# Dynamic Safety Assurance of Autonomous Cyber-Physical Systems

By

Shreyas Ramakrishna

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

August 12, 2022

Nashville, Tennessee

Approved:

Dr. Abhishek Dubey, Ph.D.

Dr. Janos Sztipanovits, Ph.D.

Dr. Gabor Karsai, Ph.D.

Dr. Xenofon, Koutsoukos, Ph.D.

Dr. Arun Ramamurthy, Ph.D.

Dr. Ayan Mukhopadhyay, Ph.D.

# ACKNOWLEDGMENTS

This research was supported by the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. I am indebted to my advisor, Prof. Abhishek Dubey, for giving me an opportunity to pursue my doctoral study. His guidance and support has been instrumental throughout the study program. I would also like to express my gratitude to my committee members Prof. Janos Sztipanovits, Prof. Gabor Karsai, Prof. Xenofon Koutsoukos, Dr. Arun Ramamurthy, and Dr. Ayan Mukhopadhyay, for their technical guidance and insightful feedback that greatly impacted this work. A special thanks to Prof. Gabor Karsai for giving me an opportunity to work on the Assured Autonomy project. I would also like to thank Nagabhushan Mahadevan and Dr. Daniel Stojcsics, who have always been available for pep talks and discussions regarding the project.

I would also like to thank Prof. Aniruddha Gokhale for the guidance and the excellent time we had during my first conference in Spain. A special thanks to Dr. Arun Ramamurthy for giving me an opportunity to intern at Siemens Corporation, Technology for the DARPA ARCOS project. This experience gave me insights into industrial research. Next, I would like to thank all my friends at ScopeLab and Institute for Software Integrated Systems for the many discussions and collaborations and for making this a memorable journey. I would also like to thank all my collaborators for working with me and guiding me throughout this journey.

I am incredibly grateful to my family, without whom I could never have pursued a doctoral degree. Thank you for your immense support and belief in me. Lastly, I would like to thank the almighty for bestowing this opportunity upon me and giving me the strength and conviction to complete the task.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Cyber-Physical Systems and Autonomy

Cyber-Physical System (CPS) is a physical and engineered system whose operations are monitored, controlled, and integrated by a computing device with communication capability [6]. While there are several other definitions, they share the same underlying principle that a CPS uses the powerful logic of the computing device to monitor and control the dynamics of the physical system. The recent proliferation of low-cost sensors, high-capacity and small form-factor computing devices, and the revolution in wireless communication has made CPSs ubiquitous in our everyday application sectors like automobiles, aerospace, manufacturing and infrastructures, healthcare, and the military. However, increased complexity, domain interdependence, and the dynamic operating nature of modern systems have raised a safety concern [6, 7, 8, 9].

This concern is further aggravated by data-driven approaches such as artificial intelligence and machine learning being used to design CPSs. These methods have a fundamental advantage of handling high-dimensional state-space and learning control actions from data rather than models. This capability of the data-driven approach has resulted in the wide usage of machine learning (ML) techniques like deep learning [10] and reinforcement learning (RL) [11] being used as perception and planning components in several CPS applications such as autonomous vehicles (AV) [4], unmanned underwater vehicle (UUV) [12, 13], autonomous industrial robots [14, 15], and unmanned aerial vehicle (UAV) [16, 17]. These components trained using ML approaches are popularly referred to as learning enabled components (LECs). CPS operating with LECs is referred to as "autonomous CPS" in this work.

An example of autonomous CPS control architecture for perception and control can be seen in Fig. 1.1. A perception component (e.g., LEC) observes the environment using different sensors (e.g., camera, lidar, radar), interprets the sensor readings to generate target actions (e.g., trajectory), and provides the interpreted information to the controller. The controller uses the interpreted sensory information with additional feedback from the plant to apply a control action (steer, throttle, and brake) to the plant, which changes the physical state of the plant. This feedback loop is referred to as the inference cycle, which is performed continuously to continue the system operation.

These autonomous systems have shown exceptional performances for tasks such as autonomous driving [18, 4], object detection [19, 20], and image classification [21, 22]. However, accidents such as Tesla's autopilot

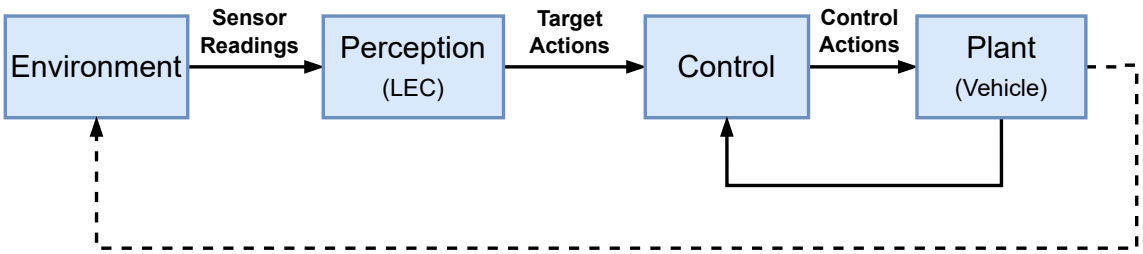



Figure 1.1: Control architecture of an autonomous CPS.

crash [23] and Uber's self-driving car accident [24] illustrate the technical debt [25] of the LEC that is leading us to designs that are not safe. The primary reason for this mistrust is the black-box nature of ML components, which hinders the use of classical software testing strategies (e.g., code coverage, function coverage) and verification approaches (e.g., model checking, reachability analysis). There has been some progress in designing tools and techniques for testing [26, 27] and verifying [28, 29] these components. However, widespread applicability remains questionable. The uncertainty, inaccuracy, and non-determinism introduced by the LECs make the safety problem even worse.

## 1.2 Safety Assurance of Cyber-Physical Systems

Safety engineering is a discipline that assures that the engineered systems are acceptably safe to operate in their intended environments. While there are several definitions of safety, Dezfuli, Homayoon, *et al.* [30] define safety as "freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment and property, or damage to the environment". CPSs that operate in regulated safety-critical fields must undergo a stringent certification process involving several audits to ensure the system complies with the industry safety standards. Several design-time safety assurance approaches (also called static approaches) have been proposed over the years to assure safety and ensure the systems abide by the set standards. These approaches have been used to engineer safe systems or augment safety into pre-developed systems. They involve the application of special engineering skills, analysis tools, and techniques for defining the system's safety requirements, identifying the system hazards, performing a risk assessment of the identified hazards, and managing them by designing control procedures to mitigate or alleviate the risk to an acceptable level [31].

A timeline of some of the well-known design-time approaches is illustrated in Fig. 1.2 and they include: verification and validation [32, 33], testing [34, 35], risk management [36, 37], and assurance cases [38, 39]. These approaches have become integral to several industry standards by demonstrating their effectiveness in reducing fatal accidents. For example, the assurance case (AC) has been the most widely adopted approach in recent years. An AC is a documented body of evidence that provides a valid argument that a system's

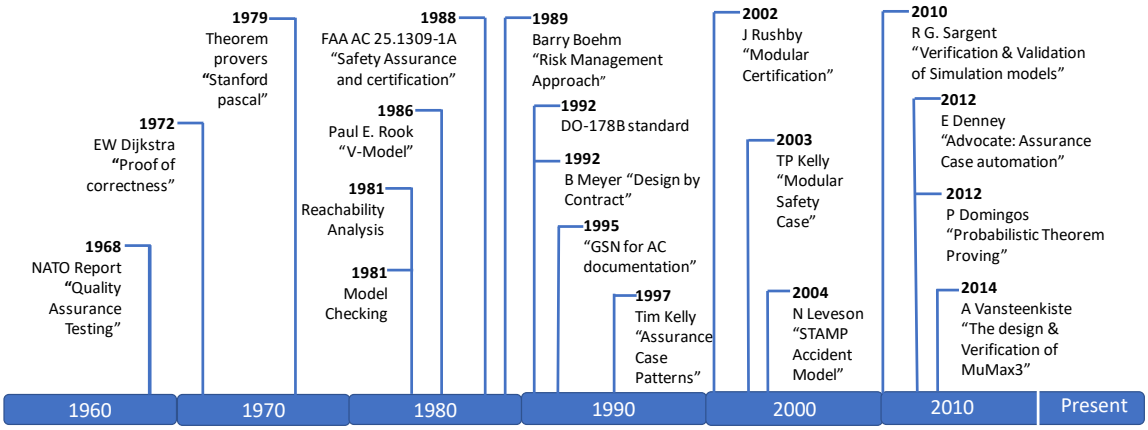



Figure 1.2: A timeline of the key design-time safety assurance approaches.

property is satisfied for a given environment, subject to several assumptions about the system and its operating conditions [38]. Several standards such as the DO-178B [40], ISO 26262 [41], SOTIF [42], and UL4600 [43] have deemed AC as a viable approach for assuring the safety of the system.

However, analysis of several aerospace accidents such as the Ariane 5 launcher [44] in 1996, the Mars climate orbiter in 1999 [45], and the Mars polar lander in 2000 [46] reveal several shortcomings of the design-time approaches [47]. A few relevant shortcomings are: (a) inadequate specification and flawed review process despite applying safety engineering, (b) mismatch in the assumptions about the intended and the actual operating environments, and (c) inappropriate risk assessment, which wrongly assume that the risk decreases over the system's operation, and (d) deficiencies in detection, collection, and use of runtime safety-related information, which was the prime reason for these accidents.

From these problems, an observation can be made that there is a mismatch in our understanding of "how the system is expected to operate" and "how the system actually operates" (**observation 1**). As discussed by Denney *et al.* [48], this misunderstanding creates a gap in the safety argument and the assumptions under which the system was developed at design time, and the actual assumptions at runtime. This gap has been the root cause of several accidents in the past [49, 50, 44]. Also, the fact that the design-time safety arguments are effectively static does not help manage this gap. Therefore, these problems have made the design-time safety assurance approaches insufficient and incapable of capturing the complexity and uncertainty introduced by the LECs [51, 52], and the dynamic operating nature of the modern CPSs [48, 50, 53, 54].

A partial solution to these problems, as suggested by Leveson, is to continuously monitor at runtime all the assumptions that are involved in building the design-time safety arguments [55]. While monitoring the assumptions is a feasible option, it is sometimes hard to identify how and when these assumptions are invalidated (**observation 2**). This is because, in addition to the *explicit assumptions* that are listed in the

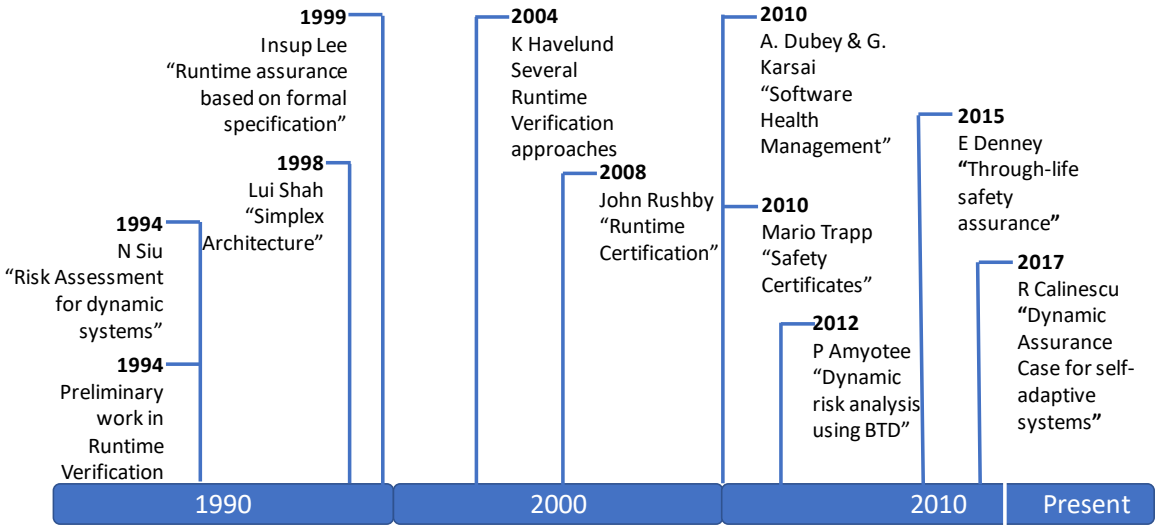



Figure 1.3: A timeline of the key runtime safety assurance approaches.

safety documents (e.g., weather conditions, sensor faults), the safety arguments could often involve several *implicit assumptions*, which are undocumented and hard to detect. For example, inductive biases (e.g., training data, model hyperparameter) are one such assumption involved in the training of the LECs. Such implicit assumptions often go unnoticed at design time and could lead to problems like out-of-distribution (OOD) data problem for LECs [56, 57]. The invalidation of these assumptions can increase the system's risk of entering a catastrophic consequence.

Several runtime assurance approaches have been designed over recent years to address the problems with the design-time approaches. A timeline of some of the well-known runtime approaches is illustrated in Fig. 1.3 and they include: dynamic risk assessment [58, 59, 60], simplex architecture [61, 62, 63], runtime verification [64, 65], safety certificates [66, 67], system health management [68, 69, 2, 70], and dynamic assurance case [48, 71]. These approaches are mostly the runtime extensions of the design-time safety approaches listed previously. For example, the dynamic AC is a design-time AC with empty evidence placeholders that will be completed with monitor results at runtime [53, 72]. Recently, some of these techniques are being applied in a limited capacity for autonomous CPSs [73, 74, 75, 76, 77, 78].

## 1.3 Dynamic Safety Assurance Framework

While the runtime approaches have been shown to overcome some problems of the design-time techniques, they cannot wholly replace the design-time assurance models generated before deployment. This is because the safety standards and the certification process are coupled tightly with the design-time safety approaches, especially the AC (**observation 3**). There is unlikely to be a change in the certification process in the near

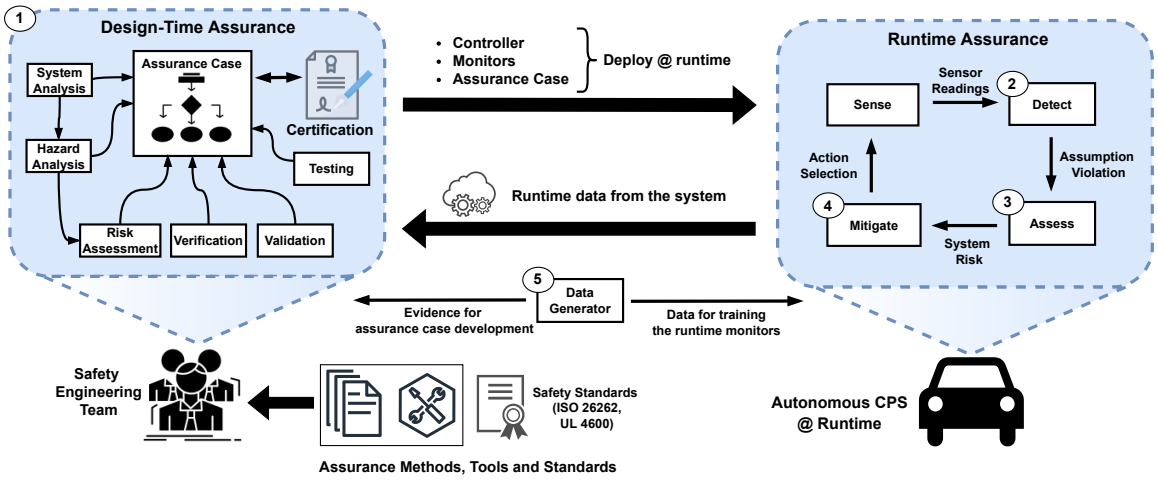



Figure 1.4: Overview of the proposed dynamic safety assurance framework, which is a synergy between the design-time assurance case and runtime monitors that (a) detect violations in the design-time assurance assumptions, (b) assess an increase in the system's risk because of the assumption violation, and (c) mitigate the risk by selecting a suitable control strategy. All these components require abundant data that is generated by the data generator.

future. It is evident from our observations about the existing assurance approaches, the problems (uncertainty and non-determinism) introduced by the LECs, and the changing requirements (operate in dynamic operating conditions) of modern CPSs that either of the approaches is independently insufficient to tackle the safety problem. We hypothesize that the design-time and runtime approaches must be combined to provide continuous safety assurance throughout the system's operational lifetime.

To perform continuous safety assurance, this research proposes the "dynamic safety assurance" approach, which is a synergy between the design-time and runtime assurance approaches. The overall objective is to perform "through life safety assurance" of autonomous CPS, a concept coined by Denney *et al.* [48] for conventional CPS. The idea is, to begin with, the design-time assurance models and include a range of monitors that considers runtime information about the system's operating condition and operational state (including information on system faults) to detect violations in the design-time assurance assumptions. Then, once the violation is detected, dynamically select a suitable action to mitigate the risk introduced to the system because of the assumption violation. Recently, there have been some ongoing research efforts [48, 71] towards achieving this objective of continuous safety assurance of autonomous systems. To join the effort, this dissertation provides a fully-fledged dynamic safety assurance framework for autonomous CPS, shown in Fig. 1.4. This work also discusses the research challenges in implementing the framework components and provides a solution approach for each identified problem.

The framework begins with the *automated development of a static assurance case*, which holds the safety arguments for the system and the assumptions under which they are valid. Once these arguments and

the assumptions are available, the framework has three components to perform the following activities at runtime. *Detect* invalidation of the design-time assumptions and identify the factor(s) responsible for the problem. Especially, this work is interested in detecting the violations of the closed-world assumption in training the LEC, resulting in the OOD data problem. *Assess* the system's operational risk, given the operating conditions, the known sensor and actuator faults, and information from system monitors. *Mitigate* the risk by dynamically selecting a suitable control action for the system. Finally, there is a *data generator*, which provides an automated data generation mechanism required for designing and training the other components of the framework.

However, designing these components is non-trivial because of two reasons. First, the increasing complexity of these autonomous CPSs has made the assurance process complex, labor-intensive, and time-consuming because of the activities that involve managing numerous requirements, curating a large number of artifacts and evidence, and developing and managing huge ACs, among others [79]. Second, the dynamic assurance components are targeted to operate on small-scale CPS testbeds like the DeepNNCar [80], F1-10 car [81] and BlueROV2 underwater vehicle [82] that have short inference times (50 - 100 milliseconds) and limited onboard computational resources (e.g., Raspberry Pi, NVIDIA Jetson TX2). Thus, the efficient design of these components is a critical requirement.

## 1.4 Motivating Research Questions

In developing the proposed dynamic safety assurance framework, a set of five research questions are formulated, each associated with one of the five components in the framework. These questions, while being abstract, describe the key problem and the solution approach devised in this work to overcome the problem.

**RQ1: In safety-critical CPS that requires certification, how to automate the AC development process to reduce the certification cost and time?** Safety certification is essential in several regulated safety-critical CPS domains such as automotive [83], aviation [84], and medical devices [85]. ACs are increasingly being integrated into safety standards [40, 41], establishing the minimum requirements for certifying the system. However, the ever-increasing complexity of these systems has made the certification process complex, labor-intensive, and time-consuming because of the activities involving managing numerous requirements, curating a considerable number of artifacts and evidence, and developing and managing huge ACs. For example, according to the Federal Aviation Administration (FAA), the certification of a new commercial aircraft can take between 5 and 9 years [86] costing upwards of 100 million dollars.

Today, these ACs are manually developed by human experts from different domains, making the develop-

ment process expensive and time-consuming. Automating the development process is essential to avoid human errors in preparing the AC reports and reducing the certification cost and time. Addressing this problem is beneficial for designing the dynamic assurance component - 1 of the proposed framework (see Fig. 1.4).

Automating the overall assurance process is not feasible because it has always been a human-centered activity with expert interpretation required for societally sensitive matters, e.g., harm and risk. However, other tasks like generating system design artifacts, component evidence, and developing graphical ACs can be automated and integrated into the design process. Currently, dedicated programs such as the DARPA Automated Rapid Certification Of Software (ARCOS) [87], and European AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) consortium [88] are looking into developing tools and techniques to automate the certification process. To join the effort, the proposed framework develops a workflow that allows for semi-automated development and evaluation of an AC.

