 2016). doi:10.1145/2897824.2925893.
URL https://doi.org/10.1145/2897824.2925893

[118] A. Singh, E. Jang, A. Irpan, D. Kappler, M. Dalal, S. Levinev, M. Khansari, C. Finn, Scalable multi-task imitation learning with autonomous improvement, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 2167–2173. doi:10.1109/ICRA40945.2020.9197020.

Acknowledgments in Korean

저의 박사 과정을 함께 하신 하나님께 영광 돌립니다. 제게 유익한 환경과 선한 동역자들을 주시고, 부족한 저를 들어 여기까지 오게 하시니 감사합니다. 이 성취는 하나님의 인도하심으로 얻은 것입니다. "너희는 가만히 있어 내가 하나님 됨을 알지어다 내가 열방과 세계 중에서 높임을 받으리라" - 시편 46장 10절.

훌륭한 연구자가 될 수 있도록 박사 과정 동안 지도해주신 많은 분께 진심으로 감사드립니다. 6년간 연구자의 자세를 가르쳐 주시고 최선으로 지도해주신 배두환 지도교수님께 깊은 감사를 드립니다. 항상 저를 믿어주시고 격려해주셔서 이렇게 성장할 수 있었습니다. 교수님의 가르침과 인품을 따라 교수님처럼 존경받는 연구자가 되도록 최선을 다하겠습니다. 여러 프로젝트와 학위 연구에서 연구의 근본적인 가치와 실효에 대해 생각하게 해주시고 방향성을 제시해주신 고인영 교수님, 더 성숙한 학위 연구가 되도록 예리한 통찰력으로 조언해주신 유신 교수님, 현업 선도자의 눈으로 학위 연구의 가치를 알려주시고 미래를 꿈꾸게 해주신 민상윤 교수님께도 감사드립니다. 긴 시간 동안 기술적 조언과 함께 학위 연구의 방향을 잡아주시고 연구자로서의 진로 고민을 함께해주신 존경하는 선배 연구자 신동환 교수님께 정말 감사드립니다. 따듯한 연구실 분위기를 만들어 주시고 연구의 모든 것을 세세하게 가르쳐주신 지은경 교수님 감사드립니다. 교수님들의 지도를 잊지 않고 자랑스러운 제자이자 후배 연구자가 되도록 노력하겠습니다.

박사 과정 동안 여러 일을 함께 겪어 온 연구실 구성원들 감사합니다. 연구를 위해 많은 고민을 함께하고 즐거운 연구실 생활을 할 수 있게 해준 동기 수민이형 감사합니다. 많은 일들을 나누며 학위를 잘 마칠 수 있도록 서로 격려하고 배려해준 상원 감사합니다. 학위 과정 동안 많은 논문을 같이 쓰며, 소중한 연구 가이드라인들을 가르쳐준 선배 영민이형 감사합니다. 멋있게 학위 과정을 마치고 연구자로서 살아가는 모습을 보여준 선배 지영누나 감사합니다. 항상 따듯한 말로 격려해주고 많은 고민을 나누며 박사 과정을 함께한 Zelalem 감사합니다. 대학원에 첫발을 내디뎠을 때, 따듯하게 맞아주시고 이곳에 잘 적응할 수 있도록 해주신 민규형, 유림누나, 치우형 감사합니다. 뛰어난 실력으로 많은 프로젝트를 함께 한 스파이럴 연구실의 낙원 선배, 종인 선배 감사합니다. 그리고 부족한 선배의 연구 지도를 따라 열심히 해준 성진, 승철, 화기애애한 연구실 분위기를 만들어준 Lingjun, Anthony, May 고맙습니다. 이 박사 학위 논문의 한 챕터가 된 서베이를 함께 수행해준 준영 고맙습니다. 학부생 때부터 함께 연구해오고, 이제는 실력과 열정으로 본인의 연구를 수행하고 있는 은호 고맙고, 응원합니다. 졸업 과정을 함께하며, 바쁜 선배들을 배려해주고 이해해준 미현, 한수 진심으로 고맙습니다. 인생의 소중한 시간에 함께 고생하고 성장한 여러분들의 앞날이 복되도록 기도하겠습니다.