**RQ2: How to efficiently perform OOD detection and factor identification on multi-labeled high-dimensional sensor data (e.g., images) at runtime?** As discussed earlier, the OOD data problem is a principal example of a violation of the closed-world implicit assumption made at design time in training the LEC. To better explain this problem, consider the two phases of the LEC operations. A *training phase* in which the components are trained using multi-modal sensor datasets, and an *operational phase* in which the trained LEC predicts the control actions needed for the system. The fundamental problem is that the LEC training is performed under the closed-world assumption that the operational data are independent and identically distributed (IID) from the same distribution as the training dataset [89]. However, when deployed into the open world, the operational data can be OOD to the training dataset [90]. Research findings [56, 26, 27] have shown these components to perform erroneously when tested with OOD data. There has been abundant research [91, 92, 93, 94, 95, 96] in identifying OOD data. However, there are two challenges in directly utilizing these approaches. First, the autonomous CPS targeted in this work use high-dimensional sensor data such as images, which requires additional resources and time for detection. These requirements are often unavailable on the low-resource platforms (e.g., DeepNNCar [80]) being targeted in this work. Second, the OOD data can manifest in several ways depending on the changes in the data factors (e.g., image features). In addition to detection, finding the root cause(s) factors causing the problem is also critical. For example, one or more image features such as the time of day, precipitation levels, and traffic density may change simultaneously, making an image OOD. Identifying the changing factors is beneficial for selecting a suitable mitigation strategy in our approach. Addressing these problems is necessary for designing the dynamic assurance component 2 of the proposed framework (see Fig. 1.4).

To efficiently perform detection on high-dimensional data such as images, this dissertation presents a work-

flow that designs and trains a detector to generate a low-dimensional representation of the high-dimensional data on which both OOD detection and factor identification are performed. Detection of lower-dimensional representation requires lesser computational resources and a shorter time.

**RQ3: How to quantitatively compute the system's operational risk at runtime by combining the risk calculated at design time and information from runtime detectors?** Violations in the design-time assumptions could potentially increase the system's risk at runtime, possibly leading to a catastrophic consequence. System risk management [31] and Safety risk management [1] have been widely adopted risk assessment and management strategies in the aviation industry. These strategies involve two steps:(1) identifying the system's threats and hazards and (2) calculating a risk score for each threat based on the prior incidents (accidents) involving the threats. The risk is often computed qualitatively using a risk matrix [37] or safety models such as Bow-Tie Diagram (BTD) [97]. The risk calculated from these approaches is then compared against a pre-selected threshold to validate the assurance argument at runtime.

These approaches are widely adopted in domains such as aviation (FAA widely adopts the risk matrix for risk calculations). However, the risk computed from these approaches is primarily gathered through a post-consequence analysis activity that involves analyzing a system accident to identify the cause and initiate corrective actions to prevent future errors. From a dynamic assurance standpoint, such design-time risk information is inefficient at runtime because the evolving system operations and operating conditions could potentially introduce new hazards or failures to the system, which cannot be accounted for by the design-time calculations. Handling this requires a "proactive assessment" strategy that incorporates runtime information. Addressing this requirement is necessary for designing the dynamic assurance component 3 of the proposed framework (see Fig. 1.4).

This dissertation presents a framework called ReSonAte (Runtime Safety Evaluation in Autonomous Systems) to perform a proactive risk assessment of the system at runtime. The framework combines the prior probabilities of the system's hazards (curated at design time) with the information from runtime detectors to compute the system's operational risk. The BTD model is used to combine the prior hazard probabilities with the runtime information from different detectors to update its likelihood of occurrence during the system's operation.

**RQ4: How to dynamically select a control action that can mitigate the risk posed to the system?** One commonly used approach for providing safety assurance to CPS is the use of controller-redundant architectures such as the simplex architecture [61] and controller sandboxing [98]. Indeed, such techniques have been successfully used in automatic landing simulation of an F-16 aircraft [99], unmanned aerial vehicles [100],

remote-controlled cars [101], and industrial infrastructures [102]. Verification-based variants like linear matrix inequality [63], reachability analysis [98], and safety certificates [103] are also widely used for decision making. These architectures augment a safety controller and a decision logic to a CPS with a high-performing but unverifiable controller (also called the performant controller). The logic is usually trained with a single objective of safety. Under unsafe operating conditions or system faults, the decision logic switches from the performant to the safety controller to maintain safety.

While these approaches have shown promising results, there are two major limitations. First, decision logic is typically trained offline. While offline training provides the advantage of invoking the policy almost instantaneously when making decisions, such policies can often become stale in non-stationary and dynamic conditions [104, 105, 106]. Second, the decision logic is usually designed to only perform a one-way switch, i.e., when the system under consideration detects an imminent threat to safety, the logic dictates a switch from the performant controller to the safety controller. Once the logic switches to the secondary controller, the control remains with it forever (barring some exceptions that perform the reverse switch based on system stability [107, 108, 109]). However, once the threat no longer exists, using the safety controller could delay or ignore the system's mission-critical objectives [110]. Addressing these limitations is necessary for designing the dynamic assurance component 4 of the proposed framework (see Fig. 1.4).

To address these limitations, this dissertation presents two extensions to the conventional simplex architecture: the "weighted simplex strategy" and the "dynamic simplex strategy". These extensions employ an adaptive and online decision logic that focuses on the system's safety and performance objectives. The decision-making problem is formulated as a Markov Decision Process (MDP). The first approach performs a weighted blending of the two controllers with the weights being learned using RL and a two-objective reward function focusing on safety and performance. The second approach performs a two-way switching among the controllers. The switching decision is performed by an online Monte Carlo tree search (MCTS) heuristic focusing on balancing both system objectives.

**RQ5: How to automatically generate data required for designing the dynamic assurance components?** Each component of the proposed framework requires a lot of data from different operating modes and operating conditions of the system. For example, calculating the system's operational risk of colliding with a pedestrian crossing the road (threat) will require data from (1) the system operating under different weather conditions (e.g., no rain, low rain, high rain) and (2) the system's operating modes including sensor fault conditions. The existing approaches generate the data manually or use samplers like random or grid searches to sample from a distribution of expected weather conditions and modes in which the system will operate. These approaches can slow the data generation process and may also miss sampling high-risk scenes

required for the probability computation. Addressing this demand for new samplers is necessary for designing the dynamic assurance component 5 of the proposed framework (see Fig. 1.4).

This demand requires an automated data generation mechanism with efficient and intelligent samplers. In this regard, this dissertation presents a framework called ANTI-CARLA, built on the popular open-source autonomous driving simulator CARLA [111]. The framework provides a set of conventional samplers (random and grid search) and adversarial samplers to automatically generate data from nominal (scenes within the training boundary) and high-risk scenes (scenes with adverse weather conditions or system sensor faults).

## 1.5 Scope of Dissertation

The remainder of this dissertation describes the work performed to answer the questions posed in developing the dynamic assurance components. The key publications addressing these questions are shown in Table 1.1.

Chapter 4 addresses the research question **RQ1** by developing a workflow for automating the AC development. The workflow uses modular assurance arguments called "patterns" with the system artifacts (e.g., hazards, risk, system models) to automatically synthesize a comprehensive AC for the system. The generated AC is then automatically evaluated using two coverage metrics, and a report is generated with additional insights into the generated AC to aid the evaluation process. Chapter 5 addresses the research question **RQ2** by developing an automated workflow that performs OOD detection and feature identification of images on a lower-dimensional latent representation. This representation is generated using a variant of the variational autoencoder called the $\beta$-variational autoencoder. The concept of disentanglement is used in generating a representation in which each latent distribution is sensitive to changes in only one input image feature while being invariant to changes in the others [112]. This capability of the disentangled latent space allows us to perform efficient detection using significantly fewer latent distributions (8) compared to the reconstruction-based approaches (currently, the state-of-the-art approaches in OOD detection) that requires a higher number of latent distributions (1024). Also, a mapping between input image features and the latent distributions is learned by performing a sensitivity analysis on the disentangled latent space to identify the feature(s) responsible for the OOD problem.

Chapter 6 addresses the research question **RQ3** by developing a dynamic risk assessment framework called Runtime Safety Evaluation in Autonomous Systems (ReSonAte). The framework uses the BTD model, which allows us to combine the prior probabilities of the system's threat calculated at design time with runtime information from system monitors to dynamically calculate the system's operational risk at runtime.

Chapter 7 and Chapter 8 addresses the research question **RQ4** by providing two extensions to the conven-

| Research Question | Relevant Publications |
|---|---|
| RQ1 | [113, 114, 93] |
| RQ2 | [115, 116] |
| RQ3 | [117] |
| RQ4 | [118, 119] |
| RQ5 | [80, 120] |

Table 1.1: Summary of key publications addressing the research questions posed in this dissertation

tional simplex architecture. The first extended simplex architecture discussed in Chapter 7 is the weighted simplex strategy, which performs a weighted blending of the two controller actions. Weights are learned using RL and a two-objective reward function focusing on safety and performance. The second extended architecture discussed in Chapter 8 is the dynamic simplex strategy, which uses a decision logic designed with the online MCTS heuristic to select an optimal controller that maintains the system's performance and safety objectives.

Finally, designing and training each dynamic assurance component of the proposed framework requires a lot of data. For example, calculating the prior probability of the system's threat for risk assessment requires a wide selection of system data from nominal and high-risk scenes (e.g., adverse weather conditions, sensor faults). For this, a data generation framework called ANTI-CARLA is presented in Chapter 9 to answer the research question **RQ5**. The framework uses a scenario description language with adversarial samplers to generate data from nominal and high-risk scenes (e.g., scenes with high precipitation, and sensor failure).

**Outline**: The remainder of this dissertation is organized as follows: Chapter 2 examines the related work in the areas of design-time and runtime assurance. Chapter 3 presents the two demonstration platforms used to demonstrate the proposed framework. Chapter 4 describes a workflow for automating the development and evaluation of an AC. Chapter 5 presents an OOD detection and factor identification approach for CPSs with perception LECs. Chapter 6 presents the ReSonAte framework for operational risk assessment. Chapter 7 presents the weighted simplex strategy with its RL-based decision logic for blending the control actions to maintain the system's objectives. Chapter 8 presents another extension to the conventional simplex architecture called the dynamic simplex strategy, which uses an online MCTS-based decision logic for selecting an optimal controller to maintain the system's objectives. Chapter 9 presents the ANTI-CARLA framework for automatically generating the data required for the dynamic assurance components. Finally, Chapter 10 concludes the dissertation with a look back at the accomplishments and proposes the possible directions in which this work can be extended in the future.

# Chapter 2

# Background and Related Work

## 2.1 Design-Time Safety Assurance Techniques

Design-time safety engineering and risk management activities have been used to engineer safe systems or augment safety into pre-developed systems. These activities involve the application of special engineering skills, analysis tools, and techniques for defining the system's safety requirements, identifying the system hazards, performing a risk assessment of the identified hazards, and managing them by designing controls procedures to mitigate or alleviate the risk to an acceptable level [31].

As discussed, verification and validation [32, 33], testing [34, 35], risk management [36, 37], and assurance based development [38, 39] are some of the existing design-time techniques used for assuring the safety of conventional CPSs. Recently, some of these techniques have been applied to assure the safety of autonomous CPS. Especially, there has been considerable progress in the areas of verification [28, 29], and testing [26, 27]. However, the wide-scale applicability of these techniques is limited because of the increasing complexity and non-linearity of the LECs. These approaches will be reviewed in the section below.

### 2.1.1 Verification and Validation

Verification and Validation (V & V) are two independent procedures often used together to check if a system is performing as intended. Verification involves building an abstract model of the system and using them to establish key properties of the system [121]. The existing verification techniques can be classified into model checking and theorem proving. Model-checking builds a finite model of the system and checks if the specified properties are satisfied by the model [33]. If a specified property is not met, several approaches yield counterexamples, which can help falsify the model [122]. This approach exhaustively explores the system's state space, which is possible for small systems; however, the exhaustive search is impossible for complex systems, which typically have a large state space. *State space explosion* is a big problem that limits the scalability of this approach [123]. However, there has been ongoing work toward alleviating this problem [124, 125].

Model-checking typically performs reachability analysis, a process of computing a set of reachable states that the system can reach over a finite time horizon [126]. The reachable set is a common term used with reachability analysis, which describes the system's trajectories for all possible combinations of the initial

conditions and the model's hyperparameters. Deriving the reachable set can be useful in finding if the system will reach an unsafe state or not. There have been several reachability algorithms varying the shape of the reachable set. Some of the reachable sets used are polyhedra [127], start-sets [128], zonotopes [129], and barrier certificates [103]. While reachability analysis has been widely adopted for several applications [73, 74, 75], computational cost, execution time, memory, and accuracy (over-approximation) are some of the known challenges in using the approach.

As listed, theorem proving is the other well-known verification technique that provides mathematical reasoning for the correctness of the system's property of interest [33]. In this approach, the model (abstract model of the system) is expressed as formulas using logic, which defines a set of rules and axioms. These rules and axioms are used to prove the system's property of interest. While model checking suffers from state explosion problems, theorem proving can handle infinite state spaces using techniques such as induction. There have been several theorem provers for formal analysis of CPSs [130, 131].

Model Validation is often used in parallel to the verification approaches discussed above. Validation is a means to "substantiate that a model within its domain of applicability possesses an acceptable range of accuracy consistent within its intended application" [132]. While verification refers to tools and techniques used to assure that the abstract model is correct, validation refers to tools, techniques, and tests used to ensure that the model under test conforms to the provided requirements and specifications. We refer the readers to the following surveys for in-depth information regarding different validation techniques in CPS [133, 134].

Recently, there have been significant ongoing research efforts to extend these techniques to autonomous CPS. However, the presence of activation functions in the LEC's model (e.g., neural network) introduces a non-linearity, which makes the problem non-convex [28]. The non-linear and non-convex nature of the LEC makes it hard to verify even simple properties, making it an NP-complete problem. Despite the problem, several testing and verification techniques have been available for LECs. For example, robustness testing is performed by perturbing the inputs of the LECs with adversarial data [135, 136]. Test coverage is another widely adopted testing approach [26, 27] in which the neural network is tested against an extensive collection of test data to evaluate whether the predicted outputs correspond to expected values [137]. However, this general testing approach fails to uncover all the possible corner cases of the network. Therefore, a neural network-specific testing approach called neuron coverage has been recently proposed [26].

As discussed by Liu *et al.* [137], the verification techniques for verifying neural networks can be broadly classified into three main categories. (1) reachability-based techniques, which perform a layer-wise reachability analysis of the network. Some representative tools are NNV [138], ERAN [139], and SymBox [140], (2) optimization-based techniques, which use optimization to falsify the network. Some examples include ILP [141], Duality [142], and ConvDual [143], and (3) search-based techniques, which involve searching for

test cases that can falsify the network. Some tools include Reluplex [28], ReluVal [144], and VeriNet [145]. Despite considerable progress, there are several challenges in using these tools and techniques. First, the primary problem in verification is that it is hard to build abstract models that accurately match the actual system. Second, in the case of neural networks, it is difficult to specify the properties that need to be verified. Third, the existing tools do not scale to large neural networks with many neurons. In addition, they are restricted to piecewise linear activation functions like ReLU [28]. Last, verifying neural networks in closed-loop CPS applications is a complex problem that still needs addressing.

### 2.1.2 Simulation and Testing

Testing ensures identifying for any difference in the intended and actual behaviors of the system under test. Since real-world testing is infeasible for several applications, simulation has become integral to several testing techniques. As discussed by zhou *et al.* [146], the existing testing techniques can be classified into one of the following categories: (1) *Model-based testing* [147], involves systematically generating test cases for testing abstract models used to represent the system under test. (2) *Search-based testing* [148] involves searching for test cases that fail the system. For this, samplers such as random and grid search are popularly used. (3) *Fault injection-based testing* [149], involves artificially injecting failures into the system to test its fault tolerance capabilities. (4) *Conformance-based testing* [150] involves testing the system to check if it has sufficient conformance with the requirements and certification standards. (5) *Robustness testing* [122], involves testing the robustness of the system by introducing faults and anomalies.

Recently, there has been significant research in the automotive domain for testing autonomous CPS. Search-based testing with simulators such as CARLA [111] and Gazebo [151] are extensively being used in generating safety-critical or high-risk scenes (or test cases) for falsifying the system under test. A scene in this domain is a short time series of similar sensor data (e.g., images) contextualized by certain environmental features such as weather, brightness, road conditions, and traffic density. Several scenario description languages (SDLs), such as Scenic [152] and measurable scenario description language (MSDL) [153] are used with samplers to generate these test case scenes. The role of the sampler is to sample a value for the different scene parameters (e.g., weather, traffic density, pedestrian density) from their respective value ranges. This is a problem of coverage, which involves searching for safety-critical scenes across the search space of the environmental variables. However, sampling across all the possible combinations of the scene parameters is expensive, time-consuming, and intractable. Several active sampling approaches have been proposed recently to address this problem of passive samplers [148, 154, 155]. These samplers use the system's previous performances as feedback to guide the search process towards new safety-critical scenes.

### 2.1.3 Risk Assessment

One way of demonstrating that the system is safe is to prove that the risk posed to the system is mitigated or sufficiently alleviated to acceptable levels that it can tolerate. Achieving perfect safety is an impossible task. So, this approach often asks two questions: "how safe is safe enough?" and "What is an acceptable risk level?". To answer this, the concept of As Low As Reasonably Practicable (ALARP) [36] and As Safe As Reasonably Practicable (ASARP) [30] were introduced. Both these approaches rely on the concept of reasonable practicability. These are not hard thresholds, and they differ between systems based on the criticality of their operations.

The basic terms involved with the risk management approach are hazards, consequences, barriers (or controls), and risk. These terms are briefly defined below.

- **Hazard** is an event or condition that could lead to an undesirable situation for the system. For example, a pedestrian crossing the road can be a hazard for automotive CPS.
- **Consequence** is an undesirable and unexpected situation for the system. For example, accidents (or collisions) are a common consequence that an automotive CPS can encounter during its operation.
- **Barrier or Controls** are components required to prevent the hazards from leading the system to a consequence. For example, emergency braking can be an example barrier for automotive CPS.
- **Risk** is a combination of the probability (or frequency) of hazard occurrence and the severity (or consequence) of the hazard as computed in Eq. (2.1).

$$Risk = frequency(hazardevents/time) \cdot severity(consequence/time) \tag{2.1}$$

The steps involved in the risk management approach as shown in Fig. 2.1 are. (1) *System Analysis*, which involves designing the system architecture and defining the system objectives (e.g., safety, functional). Next, techniques such as failure mode and effect analysis (FMEA) [156] and fault tree analysis (FTA) [157] are used to identify how the system can fail. (2) *Hazard Analysis*, which involves identifying the potential hazards to the system like operational, functional, software, and hardware hazards. This analysis also includes identifying the consequence(s) resulting from these hazards. (3) *Risk Assessment*, which involves two steps. The first step involves analyzing and quantifying the risk using Eq. (2.1). The analysis includes visualizing the hazard propagation paths using graphical tools like BTD. Next, the system risk is computed qualitatively or quantitatively using tools like risk matrix [37] and BTD [59]. The second step involves consolidating and prioritizing the risk scores. (4) *Designing risk controls*, which involves designing barriers to alleviate or mitigate the risk. These steps are used in the safety risk management workflow of FAA [1]. The risk assessed at design-time is then used in the safety assurance workflow during operation, as shown in Fig. 2.1.

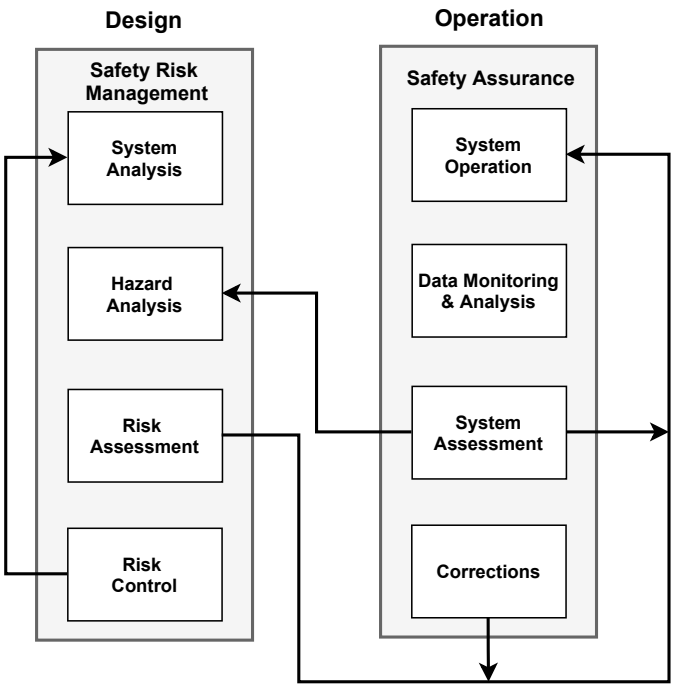



Figure 2.1: The FAA framework for system safety management adopted from [1]. It includes a design-time risk assessment procedure and an operational safety assurance procedure.

Let us briefly discuss the BTD before discussing the existing approaches. A BTD is a graphical structure often designed to analyze hazard propagation and identify possible control strategies to prevent the hazard from failing the system. The BTD can be represented as a chain of events $t_i \rightarrow b_p \rightarrow e_{top} \rightarrow b_m \rightarrow c_i$. Where $t_i$ are the threats, $b_p$ is a barrier (control) to prevent the threat propagation, $e_{top}$ is a top event, which is an escalation of the threat, $b_m$ is a barrier to prevent the propagation of the top event, and $c_i$ is a system consequence (e.g., collision). To explain these events, consider the BTD for an operational hazard of "roadway obstruction" for an AV case study (See Fig. 2.2). The BTD has two threat events of T1 and T2, corresponding to other vehicles and pedestrians in the system's travel paths. These threats could escalate and become a top event (TOP) if not prevented by the preventive barriers ($B1$ and $B2$). An adequately trained LEC is used as the preventive barrier for this example. Next, if the top event occurred, it could become a consequence if not mitigated by a mitigation barrier ($B3$). The performance of the automatic emergency braking system (AEBS) in quickly stopping the vehicle is used as the mitigation barrier for this example.

BTDs have been used as safety models in the unmanned aerial systems domain, and they are now being adopted for certification of these systems. Recently, combining assurance arguments with qualitative risk computed using BTDs has been viewed as a robust approach for safety assurance [158, 159, 160, 161]. Quantitative risk assessment has not been very popular with these design-time approaches because they need comprehensive data, which is hard to get.

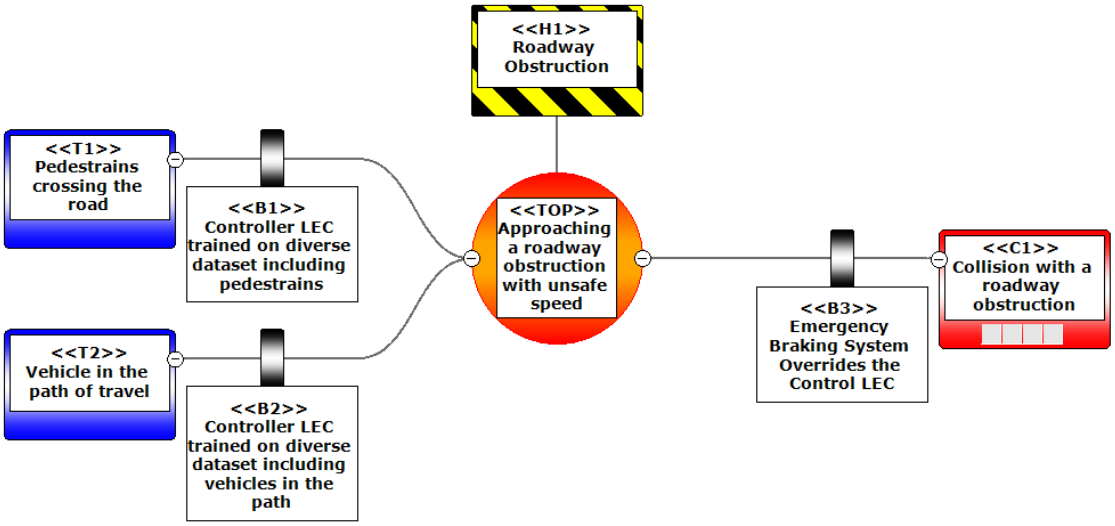



Figure 2.2: Example Bow-Tie Diagram structure for an AV case study.

### 2.1.4 Accident Models

Causal accident models are used for two reasons: (1) to identify the party responsible for the accident and (2) to analyze the cause of the accident so as to design mitigation strategies that can prevent similar accidents in the future. A common way of designing these models has been to isolate one or more events at the beginning of an event chain and hold them responsible for the accident. The intermediate events in the chain are considered to be the contributing factors [162]. The BTD model discussed in the previous section is one example of an accident model. It captures the causal relationship between the threats (events) and the consequence (effect). Though these models have worked well, the growing complexity of systems and increased component interactions have changed the nature of the accidents [2] (e.g., component interactions have become a new source of accidents), which cannot be captured by the causal accident models.

To incorporate the changing nature of accidents, Rasmussen *et al.* [163] introduced the Accimap model that takes a system-oriented approach rather than the analytic reduction approach of the traditional causal accident models. Based on this idea, Leveson introduced a new accident model called STAMP (Systems-Theoretic Accident Model and Processes) [162, 164], which allows modeling relationships between events causing the accident. STAMP has been used to analyze several accidents, including a plane crash [165] and a water contamination problem [162]. STAMP has also been applied for hazard analysis and risk assessment [162]. Extensions to STAMP called system-theoretic process analysis (STPA) [166] and causal analysis model based on STAMP (CAST) [167] have been applied for causal analysis of accidents. Despite being used in several analyses, several limitations of this approach are [168]: (1) the model can only perform qualitative analysis,

(2) designing the model requires extensive theoretical and domain knowledge, and (3) designing the model is complex and time-consuming.

### 2.1.5 Assurance Case

The safety certification process aims to prove that the system will function safely in the presence of hazards, failures, and environmental uncertainties. As discussed by Hawkins, Richard, *et al.* [169], the certification process requires an explicit assurance argument that communicates how the evidence (e.g., simulation and testing results, expert opinion) generated during system development satisfies the claims about the system's properties (e.g., safety, security, and reliability). There have been various models such as Toulim model [170], Walton model [171], and Assurance Case (AC) [38] for building assurance arguments. However, ACs have been widely adopted among them and are increasingly being used across safety standards (e.g., ISO 26262) and the certification process.

An AC is a documented body of evidence that provides a valid argument that a system's property is satisfied for a given environment, subject to several assumptions about the system and its operating conditions [38]. A safety case is a version of an AC that focuses primarily on the system's safety property. ACs were initially documented as textual documents with free text and tables that spanned several thousand pages. However, ambiguities and missing clarity in natural language used for communicating the safety cases resulted in several accidents in the past like the clapham rail disaster [172] and the piper alpha off-shore oil and gas platform disaster [173]. In addition to the language ambiguities, the assurance process followed in these accidents was highly prescriptive. The process just followed the rigid rules set by a safety standard that was generic across several industries. As a result of these accidents, the last two decades have seen progress and changes in the assurance process. Two notable changes that have benefited the process are: (1) the migration to a goal-oriented paradigm from the prescriptive-based approach and (2) the use of graphical structures like Bayesian belief networks (BBN), claims argument evidence (CAE) [174] and Goal Structuring Notation (GSN) [175] to document an AC.

GSN represents the AC elements as nodes of a graph and the relationships between elements as the graph edges. The principal elements of GSN are (1) *Goal* is a requirement or property of the system that needs to be met. (2) *Strategy* is a method used to prove the system's property. A single strategy is often insufficient, so multiple strategies are used to prove a system's property. (3) *Context* provides additional information for other elements. (4) *Assumptions* are statements taken to be true without proof or evidence. (5) *Justification* explains why a specific strategy is used in satisfying the system's property. (6) *Solution* is a system-based artifact used to support a goal.

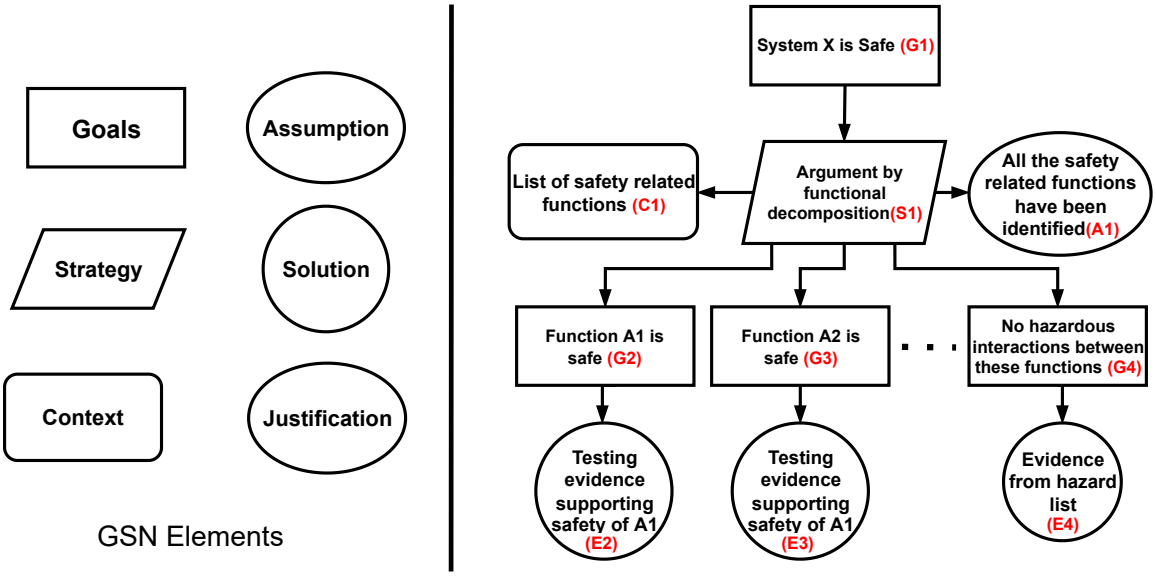



Figure 2.3: (left) modeling elements of GSN. (right) GSN fragment arguing safety of an autonomous vehicle.

Fig. 2.3 illustrates the different shapes used to represent these elements and an example GSN fragment. The root node of the graph is typically the top-level system goal or requirement that does not have direct supporting evidence. So, the GSN performs an iterative decomposition of the top-level goal to sub-goals using a strategy. The sub-goals often refer to component-level properties, for which evidence is mostly available. This decomposition is performed until evidence to support the sub-goal is found. In this example, the top-level goal of "autonomous vehicle will safely navigate without collision with an object in front" is broken down using a system functional decomposition strategy (left branch) and a hazard mitigation strategy (right branch).

**AC Patterns**: While the details relevant to an AC may differ between systems, the structure of the arguments may follow a typical pattern. Reusing such common patterns can aid the safety documentation process by avoiding duplication. To reuse and systematically design arguments for complex systems, Kelly introduced the concept of AC patterns [176]. He also made several extensions to include these patterns in the GSN modeling language. These extensions include a set of structural and entity abstractions for simplifying the assembly of these patterns into an AC. The structural abstractions allow the generalization of the object (goal) relationships. The structural abstractions include multiplicity extensions to generalize n-ary relations between the GSN nodes and optionality extensions to represent optional and alternative relations between the nodes. In addition, for generalizability, the modeling language includes placeholders that can be instantiated with system-specific information. These extensions to the GSN language allow several such patterns with placeholders to be instantiated and assembled into an AC. Other than reusability, this approach also simplifies designing arguments for complex systems.

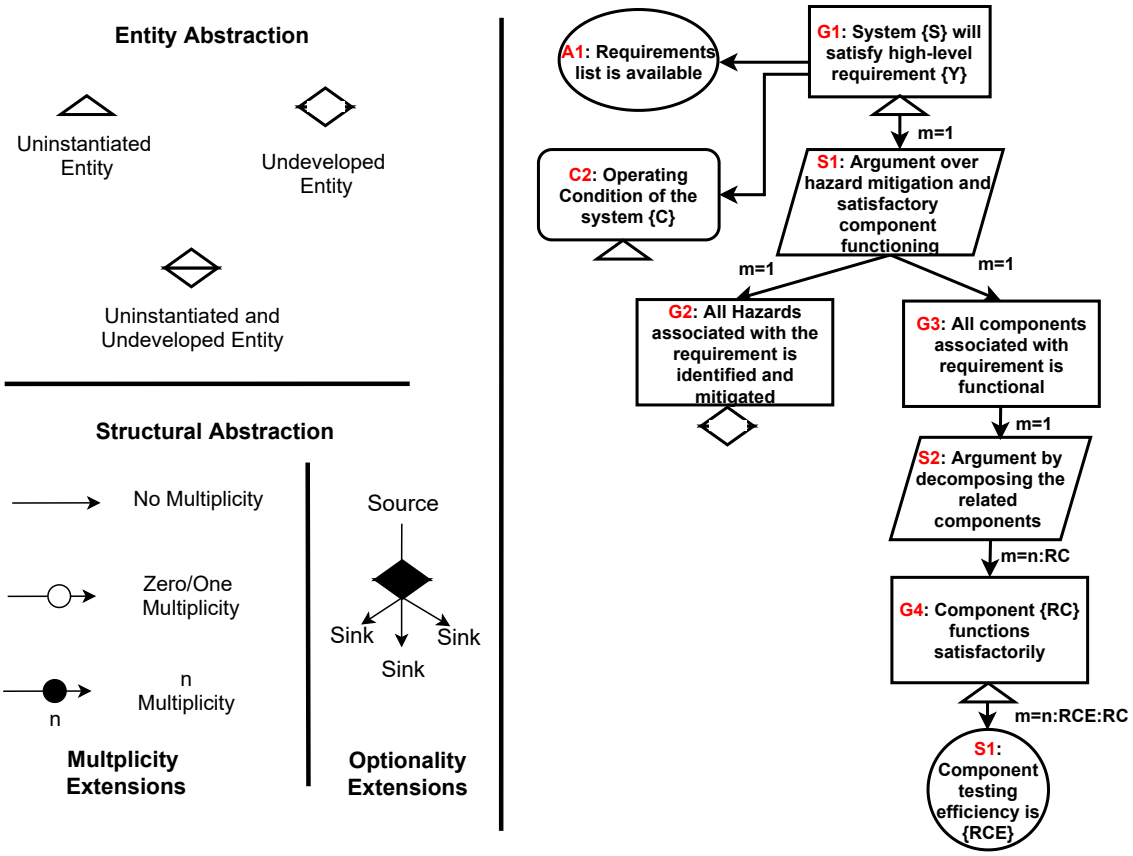



Figure 2.4: (left) GSN pattern abstractions. (right) Example GSN pattern.

Fig. 2.4 illustrates an example requirements decomposition pattern. A node in this pattern is represented using an identifier (e.g., G1, G2) and content with placeholders (e.g., S, C) that can be replaced with system-specific information. The node multiplicity ($m$) is marked on the graph edges, representing how one node is related to another. For example, $m = n : RC$ represents a one-to-many relationship, where one node of S2 is related to $n$ variants of G4, and $n$ is an integer value governed by placeholder $RC$. The nodes will have a one-to-one relationship if not explicitly mentioned. Further, "instantiate" and "develop" are the two modifier operations that can be performed on the nodes. "Instantiate" involves replacing the typed parameters in the placeholders of a node with system-specific artifacts like goals, requirements, and hazards. "Develop" means the pattern can be further grown by connecting with other patterns.

AC patterns have gained interest because of their modular and reusable properties. Recently, there have been several online pattern catalogs that are readily available for constructing an AC. For example, a catalog of hazards and risk reduction patterns adhering to the GSN standards can be found in [176]. [177] provides a catalog of safety arguments for software systems. Szczygielska *et al.* [178] also provide an online catalog with 45 reusable patterns documented in the NOR-STA [179] modeling language.

**Assurance Case construction and automation**: The existing AC construction processes can be broadly classified into one of these approaches [180]: (1) *Prescriptive* process in which safety standards heavily guide the AC development. Despite being widely used in several applications like nuclear plants and oil pipelines, the prescriptive nature of the process has resulted in several well-known accidents like the clapham rail disaster [172], and the piper alpha off-shore oil and the gas platform disaster [173]. (2) *Goal-Oriented* process in which the high-level safety goals of the system are prescribed, but the process itself is non-prescriptive. The last two decades have seen the certification process migrate from a prescriptive to a goal-oriented paradigm [180, 48]. (3) *Blended* process in which both prescriptive and goal-oriented processes are blended into the development of the AC.

Leveson [55] points to several common pitfalls in the existing AC construction process. *First*, AC construction has become a discrete activity performed at the end of the system's development. *Second*, the evidence used to support the AC is always one-sided. The generated evidence is always biased to support the arguments, and there is less prominence given to *counterexamples*. *Third*, safety analysis is often limited to what is expected and not what could be catastrophic. This bias in the analysis process has resulted in the well-known clapham rail disaster [172] and piper alpha off-shore oil and gas platform disaster [173]. *Fourth*, the development process is considered a one-time activity often forgotten after the system certification. *Fifth*, qualitative information is often used to support the goals and arguments. However, qualitative data can be subjective, making it insufficient for safety reasoning.

To summarize, the AC construction process is complicated, making the certification process expensive and time-consuming. Two notable factors responsible for this problem are: (1) the construction process relies highly on human developers, and (2) the growing complexity and the dynamic operating nature of CPSs.

There are several commercial and academic tools [181, 182, 183, 184] with visual editors for rapid construction and management of ACs. For example, Assurance Case Construction and Evaluation Support System (ACCESS) [181] is a commercial tool that provides an editor for prototyping and visualizing GSN-based ACs. Certware [184] is an eclipse-based editor with advanced features like AC templates and provenance. Dependability Case editor (D-Case) [183] is an open-source eclipse editor that allows for designing patterns and assembling them into an AC. Other than these tools, dedicated programs such as the DARPA Automated Rapid Certification Of Software (ARCOS) are looking into building tools and techniques to automate several activities of the AC construction process.

Today, some tools exist which can support automation to a certain extent. For example, the Resolute [185] tool aids in automatically synthesizing an AC from the AADL [186] system architecture models. The AdvoCATE tool [187] allows the design and reuse of patterns to construct a comprehensive case. Denney *et al.* [188] use an automated instantiation and assembly mechanism with the patterns generated from AdvoCATE

to synthesize an AC. Hawkins *et al.* [189] have used the concept of model weaving to automatically learn the artifact files from system models and use them for instantiating patterns.

### 2.1.6 Modular Certificates and Contracts

Certifying safety-critical CPSs (e.g., aircraft) is complicated, expensive, and time-consuming. For example, according to the Federal Aviation Administration (FAA), the certification of a new commercial aircraft can take between 5 and 9 years [86] costing upwards of 100 million dollars. So, instead of certifying the entire system at once, modular certification aims at using modular, compositional, and reusable certification arguments to incrementally certify the system [190, 191]. This involves generating modular safety arguments in isolation for each component of the system and then composing them in stages toward the certification goal. AC patterns [176] and modular safety cases [192, 193] have been the two approaches for designing modular safety arguments. While generating modular arguments is straightforward, composing them is complicated. The composition is typically performed using the concept of contracts [193, 194]. Despite being extensively used in certifying integrated modular avionics (IMA) architectures [190, 191, 195, 196], their applicability is limited because they can only be used for systems with well-partitioned system architecture.

Besides its use in modular certification, contracts have been an essential aspect of designing complex CPSs [197]. Design by contract has been used to design resilient [198] and safe [199, 200] CPSs in several domains, such as automotive [201] and aircraft design [202]. To briefly explain, contracts usually have two parts. The first part holds the assumptions on the behavior of the environment, and the second part holds the guarantees that the component can provide if the assumptions are satisfied [203]. Several contract-based approaches have been designed in recent years, including dependency-guarantee (D-G) contracts [193] and assume-guarantee (A-G) contract [194]. D-G contracts capture the guaranteed properties of a component when its dependency on the related component(s) is satisfied. A-G contracts capture the guaranteed properties of a component when its assumptions on the operating environment are satisfied.

## 2.2 Runtime Safety Assurance Techniques

The design-time safety assurance techniques use the available system artifacts (models and evidence) to design a safety reasoning under several assumptions about the system and its operating conditions. In doing so, a widely held notion is that the system's operating conditions at runtime will remain the same as the design-time conditions for which the system was designed. However, this notion is invalidated by the dynamic operating conditions that introduce new hazard types and failure modes not accounted for during the system's design. Such gaps in the documented safety and the actual safety at runtime have resulted in several accidents, including

the RAF Nimrod aircraft accident [204]. Concluding from several similar research findings [48, 55, 50], it is clear that design-time assurance techniques are alone, not sufficient for assuring the safety of CPSs at runtime. Therefore, to ensure systems at runtime, several runtime assurance approaches have been proposed in recent years. As discussed, dynamic risk assessment [58, 59, 60], simplex architecture [61, 62, 63], runtime verification [64, 65], safety certificates [66, 67], system health management [68, 69, 2, 70], and dynamic AC [48, 71] have been some of the well-known approaches, which will be reviewed in the section below.