이 기쁨을 온전히 누려야 할 주인공인 가족들에게 감사합니다. 항상 겸손히 살아가라고 말씀해주시며 가족을 위해 헌신하신 아버지 감사합니다. 사랑으로 손수 교육해주시고 저의 모든 성장 과정을 만들어주신 어머니 감사합니다. 다 적을 수 없는 부모님의 사랑과 가르침 덕분에 제가 박사 과정까지 잘 마칠 수 있었습니다. 지금껏 받은 은혜 기억하며 살고 자랑스러운 아들이 되겠습니다. 사랑합니다. 박사 과정을 잘 마칠 것이라 든든히 믿어주고 응원해준 멋진 지혜 감사합니다. 학위 과정 중인 저를 사위로 맞아주시고 아껴주신 장인어른, 장모님 감사합니다. 사랑 많은 두 분의 축복으로 가정을 이루고, 함께 만날 때마다 행복한 에너지를 얻어 이 과정을 잘 마칠 수 있었습니다. 사랑합니다. 그리고 저를 활기차게 응원해주고 졸업을 내 일처럼 기뻐해 준 예림, 예원, 예빈 처제들 감사합니다. 앞으로 가족들에게 받은 사랑을 돌려드리며 살겠습니다.

마지막으로 사랑하는 아내 예선이에게 감사합니다. 대학에서부터 지금까지 모든 과정을 가장 가까이서 지켜보며 모든 순간에 저를 지지하고 응원해줘서 감사합니다. 박사 과정 동안 느낀 불안감, 막연함, 좌절감, 후회, 설렘, 기쁨 등 모든 감정을 같이 느끼며 지금까지 오느라 너무 고생 많았습니다. 제 옆에서 항상 함께 해줬기 때문에 이 박사 학위를 받을 수 있었습니다. 하루하루를 열심히 살아갈 이유이자 힘이 되어줘서 정말 고맙고, 이제 제가 우리 가정의 힘이 되겠습니다. 앞으로의 여정에서도 지금까지처럼 언제나 행복합시다. 사랑합니다.

Curriculum Vitae

Name : Yong-Jun Shin

Date of Birth : June 01, 1994

Birthplace : Republic of Korea

Educations

2010. 3. – 2013. 2. Dongsan Christian High School

2013. 3. – 2017. 2. Handong Global University (BS)

2017. 3. – 2023. 2. Korea Advanced Institute of Science and Technology (KAIST) (Ph.D.)

Publications

1. **Yong-Jun Shin**, Donghwan Shin, Doo-Hwan Bae. "Environment Imitation: Data-Driven Environment Model Generation Using Imitation Learning for Efficient CPS Goal Verification." arXiv preprint arXiv:2204.06799 (2022).
2. Esther Cho, **Yong-Jun Shin**, Sangwon Hyun, Hansu Kim, Doo-Hwan Bae. "Automatic Generation of Metamorphic Relations for a Cyber-Physical System-of-Systems Using Genetic Algorithm." 29th Asia-Pacific Software Engineering Conference (APSEC), 2022
3. **Yong-Jun Shin**, Esther Cho, Hansu Kim, Doo-Hwan Bae. "Hands-On Field Operational Test Dataset of a Multi-Controller CPS: A Modeled Case Study on Autonomous Driving." 17th Annual System of Systems Engineering Conference (SoSE), 2022.
4. **Yong-Jun Shin**, Joon-Young Bae, Doo-Hwan Bae. "Concepts and Models of Environment of Self-Adaptive Systems: A Systematic Literature Review," 2021 28th Asia-Pacific Software Engineering Conference (APSEC), 2021, pp. 296-305
5. Young-Min Baek, Eunho Cho, **Yong-Jun Shin**, Doo-Hwan Bae. "A Modeling Method for Representation of Geographical Information of a System-of-Systems." 2021 16th International Conference of System of Systems Engineering (SoSE). IEEE, 2021.
6. Seungchyul Shin, Sangwon Hyun, **Yong-Jun Shin**, Jiyoung Song, Doo-Hwan Bae. "Uncertainty based Fault Type Identification for Fault Knowledge Base Generation in System of Systems." 2021 16th International Conference of System of Systems Engineering (SoSE). IEEE, 2021.
7. **Yong-Jun Shin**, Lingjun Liu, Sangwon Hyun, Doo-Hwan Bae. "Platooning LEGOs: An Open Physical Exemplar for Engineering Self-Adaptive Cyber-Physical Systems-of-Systems." 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, 2021. **Best Artifact Paper Award**
8. **Yong-Jun Shin**, Eunho Cho, Doo-Hwan Bae. "PASTA: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems." International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, 2021.

9. Seungchyul Shin, Sangwon Hyun, **Yong-Jun Shin**, Jiyoung Song, Doo-Hwan Bae. "Manifestation Location-based Classification of Uncertainty Factors Considering Characteristics of System-of-Systems." KIISE Transactions on Computing Practices (2020): 451-457.
10. Sangwon Hyun, **Yong-Jun Shin**, Doo-Hwan Bae. "Analysis of Utilization Methods of the Statistical Model Checking Results for Localizing Faults on System of Systems." Journal of KIISE (2020): 380-386.
11. Young-Min Baek, Zelalem Mihret, **Yong-Jun Shin**, Doo-Hwan Bae. "A Modeling Method for Model-based Analysis and Design of a System-of-Systems." 2020 27th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2020.
12. Sumin Park, **Yong-Jun Shin**, Sangwon Hyun, Doo-Hwan Bae. "Simva-sos: Simulation-based verification and analysis for system-of-systems." 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE). IEEE, 2020.
13. Eunho Cho, **Yong-Jun Shin**, Eunkyoung Jee, Doo-Hwan Bae. "Comparative analysis of fault-attack tree based safety and security assessment approaches." Korea Software Congress (2019): 299-301.
14. Sangwon Hyun, **Yong-Jun Shin**, Doo-Hwan Bae. "Analysis of Utilization Methods of Statistical Model Checking Results for Localizing Faults on System of Systems." Korea Computer Congress (2019): 380-386. **Best Paper Award**
15. **Yong-Jun Shin**, Sangwon Hyun, Young-Min Baek, Doo-Hwan Bae. "Spectrum-based fault localization on a collaboration graph of a system-of-systems." 2019 14th Annual Conference System of Systems Engineering (SoSE). IEEE, 2019.
16. **Yong-Jun Shin**, Sangwon Hyun, Young-Min Baek, Doo-Hwan Bae. "Data-driven environment modeling for adaptive system-of-systems." Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. 2019.
17. Young-Min Baek, Sumin Park, **Yong-Jun Shin**, Doo-Hwan Bae. "Analysis of Case Scenario to Develop a System of Systems Meta-model for Ontology Representation." Journal of KIISE (2018): 1056-1070.
18. Tae-Hwan Kim, Eunho Cho, **Yong-Jun Shin**, Doo-Hwan Bae. "Data-Driven Traffic Environment System-Dynamics Model Generation & Inference Method." Korea Software Congress (2018): 1725-1727. **Outstanding Paper Award**
19. Young-Min Baek, Jiyoung Song, **Yong-Jun Shin**, Sumin Park, Doo-Hwan Bae. "A meta-model for representing system-of-systems ontologies." 2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS). IEEE, 2018.
20. **Yong-Jun Shin**, Sumin Park, Young-Min Baek, Doo-Hwan Bae. "Scenario-based Analysis of System-of-Systems Meta-model and Applicability Analysis for Statistical Verification." 20th Korea Conference on Software Engineering (2018): 1056-1070.
21. Young-Min Baek, Sumin Park, **Yong-Jun Shin**, Doo-Hwan Bae. "Development of Ontology-based System-of-Systems Meta-model Based on the Analysis of SoS Case Scenario." 20th Korea Conference on Software Engineering (2018): 1056-1070. **Best Paper Award**
22. Do Hyun Kim, Jung Eun Kim, Ji Hag Song, **Yong-Jun Shin**, Sung Soo Hwang. "Image-based Intelligent Surveillance System Using Unmanned Aircraft." Journal of Korea Multimedia Society

20.3 (2017): 437-445.

23. **Yong-Jun Shin**, Jiyong Yang, Changbeom Choi. "Research on Flexible Method for Simulation Initialization using C-Interpreter." Korean Institute of Industrial Engineers (2015): 4255-4260.