### 2.2.1 Dynamic Assurance Case

While ACs only use design-time system artifacts, dynamic ACs combine the design-time artifacts with runtime monitoring to assess the system's safety. For example, ENTRUST [53] and Assure-It [72] provide an end-to-end methodology that models and integrates dynamic AC for self-adaptive CPSs. In these frameworks, an AC with empty evidence placeholders is constructed at design time and completed with monitor results at runtime. The D-Case editor [205] extends the syntax of conventional ACs with new GSN node types to incorporate runtime monitoring information. Denney *et al.* [48] present a concept of through-life safety assurance, which aims at transforming a static AC into a continuous, evolutionary structure using a collection of system monitors and a Bayesian confidence structure for confidence evaluation.

Recently, there has been some interest in using dynamic ACs for autonomous CPS. Asaadi, Erfan, *et al.* [71] present an assurance architecture in which a dynamic AC is the core element that is constantly monitored and evaluated at runtime using an assurance quantification model. Finally, an optimal action is chosen based on the validity of the dynamic AC. In another work [78], the authors have proposed a new assurance measure that is used as runtime evidence to complete the AC at runtime. Despite some progress, there is no clear understanding of how to proactively use the dynamic ACs for selecting a mitigation strategy.

### 2.2.2 Runtime Safety Certification

Runtime safety certification is the runtime extension of the modular certification approach discussed in Section 2.1. In the modular certification approaches [190, 191], the certificates are designed and composed at design time. However, in this approach, the certificates are designed at design time but composed at runtime. For example, conditional safety certificates (ConSerts) [66, 67, 206] involves designing modular safety certificates for the system's components at design time. Then, the assembling and evaluation of the certificates are performed at runtime. Based on ConSerts, the multidirectional modular conditional certificates (M2C2) [207] split a powertrain system into vertical and horizontal interfaces for which the modular certificates are designed and assembled at runtime.

### 2.2.3 Runtime Verification

Runtime Verification monitors the outputs of a model (abstract model of the system) against some specified properties, and it raises an alert when the specified property is violated [208]. It is a lightweight technique used in conjunction with static verification and testing techniques discussed in Section 2.1. This approach can only guarantee if the result computed by the abstract model is correct or not. It cannot provide any guarantees about the correctness of the model itself. Runtime verification is more practical than static verification approaches because it sacrifices for completeness. That is, it does not evaluate all the possible traces [209]. According to Falcone *et al.* [210], the runtime verification approaches typically have the following steps. The first step involves synthesizing a monitor for each specified property of interest. The second step consists of integrating the monitor into the system by assigning the proper input events and defining how the output needs to be analyzed. The last step involves the actual deployment of the monitor into the system during operation; when it receives input events from the system, the output is written to a log and analyzed by a decision-making component. Temporal logic is used primarily for designing the monitors [211]. While linear temporal logic (LTL) [210] is the most used logic, they need an infinite length of input event traces for reasoning, which has resulted in several interval-based logic like metric temporal logic and signal temporal logic.

Rushby [64] was one of the first to propose this concept of runtime verification for CPSs. He presented a concept of just-in-time certification that utilized static verification techniques with runtime verification monitors to verify the component's behavior towards the system's goal(s). As an extension, he also proposed the use of several runtime monitors, including anomaly detectors and runtime verification monitors, for partial certification of CPS at runtime [65]. Runtime verification monitors are used in several CPS applications [212, 208, 213, 214]. Reachability analysis, a static verification technique, has been extended to perform runtime verification. Recently, variants such as Hamilton Jacobi reachability analysis [215] have been used for applications such as safe motion planning [74]. We refer the reader to the following surveys for a more detailed explanation of runtime verification approaches [216, 217, 210, 218].

Runtime Verification techniques have been extended for CPS with LECs. For example, Chen *et al.* [73] used the Hamilton Jacobi reachability analysis framework for safe platooning of unmanned aerial vehicles. Hamilton Jacobi reachability analysis framework has also been used for safe autonomous navigation of a turtle bot in unknown environments [74]. Runtime Verification has been used in a risk analysis framework for the safety analysis of self-reconfigurable autonomous CPS. Several other research findings have used this approach for safe navigation of autonomous CPS [219, 220, 221]. Despite the considerable progress, these approaches have only been applied to less complex CPSs. The major contributing factor to this is the inaccurate and partially available abstract models of the physical systems [217].

### 2.2.4 Dynamic Risk Assessment

The existing approaches for risk assessment can be broadly classified into predetermined risk assessment and dynamic risk assessment approaches. While predetermined risk assessment is meant to be a design-time strategy (discussed in Section 2.1), its counterpart is more suited for runtime assessment [222]. Dynamic risk assessment has been used for safe reconfiguration in open adaptive systems such as automotive [222, 223], avionics [224], and medical-robotics [225]. The existing dynamic approaches can be classified into three approaches. The first approach involves designing a discrete state-space model for the system, identifying the risk associated with each possible state transition action, and acting with the least risk [222]. The second approach involves designing and using different risk metrics at runtime. For example, time-to-collision and distance-to-collision are popular metrics used in the automotive domain. If the time or distance to an object falls below a pre-selected threshold, the situation is considered risky, and a safe action is enforced [226]. However, defining these metrics in some application domains like medical CPS [225] is not straightforward.

Quantitative risk assessment using BTD models [58, 59, 60] is the third approach. Here, the causal structure of the BTD (See Fig. 2.2 in Section 2.1) is utilized in quantitatively computing the risk. This requires computing the conditional probabilities of the barrier events in the BTD from diverse data from the system's different operating conditions and failure modes. These probabilities are combined to compute the likelihood of a system consequence. In turn, the likelihood is combined with the severity of the consequence to compute the system's risk. Besides these approaches, A hierarchical self-management framework for online risk management is developed for a gas turbine aero-engine case study [224]. The different layers of the framework communicate to exchange runtime information gathered from system monitors, which are used to compute the operational risk of the system. In another work [227], the authors have associated a risk level for different configurations of an adaptive system, which is updated at runtime based on the complexity of the system's operating context and environment. [225] integrates runtime safety certificates and dynamic risk assessment to deal with the unpredictability in the cooperative behavior of collaborative medical CPS. This concept is extended using Bayesian networks [228].

### 2.2.5 System Health Management

System health management (SHM) [68, 69, 2, 70] considers that a failure in one component of the system can propagate to the other interacting components, thus failing the system. It involves proactively identifying the root cause(s) of the failure, intervening to prevent cascading failures, and performing a suitable mitigation strategy. These tasks are achieved by integrating a runtime fault management framework into the system for automated detection, diagnosis, and mitigation of adverse system failures [69, 68]. These steps are illustrated

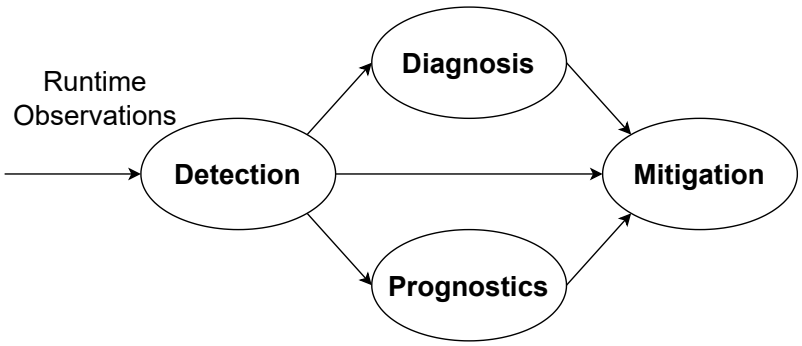



Figure 2.5: The four steps of the system health management technique adopted from [2].

in Fig. 2.5 and are defined as follows.

- **Detection** involves identifying whether the system is operating in the nominal condition or not.
- **Diagnosis** While detection reflects the symptom of failures, diagnosis finds the root cause(s) of the failure. Diagnosing the underlying cause or component is critical for health management.
- **Prognosis** involves predicting the occurrence of a fault in the future. Specifically, it involves using data and models to predict the remaining useful life of the system [2].
- **Mitigation** involves alleviating or mitigating the failure to ensure safe and uninterrupted system operation.

Srivastava *et al.* [70] provides an extensive survey of different SHM schemes along with a brief introduction to the detection, diagnosis, prognosis, and mitigation steps. Mahadevan *et al.* [69] present a two-level SHM framework with a component-level health manager for local fault management and a system-level health manager for system-level fault management. The observations from the local component monitors are sent to a suitable diagnosis engine such as Timed Failure Propagation (TFPG) for system-level fault management. As an extension, Bayesian analysis is used as a probabilistic diagnostic approach to include the uncertainties of the system [229]. These approaches have been successful for runtime fault mitigation of traditional CPSs. However, introducing LEC into the system could introduce imperfections in causality, making diagnosis difficult [230].

# Chapter 3

# Experimental Platforms

The effectiveness of the proposed dynamic safety assurance components will be demonstrated on the demonstration platforms listed below.

## 3.1 DeepNNCar Autonomous Driving Platform

The first platform is a resource-constrained autonomous driving testbed called DeepNNCar [80, 231] (See Fig. 3.1) that was developed in ScopeLab, Vanderbilt University. The goal of the car is to autonomously navigate around an indoor track (safety requirement) with the highest achievable speeds (mission requirement). In this case, the car moving out of track is considered a safety violation. The car is built on the chassis of a 1/10 scale remote-controlled car. It is equipped with two onboard motors: one for steering the car and the other for propulsion. A Raspberry Pi3 (RPi3) is the onboard computational unit that performs all the required computations and interfaces with the sensors. The available sensors include a forward-looking RGB camera, a forward-looking LIDAR, and an IR-Optocoupler attached to the wheel for speed measurements. The car is operated in two modes: a manual mode for collecting the data (images, speed, and steering control actions) and an autonomous mode for autonomously driving the car using LEC. Finally, control of the car is done with two controllers. The first controller is an OpenCV lane following controller designed using classical image processing algorithms. This controller takes forward-looking camera images as input to predict a discrete steering control action for the car. The second controller is a modified version of NVIDIA's DAVE-II DNN [4] that takes forward-looking camera images and the speed measurement as inputs to predict a continuous steering action for the car. With these controllers, the car operates at approximately 10 Frames Per Second. Please refer to our github[1] for comprehensive information about the platform. This platform has been published in the following publications:

- **Shreyas Ramakrishna**, Abhishek Dubey, Matthew P. Burruss, Charles Hartsell, Nagabhushan Mahadevan, Saideep Nannapaneni, Aron Laszka, and Gabor Karsai. "Augmenting learning components for safety in resource constrained autonomous robots." In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 108-117. IEEE, 2019.
- Matthew P. Burruss, **Shreyas Ramakrishna**, Gabor Karsai, and Abhishek Dubey. "Deepnncar: A testbed

[1] https://github.com/scope-lab-vu/deep-nn-car


Figure 3.1: DeepNNCar platform. (left) The DeepNNCar autonomous driving testbed, and (right) indoor tracks that were used in our experiments.

for deploying and testing middleware frameworks for autonomous robots." In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 87-88. IEEE, 2019.

## 3.2 Autonomous Vehicle in Carla Simulation

The second platform is an autonomous vehicle (AV) system in CARLA simulation [111] (see Fig. 3.2). The goal of the vehicle is to navigate an urban town setting under adverse weather conditions (e.g., high brightness, high precipitation) and sensor faults (e.g., camera occlusions, camera blur) without colliding with pedestrians and other vehicles in its travel path. The vehicle uses a total of nine sensors, including three forward-looking cameras, two radars, an inertial measurement unit (IMU), a global positioning system (GPS), and a speedometer. It has an LEC controller for navigating the AV, which is adapted from the work by Chen *et al.* [5]. An automatic emergency braking system (AEBS) based supervisory controller is also available for emergency braking. This simulation operates at a fixed rate of 20 Frames Per Second. Also, to mimic a resource-constrained setting, several resource restrictions (limits on memory and CPU cores) are placed on certain components during the simulation.

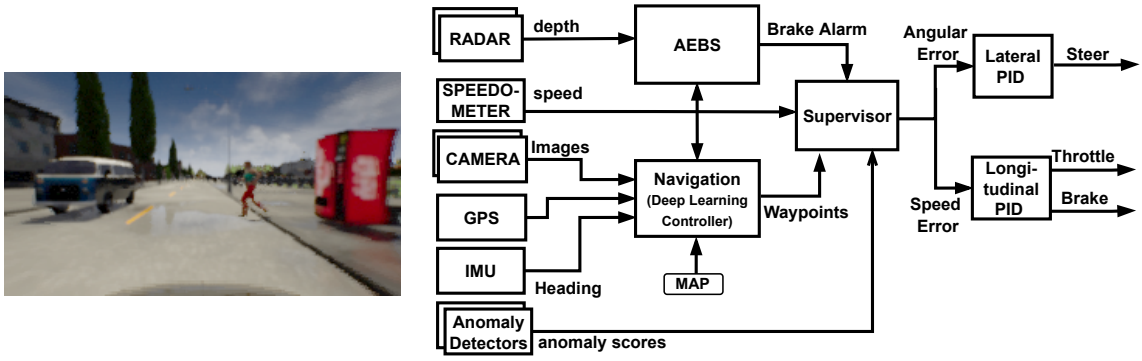



Figure 3.2: CARLA simulation. (left) image captured from the forward-looking camera of the AV as it navigates through the urban town setting in CARLA simulation. (right) the AV system model.

# Chapter 4

# Designing Static Assurance Cases for Complex CPS

## 4.1 Overview

As discussed, an assurance case (AC) is one design-time safety engineering approach that is increasingly being required for safety certification of cyber-physical systems (CPS) in several safety-critical fields, such as automotive [83], aviation [84] and medical devices [85]. For example, developing the AC is required to comply with the ISO 26262 [41] safety standard in the automotive domain, and it has also taken an overarching role in the recently formed UL4600 [43] standard for autonomous vehicles (AV). Despite its importance, there are two limitations in the existing assurance development process.

1. Today, AC development is performed manually by several engineers from different teams without a systematic approach or thorough consideration of safety arguments. While this unsystematic approach has worked in the past for small systems, the ever-increasing complexity of CPSs has made the assurance process complex, labor-intensive, and time-consuming.

2. Human evaluators currently perform the evaluation manually, which cannot scale to the growing complexity of CPSs [87]. This problem is further aggravated by the lack of widely accepted evaluation metrics and principled means to decompose evaluations for complex systems.

Addressing these limitations requires an adequate tool that can partially automate the activities involved in the development and evaluation process. To automate certain tasks of the safety assurance process, this chapter presents a workflow for developing graphical ACs from system artifacts (e.g., hazard and risk analysis information, system models) that are available at design time. The generated AC is then automatically evaluated for coverage using two coverage metrics. The feasibility of the workflow is demonstrated by automatically developing and evaluating an AC for an AV example in CARLA simulation [111].

The work comprising this chapter has been accepted for publication in the 41st International Conference on Computer Safety, Reliability and Security (SAFECOMP). This builds on [114] which was published in the Thirteenth International Tools and Methods of Competitive Engineering Symposium (TMCE).

- **Shreyas Ramakrishna**, Hyunjee Jin, Abhishek Dubey, and Arun Ramamurthy. “Automating Pattern Selection for Assurance Case Development for Cyber-Physical Systems”, Accepted for publication in the 41st international conference on Computer Safety, Reliability and Security (SAFECOMP), 2022.

- **Shreyas Ramakrishna**, Charles Hartsell, Abhishek Dubey, Partha Pal, and Gabor Karsai. "A Methodology for Automating Assurance Case Generation." In Tools and Methods of Competitive Engineering (TMCE), pp. 265-278. 2020.

## 4.2 Introduction

**Problem Domain**: Assurance Cases are being required for regulatory acceptance of CPSs in several safety-critical applications, such as automotive [83], aviation [84], and medical devices [85]. For example, the development of an AC (with a focus on safety) is a requirement for compliance with the ISO 26262 safety standard in the automotive domain [83]. An AC is a structured argument, supported by evidence, intended to demonstrate that the system satisfies its assurance guarantees under a particular context and under a given set of assumptions about the behavior of the system's components and its operating environment [38]. Goal structuring notation (GSN) [232] has been a widely used graphical modeling language used to represent the assurance arguments.

**State-of-the-art and Limitations**: AC development is getting increasingly difficult owing to the growing complexity of CPSs. Tools like Advocate [187], Resolute [185], Isabelle [233], and AMASS [234], among others [235] have been developed in recent years. In addition to managing the requirements and artifacts, these tools utilize modular assurance arguments called assurance case patterns[1] [176] to handle the size and complexity of AC being developed. Patterns are argument fragments that provide a partial solution for one aspect of the overall AC. They provide a reusable structure through parameter placeholders that can be instantiated with system-specific artifacts and assembled with other patterns into an AC.

While these tools specialize in data management and automation of the instantiation and assembly algorithm, an activity that has not been researched is the automation of the pattern selection process. To contextualize the selection process, consider the AV example in Fig. 4.1. Assume we want to develop an AC with the goal that the "automatic emergency braking system (AEBS) will function satisfactorily in applying the emergency brake", given that the operating context is a clear day. For this, we are given an artifact database with system architecture, component decomposition, component testing results (from different contexts like a clear day, rainy day, night, etc.), and a pattern database with patterns related to requirement decomposition, component decomposition, and failures, functional decomposition, hazard decomposition, etc. The problem is to select patterns that (a) support the goal and (b) have all the artifacts in the given context required for instantiation. Typically, a designer manually compares each pattern against the system artifacts to check if all the artifacts required for instantiation are available [178]. It is assumed the designer has complete knowledge

[1] In the rest of this paper, we will refer to "AC patterns" as "patterns"

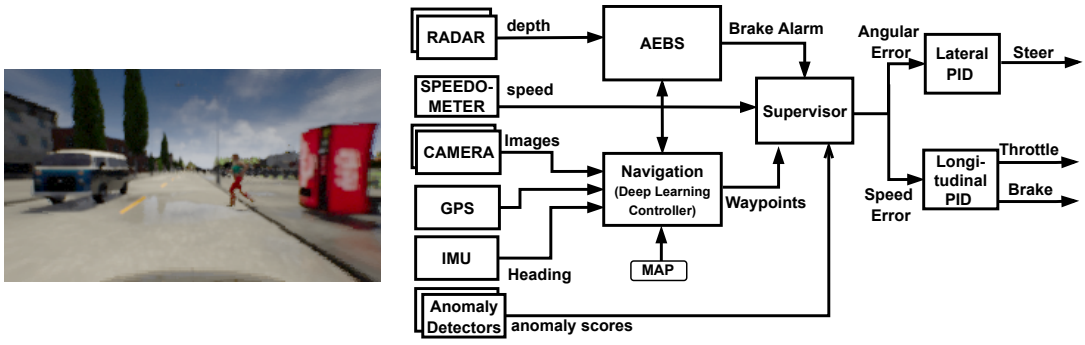



Figure 4.1: (left) Image from the forward-looking camera of the AV in CARLA simulation. (right) The system model of the AV.

of the system artifacts and is familiar with the content of the patterns and the context to which they are applicable. However, this comparison gets complicated and tedious for complex systems with more goals and diverse heterogeneous system artifacts [234]. For example, in one of the recent studies, Yamamoto, Shuichiro *et al.* [236] have shown that manual pattern selection took one designer 30 hours (14% of the development time) and required significant understanding of the available artifacts and patterns. Therefore, automating the selection process can aid the assurance process.

**Contributions**: To close this automation gap, we have developed a workflow that handles the selection problem as a coverage problem, intending to find the smallest set of patterns that can cover the system artifacts. For this, we leverage the ontology graph of the system artifacts and patterns and perform graph analytics. We address the coverage problem using an optimization problem setup, which is assisted by a data preparation function that utilizes a weaving model[2] [237] to generate data files, a mapping file, and an ontology graph of the artifacts. A selection function uses the processed files and a database of patterns to select a set of patterns, which are then plugged into an instantiation function to develop an AC. Finally, the AC is evaluated for coverage, and a report with information about unused artifacts and patterns is generated to aid the developer with future refinement. To evaluate the workflow, we have integrated it with a newly developed tool called ACCELERATE to automatically construct an AC for an autonomous vehicle [3] example within a CARLA simulation [111].

**Outline**: The rest of this chapter is organized as follows. In Section 4.3, we provide background on AC and AC patterns. In Section 4.4, we present the proposed workflow that includes the data preparation, pattern selection, and evaluation functions to automate the development and evaluation of an AC. In Section 4.5, we demonstrate the utility of our workflow with an AV example in CARLA simulation. Finally, we present the related research in Section 4.6 followed by our conclusion in Section 4.7.

[2]Captures the fine-grained relationships between different system artifacts
[3]For the CARLA AV setup, visit https://github.com/scope-lab-vu/AV-Assurance

While the details relevant to an AC may differ between systems, the structure of the arguments may follow a common pattern. Reusing such common patterns can aid the safety documentation process by avoiding duplication. To reuse and systematically design arguments for complex systems, Kelly introduced the AC patterns [176]. Patterns [176] are argument fragments that provide a partial solution for one aspect of the overall AC. They capture common repeated argument structures and abstract system-specific argument details as placeholders with free parameters to be instantiated. Patterns typically include name, intent, motivation, structure, applicability, related patterns, description of the decomposition strategy, and implementation details.

Kelly also made several extensions to include these patterns in the GSN modeling language for simplifying the assembly of these patterns into an AC. These extensions have a set of structural and entity abstractions as shown in Fig. 4.2. The structural abstractions allow the generalization of the object (goal) relationships. The structural abstractions include multiplicity extensions to generalize n-ary relations between the GSN nodes and optionality extensions to represent optional and alternative relations between the nodes. In addition, for generalizability, the modeling language includes placeholders that can be instantiated with system-specific information. These extensions to the GSN language allow several such patterns with placeholders to be instantiated and assembled into an AC.

Fig. 4.2 illustrates an example requirements decomposition pattern based on the formalization in Definition 1. This pattern argues for the satisfaction of the system's high-level requirements through the requirements decomposition of all the associated components. A node in this pattern is represented by its labels (e.g., G1, G2) and content with placeholders (e.g., SM, C) that can be replaced by system-specific information. The node multiplicity (*mul*) is marked on the graph edges, representing how one node is related to another. Further, "instantiate" and "develop" are the two modifier (*mod*) operations that can be performed on the nodes.

### 4.3.1 Pattern Formalization

We adapt the formal definition of patterns as presented by Denney *et al.* [238] with slight modifications, including a metadata field that holds additional information about the pattern and a modifier function that specifies the operations that need to be performed on the nodes. We provide a formal definition below:

**Definition 1 (Pattern)** *A pattern $\mathscr{P}$ is a tuple $\langle \mathscr{M}, \mathscr{N}, l, t, i, mul, c, mod, \rightarrow \rangle$, where $\langle \mathscr{N}, \rightarrow \rangle$ is a finite, directed hypergraph in which each edge has a single source and possibly several destination targets. $\rightarrow$: $\langle \mathscr{N}, \mathscr{N} \rangle$ is the connector relation between nodes, $\mathscr{M}$ is the pattern metadata, and the functions l, t, p, mul, and mod are defined below:*

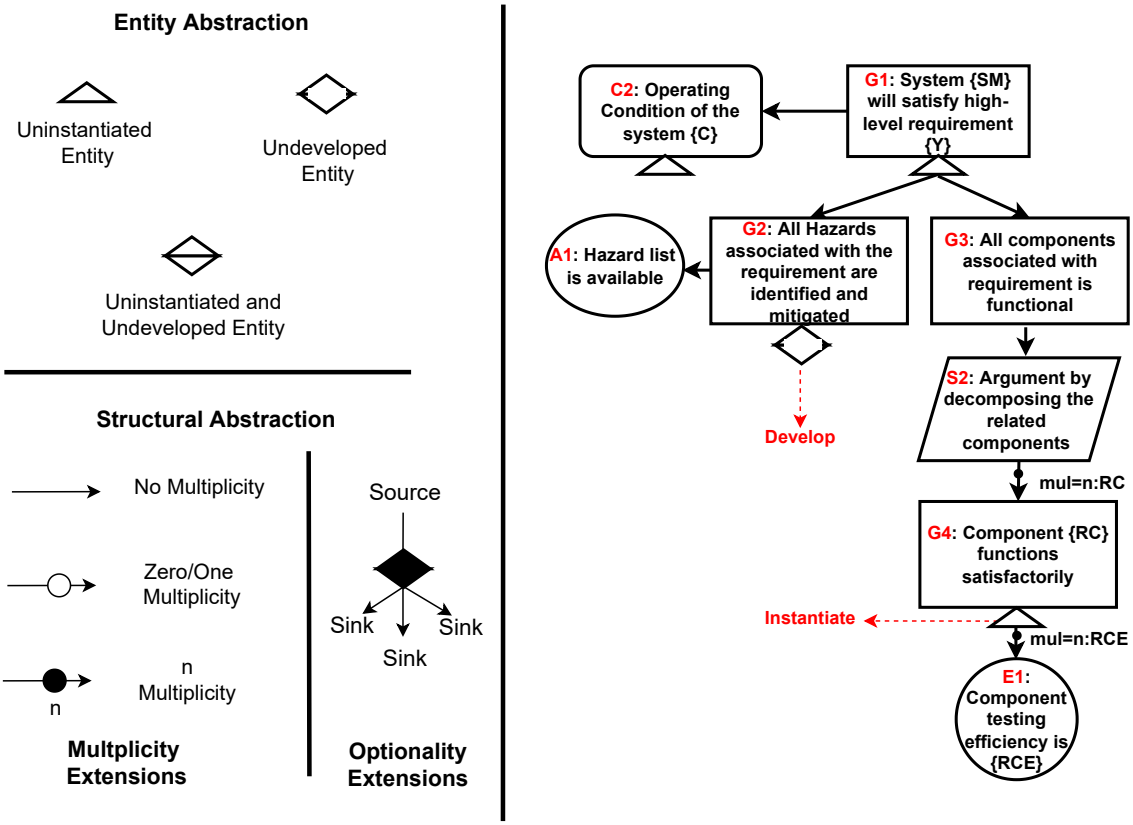



Figure 4.2: (Left) GSN pattern abstractions. (Right) Example pattern based on requirements decomposition.

- $\mathscr{M}$ is the pattern metadata tuple $\langle N, R, pl \rangle$, where $N$ is the name of the pattern, $R$ is a set of relevant patterns that share the same intent as this pattern or patterns that can be composed with this pattern for further growing the assurance case, and $pl$ is a dictionary that maps a system artifact label (key) to a placeholder variable (value) that requires instantiation. The artifact label is of the type string. An illustrative placeholder dictionary is of the form {"system" : SM, "top-level-goals : TG", "requirements" : SR}.

- $l$ and $t$ are labeling functions, where $l : \mathscr{N} \rightarrow \{g, s, c, a, j, e\}$ maps each node onto its type, namely on g (goal), s (decomposition strategy), c (context), a (assumption), j (justification), or e (evidence).

- $i$ is the id label of each node, $i$: $\mathscr{N} \rightarrow id \times class$, which returns the identifier and the type of each node, i.e. class = $\{g, s, c, a, j, e\}$.

- $mul$ provides a multiplicity label for each outgoing connector. For the example shown in Fig. 4.2, $mul = n : RC$ represents a one-to-many relationship, where n is an integer value determined by placeholder $RC$, such that each instance of node $S2$ is related to $n$ instances of node $G4$. The relationship is one-to-one if not explicitly stated otherwise.

- $mod$ indicates the modifying operation to be performed on a given node: no-operation, instantiate or develop.

In addition to the pattern entities, there are several structural rules required:

- The root node of a pattern is always a goal.
- The connectors can only go out of the goal and the strategy nodes: $n_a \rightarrow n_b \Rightarrow \mathrm{l}(n_a) \in \{g, s\}$.
- A strategy node cannot directly connect to another strategy node or an evidence node: $(n_a \rightarrow n_b) \wedge [\mathrm{l}(n_a)\text{=s}] \Rightarrow \mathrm{l}(n_b) \in \{g, a, c, j\}$.

### 4.3.2 System Artifact Engineering

As discussed, instantiating the pattern requires system artifacts (information about the system) often curated by a design and engineering procedure prescribed by the industry standards [41, 239]. While this process slightly varies across standards and application domains, the overall steps remain the same. We will briefly describe these steps in this section. We encourage interested readers to read these research findings [240, 241, 242, 243, 244] and the ISO 26262 standard [41] to get a deeper insight into these engineering steps.

The first step is to perform *requirements engineering*, which involves developing the system overview, defining the system goals and requirements, modeling the system's architecture using languages like architecture analysis and design language (AADL) and System Modeling Language (SysML), allocating requirements to subsystems, and identifying the system failure modes with techniques like failure mode and effect analysis (FMEA) [156], and fault tree analysis (FTA) [157]. The next step is to perform *hazard analysis and risk assessment* (HARA), which involves identifying events (or threats) that may lead to hazards and consequences for the system, designing strategies to mitigate the hazard, and estimating the risk posed by the identified hazard. Risk assessment typically involves two steps. The first step involves analyzing and quantifying the risk. The analysis includes visualizing the hazard propagation paths using graphical tools like Bow-Tie Diagrams (BTDs). Next, the system risk is computed qualitatively or quantitatively using tools like risk matrix [37] and BTD [59]. The final step is to perform *evidence generation*, which involves generating system-related evidence using formal verification, simulation, testing, and expert opinions.

The artifacts generated from these activities are usually in heterogeneous file formats like PDF, Text, AADL, and SysML model files. These artifact files are typically stored in an artifact database and utilized during the AC development process. It is important to note that the quality of the developed AC is highly dependent on the correctness and completeness of the system-related artifacts curated during these engineering procedures.

## 4.4 Workflow for Assurance Case Development

We present the proposed workflow that leverages an ontology graph of the system artifacts and patterns to automatically select patterns that can be instantiated to construct an AC. The workflow is composed of several functions that work as follows: First, the $prepare(\mathscr{AD}, WM)$ function uses a weaving model ($WM$) to map artifact files from the artifact database ($\mathscr{AD}$) onto several data files ($F_D$) and a mapping file $F_M$. Then, the function $select(\mathscr{AD}, \mathscr{PD})$ selects a set of patterns $\mathscr{P}_S$ from the pattern database ($\mathscr{PD}$). The selected patterns are instantiated and assembled into an AC using an external *instantiate* function. Finally, the $evaluate(AC, F_D, F_M)$ function generates a report with the coverage score ($CS$) and additional information to aid the evaluation and further refinement of the AC. We discuss these functions in the rest of this section.

### 4.4.1 Data Preparation

The artifacts (e.g., goals, requirements, system models) for the assurance process are typically curated using several engineering activities (see Section 4.3.2) and stored in a database $\mathscr{AD}$. Given these files, the *prepare* function takes in these raw artifact files to prepare the processed files required for the pattern selection discussed in the next section. The function performs two operations as shown in Algorithm 1.

The first operation processes relevant artifacts required for the AC into processed data files stored in tabular format (CSV file). The function can currently process AADL files. We are working towards automatically processing other file formats. Then, the processed files are checked for completeness, correctness, and relevance. The check is to ensure that only complete and essential artifacts necessary for the development of the AC are retained while discarding the non-essential artifacts. Non-essential artifacts bloat the $\mathscr{AD}$, which slows the selection process and impacts the evaluation metrics (discussed later in Section 4.5). In the current implementation, the checking is manually performed by a designer. We assume the designer has complete knowledge of the system for which the AC is being developed. The accepted files are passed through an arrange function to generate a set of data files $F_D$. To generate the file, we use a weaving model $WM$ that weaves the different artifacts and transforms them into a single model file. The model is developed based on our domain knowledge and previous experience with CPSs. Each data file is a table where the column headers represent the name of the artifacts, and the rows capture the content of these artifacts. Also, each column in the data file is related to the other columns, with the relationship derived using the weaving model. Finally, these accepted files are manually tagged with labels required for the selection process. For example, the context in which particular artifacts and evidence are applicable is one label that we currently include.

The second operation generates a mapping file $F_M$, which is a lookup table of system artifacts and their ontology required to bridge the data files for the pattern selection algorithm discussed in Section 4.4. $F_M$

**Algorithm 1** Data Preparation

**function** PREPARE($\mathscr{AD}$:Artifact Database,$WM$:weaving model)
  $F_D \leftarrow \{\}$, $F_M \leftarrow \{\}$, $temp \leftarrow \{\}$
  **for** each $file$ in $\mathscr{AD}$ **do**
    processed file $\leftarrow$ process($file$)
    $temp \leftarrow temp \bigcup \{$processed file$\}$
  **end for**
  $accepted\_files \leftarrow manual\_check(temp)$
  **for** each $file$ in $accepted\_files$ **do**
    data file $\leftarrow$ arrange($file$,$WM$)
    $F_D \leftarrow F_D \bigcup \{$data file$\}$
  **end for**
  $place \leftarrow \{\}$, $depend \leftarrow \{\}$, $source \leftarrow \{\}$
  **for** each $file$ in $F_D$ **do**
    $source \leftarrow get\_source\_query(file)$
    $place \leftarrow extract(header)$
    **for** each $entry$ in $header$ **do**
      result $\leftarrow search(entry, F_D)$
      depend $\leftarrow$ result
    **end for**
    $F_M \leftarrow \{place, depend, source\}$
    $\mathscr{G}_A \leftarrow make\_graph$(place,depend)
  **end for**
  $\mathscr{AD} \leftarrow F_M, F_D, \mathscr{G}_A$
  **return** $F_M$, $F_D$
**end function**

holds the physical link to the data file location obtained using a simple query to the database. It also holds the placeholder and dependency mapping derived from an *extract* function, which reads the header of each data file to create an intra-file dependency mapping between them. For example, one entry capturing the relationship between a cause and a hazard in the dependency mapping file looks like $[cause, cause\_table, hazard]$. If there are multiple causes for the same hazard, they will be stored as separate entries. Next, to capture the inter-file dependencies, each header (e.g., cause) is searched across every data file using a *search* function. The search result is used for placeholder mapping, required for pattern selection. For example, the search result for the cause header is $\{[mitigation\_table, cause], [cause\_table, cause], [risk\_table, cause]\}$, which shows all the other files in which the entry is present. Finally, the ontology captured in $F_M$ is also stored as an artifact graph $\mathscr{G}_A$ as shown in Fig. 4.3.

Then, we curate $\mathscr{PD}$ for which we gather patterns from online catalogs [176, 178] and manually re-design them using the formalization and rules discussed in Section 4.3. While re-designing, a designer checks the language consistency across data in the nodes. These patterns are stored in textual format (JSON) and as a graph $\mathscr{G}_P$ with placeholders as nodes (See Fig. 4.3).

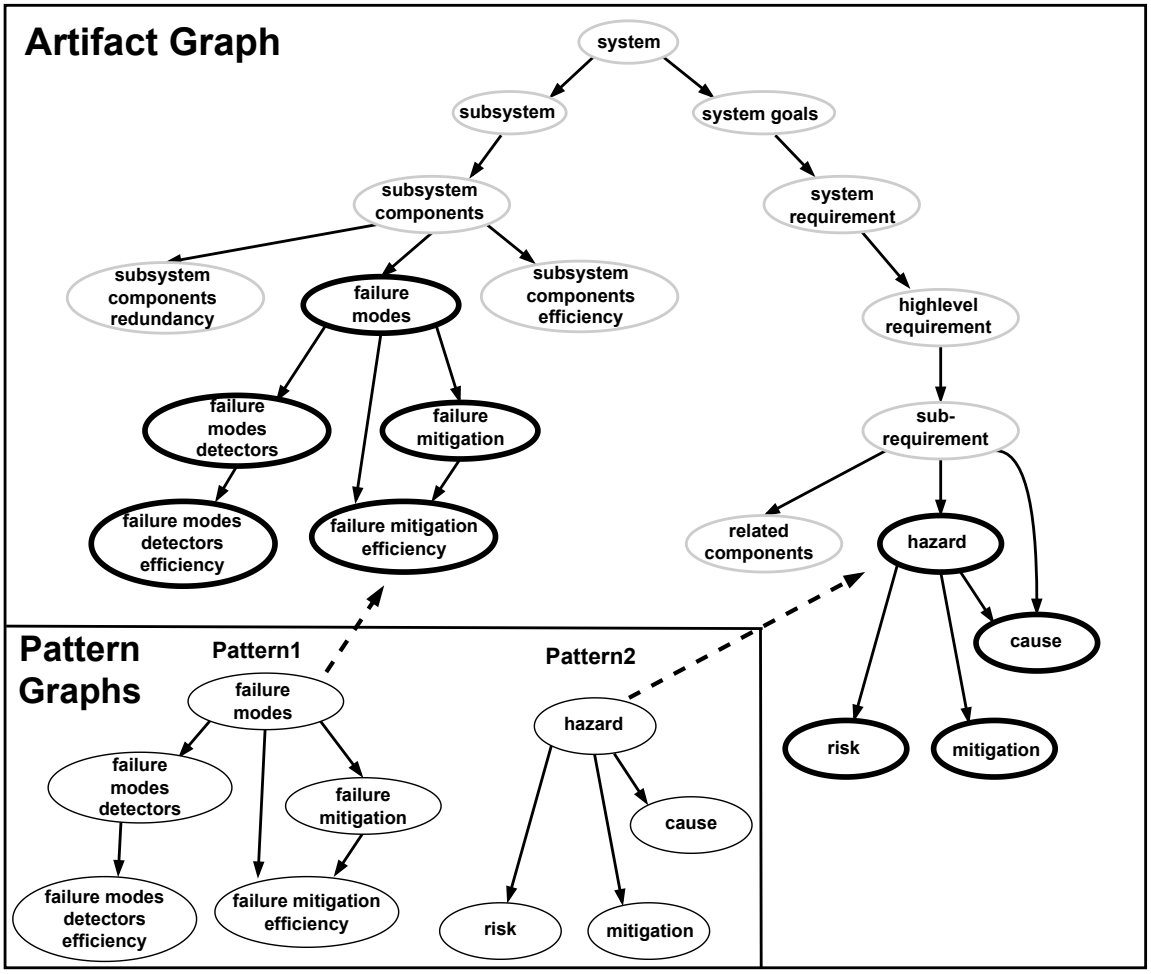



Figure 4.3: The artifact and pattern ontology extracted for pattern selection. The selected patterns are highlighted in red in the artifact graph.

### 4.4.2 Pattern Selection

As discussed earlier, the goal of the selection algorithm is to select a smallest set of patterns $\mathscr{P}_S$ from the database $\mathscr{PD} = \{\mathscr{P}_1, \mathscr{P}_2, \cdots, \mathscr{P}_n\}$ that maximizes the artifact coverage. We formulate the selection as a two-objective optimization problem: (a) maximizing the coverage such that the placeholders of every selected pattern have the corresponding artifact for instantiation and (b) minimizing the number of patterns selected by iteratively comparing the pattern graph to the artifact graph (See Fig. 4.3).

The optimization is realized using the $select(\mathscr{PD}, F_M, F_D)$ function shown in Algorithm 2. It takes the patterns from $\mathscr{PD}$, the mapping file, and the data files as inputs to select $\mathscr{P}_S$. The selection is performed using the *findmatch*, the *findconflict*, and the *findsubgraph* functions. The selected patterns are then instantiated and assembled into an AC using an external *instantiate* function. An existing algorithm [189, 245] can be used for instantiation and assembly.

**Algorithm 2** Pattern Selection

```
1:  function SELECT(𝒫𝒟:Pattern Database,F_M:Mapping File,F_D:Data Files)
2:      𝒫_S ← {}, dups ← {}
3:      for each pattern 𝒫 ∈ 𝒫𝒟 do
4:          temp ← True
5:          for each placeholder p ∈ 𝒫 do
6:              temp ← temp ∧ findmatch(p)
7:          end for
8:          if temp is True then
9:              𝒫_S ← 𝒫_S ⋃ {𝒫}
10:         end if
11:     end for
12:     for each pattern 𝒫 ∈ 𝒫_S do
13:         if findsubgraph(𝒢_A, 𝒢_P) is False then
14:             𝒫_S.remove(𝒢_P)
15:         end if
16:     end for
17:     for i ← 1 to len(𝒫_S) do
18:         for j ← i+1 to len(𝒫_S) do
19:             match ← findconflict(𝒫_i, 𝒫_j)
20:             if match is True then
21:                 dups ← dups ⋃ {𝒫_j}
22:             end if
23:         end for
24:     end for
25:     for each entry E in dups do
26:         𝒫_S.remove(E)
27:     end for
28:     AC ← instantiate(𝒫_S,F_M,F_D)
29:     return AC
30: end function
```

Next, the $findcomplete(\mathscr{P}, F_M)$ function in Definition 2 checks if the placeholders in the patterns have a matching entry in $F_M$. If all the placeholders have corresponding entries, the pattern is said to be complete, and it is added to $\mathscr{P}_S$. Otherwise, the pattern is discarded from the selection process.

**Definition 2 (Pattern Completeness)** *We say a pattern is complete if each placeholder has a corresponding entry in $F_M$. We define a function $findmatch(pl)$ that determines if a given placeholder has a corresponding entry.*

Once $\mathscr{P}_S$ has been selected, the *findsubgraph* and the *findconflict* functions are used to minimize the cardinality of $\mathscr{P}_S$ and remove duplicate patterns. First, the $findsubgraph(\mathscr{G}_A, \mathscr{G}_P i)$ function checks whether the artifact graph $\mathscr{G}_A$ contains a subgraph that is isomorphic to $\mathscr{G}_P i$, the graph of the $i^{th}$ pattern. Two graphs are isomorphic (or equivalent) if their structures preserve a one-to-one correspondence between their vertices and between their edges. For example, in Fig. 4.3, pattern1 and pattern2 are isomorphic to subgraphs of $\mathscr{G}_A$. The non-isomorphic patterns are removed from $\mathscr{P}_S$.

Next, the $findconflict(\mathscr{P}_1, \mathscr{P}_2)$ function checks $\mathscr{P}_S$ for redundant patterns. For this, it performs the following steps: (a) duplication checking checks if the patterns have the same set of placeholders requiring instantiation (see Definition 3). (b) graph checking checks if the graphs of the two patterns are isomorphic. While performing this, we also require data on corresponding nodes of the patterns to be equivalent. Only if the duplication checking fails, the function performs graph checking. On the whole, the function $findconflict$ returns true if steps (a) or (b) return true, i.e., if the patterns are redundant.

**Definition 3 (Duplication Checking)** *We say two patterns* $\mathscr{P}_1 = \langle \mathscr{M}_1, \mathscr{N}_1, l_1, t_1, i_1, m_1, s_1, \rightarrow_1 \rangle$ *and* $\mathscr{P}_2 = \langle \mathscr{M}_2, \mathscr{N}_2, l_2, t_2, i_2, m_2, s_2, \rightarrow_2 \rangle$ *are duplicates if they contain exactly the same placeholders.*

### 4.4.3 Coverage Evaluation

As discussed previously, automating different activities of the assurance process reduces the development time and manual efforts. However, this gain is at the expense of increased effort and time required to review and evaluate the quality and correctness of the generated AC. To aid the evaluation process, the $evaluate(AC, F_D, F_M)$ function takes the AC and generates a report to provide qualitative insights which is not available in the generated AC graphical structure. We believe this information can aid the designer in further refinement. The report includes the coverage score, the selected and unused patterns, and the unused artifacts. The coverage score is a tuple $\langle \mathscr{A}, \mathscr{S} \rangle$ of the artifact coverage ($\mathscr{A}$) and the problem coverage ($\mathscr{S}$).

**1) Artifact Coverage ($\mathscr{A}$):** The artifact coverage metric measures the proportion of the artifacts available in $\mathscr{AD}$ that have been included in the AC. Also, a relevance check (discussed in Section 4.4.1) on the artifact is essential for this metric to be accurate. Besides the score itself, this metric can also be used to derive a list of unused artifacts.

$$\mathscr{A} = \frac{\text{\# Artifacts used in the AC}}{\text{\# Artifacts available in } \mathscr{AD}} \tag{4.1}$$

**2) Problem Coverage ($\mathscr{S}$):** The problem coverage metric quantifies the coverage of all the known problems affecting a system's property (e.g., safety, availability). It is a tuple $\mathscr{S} = (C_{\mathscr{P}_1}, C_{\mathscr{P}_2}, \cdots, C_{\mathscr{P}_n})$ consisting of coverage measures $C_{\mathscr{P}_i}$ related to different problem classes $\mathscr{P}_i$. The coverage measure is shown in Eq. (4.2), and it is computed as the percentage of problems within a given problem class that are addressed by an AC.

$$C_{\mathscr{P}_i} = \frac{\text{\# Problems from } \mathscr{P}_i \text{ addressed by the AC}}{\text{\# Problems in } \mathscr{P}_i \text{ identified during analysis}} \tag{4.2}$$

While coverage metrics and the report can aid the refinement process by providing insights into the missing patterns or artifacts, they do not fully quantify the quality of artifacts (e.g., evidence) required for AC selection.

So, the coverage metrics cannot solely rely on measuring the quality of the AC. A combination of coverage and confidence metrics is needed for robust quantitative assessment. We are therefore working towards integrating a confidence metric.

## 4.5 Illustrative Example

In this section, we provide an illustrative example by applying the proposed workflow to develop an AC for an AV in the CARLA simulator [111]. In this example, the AV is required to navigate a town while avoiding collisions with obstacles in its travel path. We integrate our workflow with the ACCELERATE tool [4] for pattern instantiation and assembly as shown in Fig. 4.4.

**Artifacts and Patterns Preparation**: We performed the analysis steps listed in Section 4.4.1 to curate $\mathscr{AD}$. We first performed a requirement and system analysis using the given requirements document. The vehicle has three goals associated with two system requirements and three high-level requirements, each associated with several sub-requirements. We then designed the AV system model shown in Fig. 4.1. It has a navigation component that uses three cameras, a global positioning system (GPS), an inertial measurement unit (IMU), a speedometer, and a route planner to compute the vehicle's next position. Then a velocity planner calculates the average velocity needed to traverse from the current position to the next position. The velocity and the camera images are fed to a deep-learning controller to predict the waypoints, which are passed to a motion estimator to compute throttle, brake, and steer errors. In addition, it has an AEBS controller that uses two radars to raise a brake alarm on detecting obstacles. We then performed fault analysis of the system model to identify 14 component faults and analyzed the possible mitigation strategies. Further, we performed hazard analysis to identify eight operational and functional hazards associated with the different system components. Finally, we curated $\mathscr{PD}$ for which we gathered several patterns from online catalogs [176, 178] and re-designed them using the formalization discussed in Section 4.4.2.

**Results**: We applied the integrated tool to develop an AC for the vehicle. We summarize the key results in terms of the coverage metrics and the size of the AC (computed in terms of GSN nodes) in two revisions. We used the AC report[5] from "revision1" to refine the AC in "revision2". The pattern database had seven patterns for the selection process to choose from in these revisions. The analysis of the revisions is: In "revision1", four patterns were selected to develop an AC with 805 nodes. The evaluation function returned a coverage score with artifact coverage of 76%, a problem coverage of $[C_H : 60\%, C_F : 100\%]$ with five unused artifacts. Here, $C_H$ represents the percentage of known hazards that are covered, and $C_F$ represents the percentage of known system faults that are covered. From the report, we analyzed the artifacts relating to failure decomposition that

[4] Tool is being built as part of the DARPA ARCOS program. Check our GitHub for release information.

[5] For a bird's eye view of the "revision1" assurance case and the report, visit https://github.com/scope-lab-vu/AV-Assurance

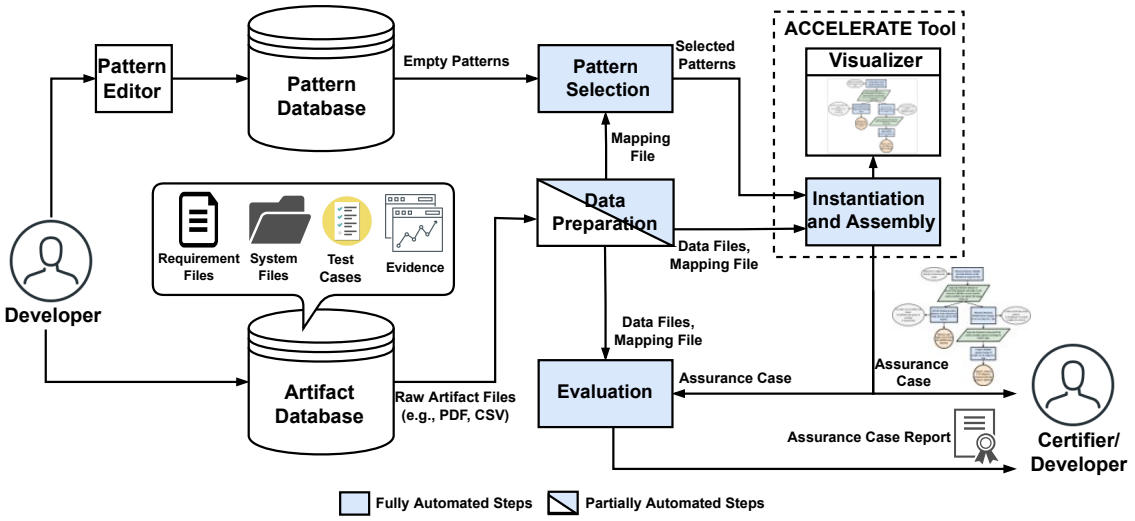



Figure 4.4: The proposed pattern selection workflow integrated with the ACCELERATE tool for AC development.

was unused. So, we designed a new failure decomposition pattern that was added to $\mathscr{PD}$. Further, some of the sub-requirements associated with the hazards were missing, which we included. In "revision2", the selection mechanism selected five patterns, including the new pattern to develop an AC with 909 nodes. The refined AC had a higher coverage with an artifact coverage of 90%, a problem coverage of $[C_H : 85\%, C_F : 100\%]$ with unused artifacts reduced by two. We performed several iterations until all the artifacts were included in the AC.

To estimate the time saved by the workflow, the data preparation, and selection steps were first performed manually by a developer who performed the following tasks: re-design of patterns into the defined formalization, which took approximately one hour, processing the artifact files, extracting the artifact dependencies to generate an ontology file, and instantiation and assembly of the patterns using the ACCELERATE tool. While instantiation and assembly were performed in less than a minute, the manual selection and data curation process took approximately three hours. Next, for comparison, we fed the manually re-designed patterns and the artifacts to the integrated tool (See Fig. 4.4), which only took close to one minute for data preparation, pattern selection, instantiation, and assembly. Finally, to stress test the integration, we increased the artifacts and patterns in the database. Our workflow took less than five minutes, even for a large AC with 1500 to 3000 nodes. Significantly less manual processing was needed when the artifact files were changed or updated. We expect the time saving to get even more significant as the size of the artifact database grows. However, the manual steps involved in the data preparation step are a bottleneck for scaling the workflow, which we want to address in the future.

## 4.6 Related Work

The existing AC development approaches can be broadly classified into one of these approaches [180]. (1) *Prescriptive*, where safety standards heavily guide the development process. This approach was long used in several applications like nuclear plants and oil pipelines. However, the reliance on generic standards and regulations resulted in several well-known accidents like the clapham rail disaster [172], and the piper alpha off-shore oil and gas platform disaster [173], (2) *Goal-Oriented*, where high-level safety goals are specified but the process for achieving them is not guided but left to the developer. As discussed earlier, the last two decades have seen the certification process migrate from a prescriptive to a goal-oriented paradigm [180, 48]. Goal-oriented approaches such as Risk-Informed Safety Case [30] approach from NASA is increasingly being developed, and (3) *Blended*, where both prescriptive and goal-oriented approaches are fairly used in the development process.

The last decade has seen several commercial and research tools using one of the above-listed approaches to support different activities of the AC development process. A comprehensive survey on these tools is available in [235]. This survey reports 46 commercial and research tools and evaluates them based on their capability to generate, maintain, assess, and report safety cases. Our understanding from the survey is that the commercial tools mainly focus on providing a platform for developing and managing ACs. AC Construction and Evaluation Support System (ACCESS) [181] is a tool based on Microsoft Visio that aids in creating and maintaining ACs. It provides a platform for rapid prototyping, node creation, and node coloring of the GSN argument structure it generates. CertWare workbench [184] is another tool based on Eclipse that provides various functionalities like multi-user safety case editing, change tracking, standard safety case templates, and cheat sheets for simple and fast safety case development. Similarly, Assurance and safety case environment (ASCE) [182] provides an environment for simple safety case creation and management and allows for the simple and low-cost generation of safety case reports. Also, the D-Case editor from DEOS [183] is an open-source platform implemented as an eclipse plugin to generate and manage GSN argument structures.

There have been several research tools that support automation. Advocate [187] is one such tool that provides an editor for designing system architectures, patterns, and automated development of an AC from patterns. A pattern formalization and the instantiation algorithm are built into the tool for automating pattern instantiation [238]. Here, a pattern dataset ($\mathscr{P}$-dataset) and a parameter table ($\mathscr{P}$-table) are manually created to assist the instantiation algorithm. Resolute [185] is another tool that automatically synthesizes an AC from AADL models. Isabelle [233] is a recently developed tool with integrated formal methods for evidence generation. Assurance language and automated document processing are a few tool features that support the development process. AMASS tool [234] provides a partially automated heterogeneous collaborative

environment that supports activities such as requirement management, artifacts, evidence generation, pattern composition, and AC development.

There are several independent efforts. For example, Ramakrishna *et al.* [114] have presented a methodology to partially automate AC development directly from system models and graphs. Hawkins *et al.* [189] utilize the concept of model weaving to automatically learn the artifact files from system models and use them for instantiating patterns. The authors of [245] provide an automated mechanism for instantiation and composition of patterns, where the artifacts are heterogeneous system models that are linked to represent the cross-domain relationship. While these tools and approaches automate the instantiation and assembly of patterns, their selection largely remains manual.

Further, evaluation is key to automating AC development. Confidence metrics are often used to represent the assurance deficit [246]. However, there has been minimal work in coverage evaluation. Denney *et al.* [247] have presented several coverage metrics for different system artifacts like hazards and requirements. These metrics measure the proportion of the system artifacts used in the AC to those available in the database. Chindamaikul *et al.* [248] have presented two coverage metrics: a claim coverage that is like the ones in [247], and an argument coverage metric that measures the arguments and evidence covered in the AC. We build on the prior work to provide additional coverage metrics.

## 4.7 Conclusions and Future Work

In this paper, we have presented a workflow that can automate the pattern selection process. We formulate the selection problem as a coverage problem that selects the smallest set of patterns that can maximally cover the available system artifacts. The coverage problem is realized using an optimization problem that leverages the ontology graphs of the artifacts and patterns and performs graph analytics. The optimization is aided by an array of functions that perform data preparation, pattern selection, and AC evaluation. These functions collectively reduce the manual effort and time required in selecting the necessary patterns. Further, we plan to move this research in several directions. First, fully automating the data processing function using natural language processing (NLP). Second, design a translator to convert textual patterns into our format. Third, automate the language check using NLP and relevance check using topic modeling [248]. Finally, include confidence metrics for AC evaluation.

To conclude, while our automated workflow targets reducing the cost and time to safety certification, we think the generated AC and the evaluation report are insufficient to assure the safety of CPSs. This is because the development process hinges on the assumption that operating conditions of the system will remain within the distribution observed at design time. However, this notion is invalidated by the dynamic operating

conditions introducing new hazard types and failure modes not accounted for during the system's design. Such gaps in the documented safety and the actual safety at runtime have resulted in several accidents, including the RAF Nimrod aircraft accident [204]. Concluding several similar research findings [48, 55, 50], it is clear that design-time assurance techniques, alone are, not sufficient for assuring the safety of CPSs at runtime. Therefore, as noted by several authors [55, 48], safety assurance is not a one-time activity. It is a process that continues through the system's operational lifetime by including runtime monitors to detect whether the assumptions about the system's design-time operating conditions are valid. Therefore, an open question that still needs to be addressed in this dissertation is: *how can we efficiently detect violations in the design-time assumptions? Especially the implicit assumptions introduced by the LECs.*

# Chapter 5

# Simultaneous Out-of-distribution Detection and Diagnosis for CPS with LECs

## 5.1 Overview

Detecting violations in the assumptions of the design-time safety assurance models is a crucial step in our framework. While it is easy to detect changes in the explicit assumptions documented in the system's design documents, it is often hard to identify the implicit assumptions that are not explicitly stated. Inductive biases (e.g., training data, model hyperparameter) involved in training the learning enabled components (LECs) are one example of implicit assumptions that are of interest in this work. These assumptions are mostly not documented and go unnoticed at design time, which often leads to problems like the out-of-distribution (OOD) data problem [56, 57].

This problem arises because of the violation of the closed-world assumption that occurs when a trained LEC is deployed to operate in open-world scenarios [90]. The operational data to the LEC that does not belong to the dataset it was trained with is called OOD data. LECs tend to over predict the control actions for OOD data and must be detected before they result in a system consequence [26, 27]. For example, Cai *et al.* [92] have shown changes in the weather conditions affect perception-based LEC systems, resulting in their collision. While there have been abundant research findings [91, 92, 94, 95, 96] in identifying OOD data. Performing detection on high-dimensional sensor data such as images (which is of interest in this work) is expensive and requires additional resources and time for detection. These requirements are often unavailable on the low-resource platforms (e.g., DeepNNCar [80]) targeted in this work. So, an efficient detection mechanism is required to be useful for our approach.

Next, as OOD data can manifest in several ways depending on the changes in the data factors (e.g., image features), finding the root cause(s) factors causing the problem is critical. Several application areas of CPSs, such as system health management [249, 69] and software product lines [250] have been shown to benefit from anomaly explanation. Therefore, detecting when the assumptions are violated and interpreting the cause(s) of the violation can help the proposed framework.

This chapter presents an automated workflow to design and train a detector to perform OOD detection and factor identification on a low-dimensional latent space that requires lesser computational resources. We use a variant of the variational autoencoder (VAE) called the $\beta$-variational autoencoder to reduce the image dimensionality. We train the network to generate a disentangled lower-dimensional latent space on

which we perform detection and factor identification at runtime. The feasibility of the proposed workflow is demonstrated using two CPS applications: an autonomous vehicle (AV) example in CARLA simulation [111] and a real-world autonomous driving dataset called nuImages [251].

The work comprising this chapter has been published in Transactions on Cyber-Physical Systems (TCPS) [116]. This builds on [115] and [93], which were published in the 2020 International Conference on Embedded Software (EMSOFT), and the 2020 IEEE Security and Privacy Workshops (SPW).

- **Shreyas Ramakrishna**, Zahra Rahiminasab, Gabor Karsai, Arvind Easwaran, and Abhishek Dubey. "Efficient Out-of-Distribution Detection Using Latent Space of $\beta$-VAE for Cyber-Physical Systems." In Transactions on Cyber-Physical Systems (TCPS), 2021.
- **Shreyas Ramakrishna**, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. "Efficient Multi-Class Out-of-Distribution Reasoning for Perception Based Networks: Work-in-Progress." In 2020 International Conference on Embedded Software (EMSOFT), pp. 40-42. IEEE, 2020.
- Vijaya Kumar Sundar[1], **Shreyas Ramakrishna**[1], Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. "Out-of-distribution detection in multi-label datasets using latent space of $\beta$-VAE." In IEEE Security and Privacy Workshops (SPW), pp. 250-255. IEEE, 2020.

## 5.2 Introduction

**The Safety Conundrum:** CPS design flows focus on designing a system $S$ that satisfies some requirements $R$ in an environment $E$. During the design process, the developer selects component models, each including a parameter vector and typed ports representing the component interface, from a repository and defines an architectural instance[2] $A_S$ of the system such that $A_S \parallel E \models R$, while satisfying any compositional constraints across the component boundaries, often specified as pre-conditions and post-conditions [68]. The difficulty in this process is that in practice, the environment is only approximated using a surrogate model $\hat{E}$ or a set of observations $|\tilde{\hat{E}}|$ collected from real-world data. It is clear that $A_S \parallel \tilde{\hat{E}} \models R$ does not imply $A_S \parallel E \models R$. In this sense, the system is being deployed with the assumption that $\tilde{\hat{E}} \approx E$. However, this is not a strong guarantee and could result in scenarios where the designed architecture may fail in the physical environment. Hence, runtime monitoring of the system is required to identify when $\tilde{\hat{E}} \not\models E$, i.e., the observed samples are out-of-distribution (OOD) with the real environment.

To explain these problems, consider a perception LEC (e.g., NVIDIA DAVE-II deep neural network [4]) which consumes streams of camera images to predict the steering control action for end-to-end autonomous

[1] These authors have contributed equally

[2] architectural instance of a design is a labeled graph where nodes are the ports of components, and the edges represent interactions between the ports.

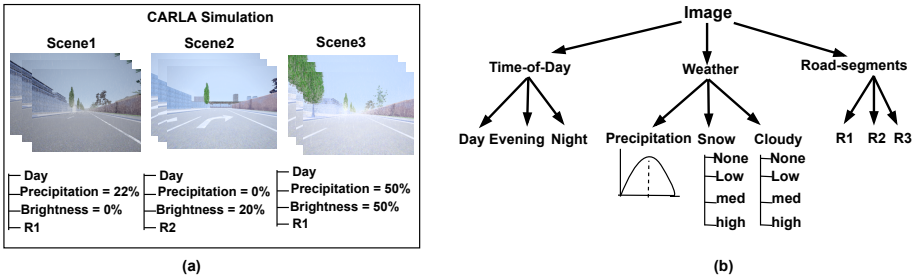



Figure 5.1: (a) scenes with feature labels from CARLA simulation, (b) hierarchical representation of an image using its features. The features can take discrete or continuous values.

driving. These streams of images can be categorized as *scenes* illustrated in Fig. 5.1. A scene is a short time series of similar images contextualized by certain environmental features such as weather, brightness, road conditions, and traffic density. These features can take a continuous or discrete value. Images from several such scenes collectively form a dataset on which the network is trained. For example, the nuScenes autonomous driving dataset [252] comprises 1000 scenes, each 20 seconds long and annotated with 38 feature labels regarding time-of-day, weather, traffic, road type, scenery, rain, pedestrians, night lights, and the vehicle type. The features and their value ranges specify the context in which the system operates and influences the predictions of the trained network. During operation, the network can encounter images having the same features with values belonging to the training dataset (in-distribution images). However, the network can also encounter OOD images of two kinds. *First*, the features of the images are the same as the training dataset, but their values may fall outside the training ranges. *Second*, the images may have new features (e.g., fog, snow) that were not present in the images of the training dataset. The second category is also known as open set classification. In this work, we are interested in the first category of images. The goal of detection, in this case, is to identify: (a) if the current image of an operational scene is dissimilar from the images previously used for training the LEC, and (b) the known image feature(s) responsible for the OOD data problem.

**Problem Definition:** The problem, in this case, is to identify: (a) if the current image of the operational scene is OOD with respect to the training set, and (b) feature(s) likely responsible for the OOD. By this, we mean that if the training set used to train a LEC did not include the scenes with heavy rain, then we want to identify during operation that the OOD is due to precipitation. In addition, as the images received by CPSs are in time series, it is important to identify if the current image has changed with respect to the previous images in the time series. Identifying these changes is referred to as change point detection in literature. Change points in the values of a feature can increase the system's risk, as illustrated in our previous work [117]. So, it is critical to identify these change points during operation. Formally, we summarize the problems as follows: ***Problem 1a*** - identify if the current image is OOD with respect to the training set. ***Problem 1b*** - identify the

feature(s) most likely responsible for the OOD, and ***Problem 2*** - identify if the current image is OOD to the previous images in time series.

**State-of-the-art and Limitations**: Multi-chained one-class classifiers [253] are commonly used for solving the multi-label anomaly detection problem. But the performance of these chains deteriorates in the presence of strong label correlation [254]. Additionally, training one classifier for each label is expensive for real-world datasets with a large label set [255]. Another problem with traditional classifiers like Principal Component Analysis [256], Support Vector Machine [257], and Support Vector Data Description [258] is that they fail in images due to computational scalability [259].

To improve the effectiveness of the classifiers, researchers have started investigating probabilistic classifiers like generative adversarial network (GAN) [260] and variational autoencoder (VAE) [261]. GAN has emerged as the leading paradigm in performing unsupervised and semi-supervised OOD detection [262, 263], but problems of training instability and mode collapse [264, 265] have resulted in VAE-based methods being used instead. In particular, the VAE-based reconstruction approach has become popular for detecting OOD data [94, 92]. However, this approach is less robust in detecting anomalous data that lie on the boundary of the training distribution [95]. To address this, the latent space generated by a VAE is being explored [95, 96, 93]. The latent space is a collection of latent variables ($L$), where each latent variable is a tuple defined by the parameters ($\mu$,$\sigma$) of a latent distribution ($z$) and a sample generated from the distribution. However, the traditional approach of just training a single VAE on all input data leads to unstructured and entangled distributions [266], which makes the task of isolating the feature(s) responsible for OOD hard.

**This paper**: Our approach in this work is to investigate the use of latent space disentanglement for detecting OOD images used by perception LECs. This idea builds upon recent progress in structuring and disentangling the latent space [112, 267]. Effectively, the latent space generated by the encoder of a VAE is a Gaussian mixture model of several overlapping and entangled latent variables, each of which encodes information about the image features. However, as can be seen in Fig. 5.2-a, the latent variables form a single large cluster, making it hard to use them for OOD detection. Disentanglement is a state of the latent space where each latent variable is sensitive to changes in only one feature while being invariant to changes in the others [112]. That is, the single large cluster of Fig. 5.2-a is separated into several smaller clusters of single latent variables if the features are independent. Such disentangled latent variables have been successfully used in several tasks like face recognition [268, 269], video predictions [270], and anomaly detection [271]. However, disentangling all the latent variables is extremely hard for real-world datasets and is shown to be highly dependent on inductive biases [272] and feature correlations.

Nevertheless, even partially disentangling the latent variables can lead to substantial gains, as shown by Jakab *et al.* [273] and Mathieu *et al.* [274]. Partial disentanglement is a heuristic that groups the most

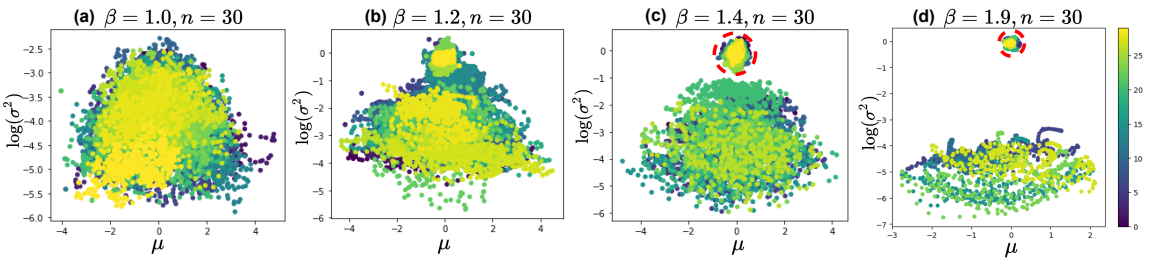



Figure 5.2: Scatter plots illustrating the latent distributions ($\mu$, $\log(\sigma^2)$) generated using a $\beta$-VAE with different $\beta$ values. Each latent distribution in the latent space is represented using distinct color shades. CARLA images generated in Section 4.5 were used to generate these latent distributions. For $\beta$=1, the generated latent distributions are entangled. For $\beta > 1$, the latent distributions are partially disentangled with a few latent distributions (inside the red circle) encoding independent features moving close to $\mu$=0 and $\log(\sigma^2)$=0, and the others moving away. Plot axis: x-axis represents the mean of the latent distributions in the range [-5,5], and the y-axis represents the log of variance in the range [-5,5].

informative latent variables into one cluster and the remaining latent variables that are less informative into another, as shown in Fig. 5.2-c. This selective grouping enables better separation in the latent space for the train and test images as shown in Fig. 5.10 (see Section 4.5). However, the procedure for training a disentangled latent space for real-world CPSs is hard. As a result, it is one of the aspects we focus on in this paper, along with interpreting the source of the OOD.

**Our Contributions**: We present an approach to generate a partially disentangled latent space and learn an approximate mapping between the latent variables and the image features to perform OOD detection and reasoning. The steps in our approach are data partitioning, latent space encoding, latent variable mapping, and runtime anomaly detection. To generate the partially disentangled latent space, we use a $\beta$-VAE, which has a gating parameter $\beta$ that can be tuned to control the information flow between the features and the latent space. For a specific combination of $\beta > 1$ and the number of latent variables ($n$), the latent space gets partially disentangled with a few latent variables encoding most of the feature's information while the others encode little information. We present a Bayesian optimization heuristic to find the appropriate combination of the $\beta$ and $n$ hyperparameters. The heuristic establishes disentanglement as a problem of tuning the two hyperparameters. The selected hyperparameters are used to train a $\beta$-VAE that is used by a latent variable mapping heuristic to select a set of most informative latent variables that are used for detection and identify the sensitivity of the latent variables towards specific features. The sensitivity information is used for reasoning the OOD images. Finally, the trained $\beta$-VAE and the selected latent variables are used at runtime for OOD detection and reasoning. We demonstrate our approach in two CPS case studies: (a) an AV in CARLA simulator [111], and (b) a real-world automotive dataset called nuImages [251].

**Outline**: The outline of this paper is as follows. We formulate the OOD detection problem in Section 5.3. We introduce the background concepts in Section 5.4. We present our OOD detection approach in Section 5.5.

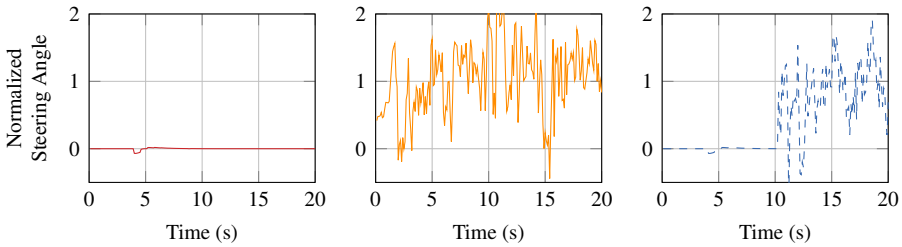



Figure 5.3: Problem illustration. We trained an NVIDIA DAVE-II DNN with images from scene1 and scene2 (Fig. 5.1) to predict the steering values of an AV that travels on a straight road segment in CARLA simulator. We tested the network on three test scenes: (a) A scene that had precipitation and brightness values within the training distribution, (b) A scene with high precipitation (60%) that was not in the training distribution. For this scene, the DNN predictions deviate from the nominal value shown in the plot a, and (c) A scene where brightness value changed from low (20%) to high (50%) at $t = 10$ seconds. The DNN predictions were accurate until $t = 10$ seconds, thereafter the predictions got erroneous.

We discuss our experiment setup and evaluation results in Section 5.6. Finally, we present related research in Section 5.7 followed by conclusions in Section 5.8.

## 5.3 Problem Formulation

To set up the problem, consider a CPS that uses a perception LEC trained on image distribution $\mathscr{T}$, where $\mathscr{T} = \{s_1, s_2, \ldots, s_i\}$ is a collection of scenes. A scene is a collection of sequential images $\{I_1, I_2, \ldots, I_m\}$ generated from a training distribution $P(\mathscr{T})$. Every image in a scene is associated with a set of discrete or continuous-valued labels ($I \rightarrow 2^{\mathbb{L}}$) belonging to the generative features of the environment (see Fig. 5.1). It is important to note that the sampling rate of images depends on the dynamics of the system. With this model, we can define the problems we study as follows:

**Problem 1** *Given a test image $I_t$, determine **(a)** if $I_t \in P(\mathscr{T})$, and **(b)** if $(I_t \notin P(\mathscr{T}))$ then identify the feature $f$ whose Label$(I_t, f) \notin P_f(\mathscr{T})$, where $P_f$ is the training distribution on the feature $f$.*

**Example 1** *To illustrate the problem, we trained an NVIDIA DAVE-II LEC [4] to perform end-to-end driving of an AV in CARLA simulation. We trained the network on camera images from scene1 and scene2 (see Fig. 5.1) to predict the steering control action for the AV. As shown in Fig. 5.3-a, the network's steering predictions were accurate when tested on images from training scenes. However, the predictions got erroneous when we used the network to predict images of a new scene (scene3) with higher precipitation values outside the training distribution (Fig. 5.3-b). The error in steering predictions caused the AV to crash on a sidewall. For this reason, if we know that the precipitation level is compromising the network's predictions, we can switch to an alternative controller that operates on other sensor inputs (e.g., Radar or Lidar) rather than the camera.*

**Problem 2** *Given the test image $I_t$ at time t, the goal is to determine if $I_t$ has changed with respect to the previous images in a time series window $(I[t-M+1],\ldots,I[t])$, where M is the window size.*

**Example 2** *To illustrate the problem, we use the same AV setup discussed in problem1. We tested the trained network on a new scene (scene4) with low brightness (in-distribution) for up to ten seconds, and the brightness was briefly high (OOD) for the next ten seconds. For this scene, the network predicted accurately for the first ten seconds and erroneously for the next ten seconds, as shown in Fig. 5.3-c. Such an abrupt change in the image feature increases the AV's risk of collision, as demonstrated in our recent work [117]. So, identifying the change points can be beneficial for reducing the system's risk of a consequence.*

We evaluate the detector that solves these problems against the following properties.

- **Robustness** - The detector should have low false positives and false negatives. A well known metric to measure robustness is F1-score = $(2 \cdot Precision \cdot Recall) \div (Precision + Recall)$. Precision is computed as $TP \div (TP + FN)$, and Recall is computed as $TP \div (TP + FP)$. Where TP is True positive, FP is False positive, and FN is false negative.
- **Minimum Sensitivity (MS)** - The detector should have a minimum sensitivity towards each feature [275]. For an image $I_t$ with feature labels $\mathbb{L}$=$\{l_1, l_2, \cdots l_k\}$, minimum sensitivity is defined as the minimum value of the detector's sensitivity for each feature label. $MS = min\{Si; i = 1, 2, \cdots, k\}$, where $S_i$ is the sensitivity of the detector to the feature *i*. Recall has been a well known metric to measure the detector's sensitivity.
- **Low Computation Overhead** - Our target platforms are resource-constrained autonomous CPSs like DeepNNCar [80]. Therefore, the detector should have a low resource signature.
- **Low Execution time** - Our target platforms typically have a short sampling period (50 to 100 milliseconds). Therefore, the detector should have an execution time shorter than the system's sampling period.

## 5.4 Background

### 5.4.1 Kullback-Leibler (KL) divergence

KL-divergence [276] is a non-symmetric metric that can measure the similarity between two distributions. For any probability distribution $p$ and $q$, the KL-divergence can be computed as illustrated in Eq. (5.1). A KL-divergence value close to zero indicates the two distributions are similar, while a larger value indicates their dissimilarity.

$$D_{KL}(p||q) = \sum_{x \in \chi} p(x) log \frac{p(x)}{q(x)} \tag{5.1}$$

Recently, the KL-divergence metric has been utilized in several ways: (a) training loss function of different generative models (e.g., VAE) called the evidential lower bound (ELBO) [277] and (b) OOD detection metric

[96] that measures if the latent distributions generated by a generative model deviate from a standard normal distribution. The metric can be computed as:

$$KL(x) = D_{KL}(q_\phi(z_i|x)||\mathcal{N}(0,1)) \tag{5.2}$$

Where, $\mathcal{N}(0,1)$ is a standard normal distribution. $q_\phi(z_i|x)$ is the distribution generated by the encoder of a VAE for a latent variable $L_i$, and input $x$.

### 5.4.2 $\beta$-Variational Autoencoder

$\beta$-VAE [267] is a variant of the original VAE with a $\beta$ hyperparameter attached to the KL-divergence (second term) of the ELBO loss function shown in Eq. (5.3). The network has an encoder that maps the input data ($x$) distribution $P(x)$ to a latent space ($z$) by learning a posterior distribution $q_\phi(z|x)$. The decoder then reconstructs a copy of the input data ($x'$) by sampling the learned distributions of the latent space. In doing so, the decoder also learns a likelihood distribution $p_\theta(x|z)$. The latent space is a collection of $n$ latent variables that needs to be selectively tuned in accordance with the input dataset. To remind, a latent variable is a tuple defined by the parameters ($\mu$,$\sigma$) of a latent distribution ($z$) and a sample generated from the distribution.

$$ELBO(\theta,\phi,\beta;x,z) = \mathbb{E}_{q_\phi(z|x)}[log p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \tag{5.3}$$

Where, $\theta$ and $\phi$ parameterize the latent variables of the encoder and the decoder, and $D_{KL}$ is the KL-divergence metric discussed in Section 5.4.1. The first term computes the similarity between the input data $x$ and the reconstructed data $x'$. The second term computes the KL-divergence between $q_\phi(z|x)$ and a predefined distribution $p_\theta(x|z)$, which is mostly the standard normal distribution $\mathcal{N}(0,1)$.

**Tuning $\beta$ for disentanglement**: As suggested by Higgins *et al.* [267], $\beta$ controls the amount of information that flows from the features to the latent space, and an appropriate hyperparameter combination of $\beta > 1$ and $n$ is shown to disentangle the latent space for independent features. Fig. 5.2 shows the latent space disentanglement for different values of $\beta$ while $n$ is fixed to 30. For $\beta = 1$, the latent distributions are entangled. With $\beta > 1$, the latent space starts to partially disentangle with a few latent variables (inside the red circle) encoding most information about the features stay as a cluster close to $\mu$=0 and log($\sigma^2$)=0, and the others are uninformative and form a cluster that lies farther. However, as the $\beta$ value gets larger ($\beta$=1.9), the information flow gets so stringent that the latent space becomes uninformative [274]. So, finding an appropriate combination of $\beta$ and $n$ for disentanglement is a hard problem. We provide a heuristic in Section 5.5.2 to address this problem.

**Algorithm 3** Computing MIG

**Parameter**: number of latent variables $n$, image features $\mathscr{F}$, number of iterations $t$, trained $\beta$-VAE
**Input**: data partition $\mathscr{P} = \{P_1, P_2, ...., P_m\}$
**Output**: average MIG

**while** $i \leq t$ **do**
  Generate latent variable parameters ($\mu$, $\sigma$) using a trained $\beta$-VAE
  **for each** $P \in \mathscr{P}$ **do**
    Extract $\mu$, $\sigma$ parameters
    Compute feature entropy H(f)
    Compute latent variable entropies $H(L_j)$ and $H(L'_j)$
    Compute conditional entropies $H(L_j|f)$ and $H(L'_j|f)$
    Compute mutual information of most informative latent variable $I_1(L_j;f) = H(L_j) - H(L_j|f)$
    Compute mutual information of second most informative latent variable $I_2(L'_j;f) = H(L'_j) - H(L'_j|f)$
  **end for**
  compute $MIG = \frac{1}{|\mathscr{P}|} \sum_{f \in \mathscr{P}} \frac{1}{H(f)} (I_1(L_j;f) - I_2(L'_j;f))$
  Append MIG to $L$
**end while**
**return** average MIG = sum($L$)/$t$

### 5.4.3 Mutual Information Gap

Mutual Information Gap (MIG) is a metric proposed by Chen, Ricky TQ *et al.* [278] to measure the latent space disentanglement. It is an information-theoretic metric that measures the mutual information between features and latent variables. It measures the average difference between the empirical mutual information of the two most informative latent variables for each feature and normalizes this result by the entropy of the feature. MIG is computed using the following equation.

$$MIG = \frac{1}{|\mathscr{F}|} \sum_{f \in \mathscr{F}} \frac{1}{H(f)} (I_1(L_j;f) - I_2(L'_j;f)) \tag{5.4}$$

Where, $I_1(L_j;f)$ represents the empirical mutual information between the most informative latent variable $L_j$ and the feature $f$. $I_2(L'_j;f)$ represents the empirical mutual information between the second most informative latent variable $L'_j$ and the feature $f$. $H(f)$ is the entropy of information about the feature $f$. In this work, we use MIG as a measure for selecting the right hyperparameter combination ($\beta$ and $n$) for the $\beta$-VAE.

**Implementation**: We have implemented Algorithm 3 to compute MIG. Since MIG is computed based on entropy, the training set $\mathscr{T}$ should have images with feature labels that take different discrete and continuous values, as shown in Fig. 5.1. For the ease of computation, we partition $\mathscr{T}$ into different partitions $\mathscr{P}$ using our approach discussed in Section 5.5.1. Our approach is to create partitions such that each partition will have images that have a variance in the value of a specific feature $f$, irrespective of the variance in the others. The feature with the highest variance represents the partition. Then, in each iteration, we use the feature representing the partition to compute the feature entropy, the conditional entropy, and the mutual information of the two most informative latent variables. The mutual information is then used to compute the MIG, as

shown in the algorithm. Finally, for robustness, we average the MIG across $t$ iterations.

**Complexity Analysis**: The complexity of the MIG algorithm in worst case is $O(t * |\mathscr{P}| * nq * |\mathscr{F}| * |\mathscr{T}| * n^2 * ns)$. Where $t$ is the number of iterations, $|\mathscr{P}|$ is the number of partitions, $n$ is the number of latent variables, $|\mathscr{F}|$ is the number of features considered for the calculations, $nq$ is the number of unique values that each feature has in the partition (e.g., for a partition with brightness=10%, and brightness=20%, $nq$=2), $ns$ is the number of samples generated from each latent variable, and $|\mathscr{T}|$ is the size of training data. The specific values of these parameters for the AV example in CARLA simulation are $t = 5$, $|p| = 2$, $n = 30$, $\mathscr{F} = 2$, $ns = 500$, and $nq$=3.

### 5.4.4 Inductive Conformal Prediction

Inductive Conformal Prediction (ICP) is a variant of the Conformal Prediction algorithm [279] that tests if a test observation ($x_t$) conforms to every observation in the training dataset ($\mathscr{T}$). However, comparing $x_t$ to every observation of $\mathscr{T}$ is expensive and gets complex with the size of $\mathscr{T}$. To address this, ICP splits $\mathscr{T}$ into two non-overlapping sets called the proper training set ($\mathscr{T}_P$), which is used to train the prediction algorithm (e.g., LEC) and the calibration set ($\mathscr{C}$) which is used to calibrate the test observations. In splitting the datasets, ICP performs a comparison of the $x_t$ to each element in the $\mathscr{C}$ which is a smaller representative set of $\mathscr{T}$. The ICP algorithm has two steps: The first step involves computing the non-conformity measure, which represents the dissimilarity between $x_t$ to the elements in the set $\mathscr{C}$. The non-conformity measure is usually computed using conventional metrics like euclidean distance or K-nearest neighbors. But, in this work, we use KL-divergence as the non-conformity measure. The next step involves computing the p-value, which serves as evidence for the hypothesis that $x_t$ conforms to $\mathscr{C}$. Mathematically, the p-value is computed as the fraction of the observations in $\mathscr{C}$ that have non-conformity measure above the test observation $x_t$: $p_{x_t} = |\{\forall \alpha \in \mathscr{C} | \alpha \geq \alpha_{x_t}\}| / |\mathscr{C}|$. Note for brevity, we drop the notation of $x$ and use the term $p_t$ and $\alpha_t$. Here $\alpha$ denotes the non-conformity measure for each observation in $\mathscr{C}$ and $\alpha_t$ denotes the non-conformity measure for $x_t$.

Once $p_t$ is computed, it can be compared against a threshold $\tau \in (0,1)$ to confirm if $x_t$ belongs to $\mathscr{T}$. However, such a threshold-based comparison is only valid if each test observation is i.i.d (independent and identically distributed) to $\mathscr{T}$, which is not true for CPS [92]. Although the assumptions about i.i.d are not valid, ICP can still be applied under the weaker assumption of exchangeability. In our context, exchangeability means testing whether the observations in $\mathscr{C}$ have the same joint probability distribution as the sequence of the test observations under consideration (are they permutations of each other). If they are, then we can expect the p-values to be independent and uniformly distributed in [0, 1] (Theorem 8.2, [279]), which can be tested using the martingale. Exchangeability martingale [280] has been used as a popular tool for testing the exchangeability

and the IID assumptions of the test observation with respect to $\mathscr{C}$. So, once the p-value for a test observation is computed, the simple mixture martingale [280] can be computed as $\mathscr{M}_t = \int_0^1 \left[\prod_{i=1}^{t} \varepsilon p_i^{\varepsilon-1}\right] d\varepsilon$.

Also, it is desirable to use a sequence of test observations rather than a single observation for improving the detection robustness. However, the observations received by CPS are in time series, making them non-exchangeable [92]. The non-exchangeability hinders the direct application of the martingale to an infinitely long sequence of test observations. To address this, the authors in [92] have suggested applying the martingale over a short time series window in which the test observations can be assumed to be exchangeable. Then, the simple mixture martingale over a short time window $[t-M+1,t]$ of past $M$ p-values can be computed as $\mathscr{M}_t = \int_0^1 \left[\prod_{i=t-M+1}^{t} \varepsilon p_i^{\varepsilon-1}\right] d\varepsilon$. The martingale will grow over time if there are consistently low p-values within the time window, and the corresponding test observations are i.i.d. Otherwise, the martingale will not grow. It is important to note that the martingale computation is only valid for a short time. The size of the window is dependent on the CPS dynamics, like the speed of the system. The system's speed was constant in our experiments, so we used a fixed window size of 20 images.

### 5.4.5 Cumulative Sum

One of the problems we are dealing with is change detection, which is typically solved using cumulative sum (CUSUM) [281], a statistical quality control procedure used to identify variation based on historical data. It is computed as $S_0 = 0$ and $S_{t+1} = \max(0, S_t + x_t - \omega)$, where $x_t$ is the sample from a process, $\omega$ is the weight assigned to prevent $S_t$ from consistently increasing to a large value. $S_t$ can be compared to a predefined threshold $\tau$ to perform the detection. $\omega$ and $\tau$ are hyperparameters that decide the detector's precision.

## 5.5 Our Approach

In this section, we present our approach that uses a $\beta$-VAE to perform latent space-based OOD detection and reasoning. Our workflow is shown in Fig. 5.4 and the steps involved work as follows: **First**, we divide the multi-labeled datasets into partitions based on the variance in the feature values. A partition consists of images with one feature having a higher variance in its values than others. **Second**, we generate a partially disentangled latent space using a $\beta$-VAE. As discussed earlier, the combination of $\beta$ and $n$ influences the level of latent space interpretability. To find the optimal combination, we propose a heuristic that uses the Bayesian optimization algorithm [282] along with MIG to measure disentanglement. **Third**, we discuss a heuristic to perform latent variable mapping to identify the set of latent variables $\mathscr{L}_d$ that encodes most information of the image features. These latent variables are collectively used as the detector for OOD detection. Further, we perform a KL-divergence based sensitivity analysis to identify the latent variable(s) $\mathscr{L}_f \subseteq \mathscr{L}_d$, that is sensitive

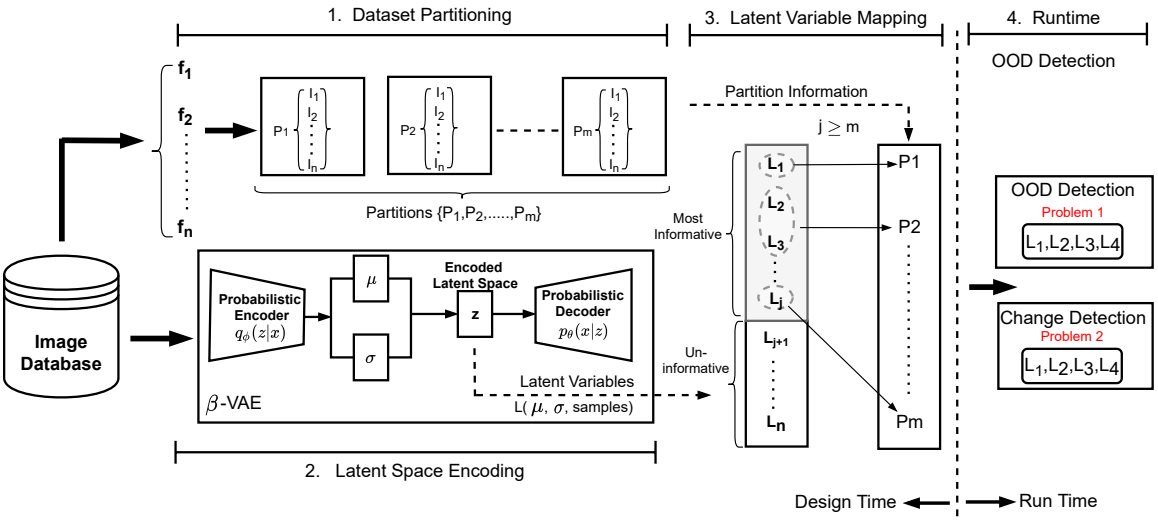



Figure 5.4: The steps of our workflow include data partitioning, latent space encoding, latent variable mapping, and runtime OOD detection.

to specific features. The latent variable(s) in $\mathscr{L}_f$ are used as reasoners to identify the OOD causing features. We discuss these steps in the rest of this section.

### 5.5.1 Data partitioning

Data partitioning is one of the core steps of our approach. It is required for implementing the MIG and the latent variable mapping heuristics. We define a partition $P$ as a collection of images with a higher variance in the value of one feature compared to the variances in the values of other features. To explain the concept of a partition, consider the features of images in training set $\mathscr{T}$ to be $\mathscr{F} = \{f_1, f_2, ..., f_n\}$, and each feature can take a value either discrete or continuous as shown in Fig. 5.1. We normalize these values for the ease of partitioning. Our goal is to group images in $\mathscr{T}$ into $m$ partitions $\mathscr{P} = \{P_1, P_2, ...., P_m\}$, such that a partition $P_j$ will have all the images with high variance in the values of feature $f_j$. For creating these partitions, we generate sub-clusters for each discrete-valued feature(s) and sub-clusters for each continuous-valued feature. In each of these sub-clusters, only the value of the feature under consideration $f_j$ changes while the value of other features remains unchanged. The clustering is done effectively through an agglomerative clustering algorithm. Thereafter, the partition for a feature is the union of all sub-clusters which can be represented as $P_j = \{C_1 \cup C_2 .... \cup C_n\}$. It is important to note that each partition should have $variance > 0$ in the value of the feature under consideration ($f_j$), irrespective of changes in the values of other features. To illustrate the partitioning concept, consider the example scenes in Fig. 5.1. A precipitation partition for this example is a collection of images from two combinations, $C1$ = {day, precipitation=22%, brightness=0%, R1}, and $C2$ = {day, precipitation=50%, brightness=50%, R1}.

**Algorithm 4** Bayesian Optimization Hyperparameter Selection

**Parameter**: number of iterations $t$, initialization iterations $k$, explored list $\mathscr{X}$
**Input**: training set $\mathscr{T}$, data partition $\mathscr{P}$, $N$, $B$
**Output**: best $n$ and $\beta$

**for** $x = 1, 2, ..., t$ **do**
  **if** $x \leq k$ **then**
    Randomly sample $n$, $\beta$ from $N$ and $B$
  **else**
    Find $n$, $\beta$ that optimizes the acquisition function over Gaussian Process
  **end if**
  Train $\beta$-VAE on $\mathscr{T}$ using the selected $n$ and $\beta$
  Compute Average MIG
  Append $n$, $\beta$ and MIG to $\mathscr{X}$
  Update the Gaussian Process posterior distribution using $\mathscr{X}$
**end for**
**return** $n$ and $\beta$

Our data partitioning approach works well for datasets with well-defined labels that provide feature value(s). However, real-world automotive datasets such as nuScenes [252] and nuImages [251] provide semantic labels that are not always well defined, and they do not contain feature values. Also, they often have images in which several feature values change simultaneously. Since the feature-related information is not fully available, some prepossessing and threshold selection for feature values are required for partitioning. Currently, the threshold selection for partitioning is performed by a human supervisor, but we want to automate it in the future. We have applied this partitioning technique to the nuScenes dataset in our previous work [93] and used it in this work for partitioning the nuImages dataset.

### 5.5.2 Latent Space Encoding

The second step of our design procedure is selecting and training $\beta$-VAE to generate a partially disentangled latent space encoding. However, the challenge is determining the best combination of the $\beta$ and $n$ hyperparameters. To find this, we propose a novel greedy heuristic that formulates disentanglement as a hyperparameter search problem. The heuristic uses a Bayesian optimization algorithm with MIG as the objective function to maximize.

**Implementation**: The Bayesian optimization algorithm builds a probability model of the objective function and uses it to identify the optimal model hyperparameter(s). The algorithm has two steps. First, a probabilistic Gaussian Process model is fitted across all the hyperparameters explored thus far. Next, an acquisition function to determine which hyperparameter point to evaluate next [282]. We use these steps to search for an optimal hyperparameter for n $\in N$, and $\beta \in B$. The heuristic using the Bayesian optimization algorithm is shown in Algorithm 4, and the steps are discussed below.

First, for $k$ initial iterations, we randomly pick values for $\beta$ and $n$ from the hyperparameter search space. The randomly selected n and $\beta$ combination is used to train a $\beta$-VAE network and compute the MIG as discussed in Algorithm 3. The selected hyperparameters and the computed MIG are added to an explored list $\mathscr{X}$. After the initial iterations, the trained Gaussian Process model (the initial iterations are used to train and stabilize the Gaussian process model) is fitted across all the hyperparameter points that were previously explored. The marginalization property of the Gaussian distribution then allows the calculation of a new posterior distribution g($x_n$) with posterior belief $\tilde{g}(x_n)$. Finally, the parameters ($\mu$, $\sigma$) of the resulting distribution are determined to be used by the acquisition function.

Second, an acquisition function is used to guide the search by selecting the hyperparameter(s) for the next iteration. For this, it uses the $\mu$ and $\sigma$ computed by the Gaussian process. A commonly used function is the expected improvement (EI) [283, 284], which can be described as follows. Consider, $x_n$ to be some hyperparameter(s) point in the distribution g($x_n$) with posterior belief $\tilde{g}(x_n)$, and $x_+$ is the best hyperparameter(s) in $\mathscr{X}$ (explored list), then the improvement of the point $x_n$ is computed against $x_+$ as $I(x_n) = max\{0, (g(x_+) - g(x_n))\}$. Then, the expected improvement is computed as $EI(x_n) = \mathbb{E}\big[I(x_n)|x_n\big]$. Finally, the new hyperparameter(s) is computed as the point with the largest expected improvement as $x_{n+1} = argmax(EI(x_n))$. The new hyperparameter point is used to train a $\beta$-VAE and compute the MIG, which is then added to the explored list $\mathscr{X}$. The list $\mathscr{X}$ is used to update the posterior distribution of the Gaussian process model in the next iteration. The two steps of the algorithm are iterated until the maximum number of iterations is reached or can be terminated early if optimal hyperparameter(s) is consecutively selected by the algorithm for $j$ iterations.

**Design Space Complexity**: Searching for optimal hyperparameters is a combinatorial problem that requires optimizing an objective function over a combination of hyperparameters. In our context, the $\beta$-VAE hyperparameters to be selected are $\beta$ and $n$, and the objective function to optimize is the MIG, whose complexity we have reported in the previous section. The range of the hyperparameters are n $\in \{n_1, n_2, \ldots, n_l\}$ and $\beta \in \{\beta_1, \beta_2, \ldots, \beta_m\}$. The hyperparameter search space then becomes the Cartesian product of the two sets. In the case of the grid search, each point of this search space is explored, which requires training the $\beta$-VAE and computing MIG. Grid search suffers from the curse of dimensionality since the number of evaluations exponentially grows with the size of the search space [285]. In comparison, the random search and Bayesian optimization algorithms do not search the entire space but search for a selected number of iterations $t$. While random search selects each point in the search space randomly, the Bayesian optimization algorithm performs a guided search using a Gaussian process model and the acquisition function.

In the Bayesian optimization algorithm, the first $k$ iterations stabilize the Gaussian Process model using randomly selected points in the space to train a $\beta$-VAE and compute the MIG. After these iterations, the Gaussian process model is fitted to the previously sampled hyperparameters. Then, the acquisition function

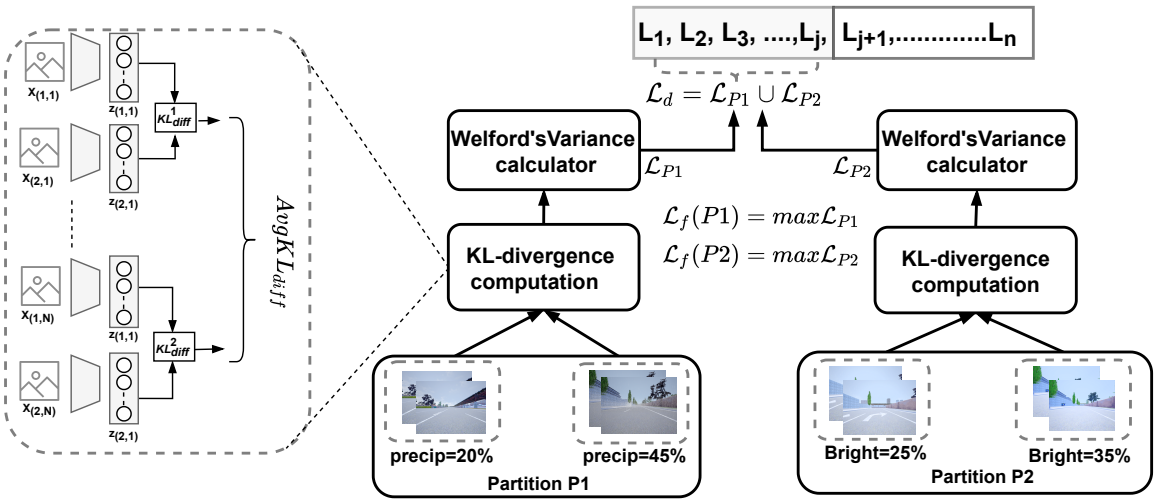



Figure 5.5: Heuristic-based on KL-divergence and Welford's variance calculator to select latent variables for $\mathscr{L}_d$ and $\mathscr{L}_f$.

based on expected improvement uses the posterior distribution of the Gaussian process model to find new hyperparameter(s) that may optimize the MIG. The new hyperparameter(s) are used to train a $\beta$-VAE and compute the MIG. So, this process is repeated for $t$ iterations, and in every iteration, a $\beta$-VAE is trained, and the MIG is computed as shown in Algorithm 4. This intelligent search mechanism based on prior information makes the search technique efficient as it takes fewer points to explore in the search space compared to both grid search and random search [282, 285]. The Bayesian optimization algorithm has a polynomial time complexity because the most time-consuming operation is the Gaussian process which takes polynomial time [285]. We report the experimental results for the three hyperparameter algorithms in Table 5.1 (see Section 5.6). As it will be seen from the table, the Bayesian optimization algorithm takes the least time and iterations to select the hyperparameters that achieve the best MIG value as compared to the other algorithms.

### 5.5.3 Latent Variable Mapping

Given the data partition set $\mathscr{P}$ and a trained $\beta$-VAE that can generate the latent variable set $\mathscr{L}$, we find the most informative latent variables set $\mathscr{L}_d \subseteq \mathscr{L}$ that encodes information about the image features in the training set $\mathscr{T}$. As discussed earlier, the most informative latent variables form a separate cluster from the less informative ones when we partially disentangle the latent space using the Bayesian optimization heuristic. Although the most informative latent variables are separately clustered in the latent space, we need a mechanism to identify them. To do this, we present a KL-divergence based heuristic in Fig. 5.5.

The latent variables in $\mathscr{L}_d$ are used as detectors, i.e., they can detect overall distribution shifts in the images (we discuss the exact procedure later). However, this does not solve the problem of identifying the

specific feature(s) whose labels caused the OOD. For this, given the representative feature $f$ of a partition has high variance, we identify the subset of latent variable(s) $\mathscr{L}_f \subseteq \mathscr{L}_d$ that is sensitive to the variations in $f$. For example, if we have a partition with images that have different values in the precipitation (e.g., precipitation=20%, precipitation=50%, etc.), then we map latent variable(s) that are sensitive to changes in precipitation level. The latent variable(s) in $\mathscr{L}_f$ is called the reasoner for feature $f$ and is used to identify if $f$ is responsible for the OOD. The steps in our heuristic are listed in Algorithm 5, and it works as follows: For each partition, $P \in \mathscr{P}$, and for each scene $s \in P$, we perform the following steps.

First, we take two subsequent images $x_l$ and $x_{l+1}$ and pass each of them separately to the trained $\beta$-VAE to generate the latent variable set $\mathscr{L}$ for each of the images. As a remainder, $\mathscr{L}$ is a collection of $n$ latent variables, and each latent variable has a latent distribution ($z$) with parameters $\mu$ and $\sigma$. Then, for each latent variable of the images ($x$) we compute a KL-divergence between its latent distribution $q(z_i|x)$ and the standard normal distribution $\mathscr{N}(0,1)$ as discussed in Section 5.4.1. The computed KL-divergence is $KL^i(x) = D_{KL}(q(z_i|x)||\mathscr{N}(0,1))$.

Second, we calculate the KL-divergence difference between corresponding latent variables of the two images as: $KL^i_l(diff) = |KL^i(x_{l+1}) - KL^i(x_l)|$. This procedure is repeated across all the subsequent images in the scene $s$. Third, we compute an average KL-divergence difference for each latent variable across all the subsequent scene images as follows.

$$AvgKL^i_{diff} = \frac{1}{len(s)-1} \sum_{l=1}^{len(s)-1} KL^i_l(diff) \tag{5.5}$$

Where, $KL^i_l(diff)$ is the KL-divergence difference of the latent variable $L_i$ for the $l^{th}$ subsequent image pair in a scene $s$ and $len(s)$ is the number of images in $s$. This value indicates the average variance in the KL-divergence value across each latent variable for all the images in the $s$. This approach to computing the variations across $s$ is motivated by the manual latent variable mapping technique in [267]. Further, the $AvgKL^i_{diff}$ value is computed $\forall s \in P$.

Finally, we then use Welford's variance calculator [286] to compute the variance in the $AvgKL^i_{diff}$ value across all the scenes in the partition. Welford's variance calculator computes and updates the variance in a single pass as the measurements are available. It does not require storing the measurements till the end of the variance calculation, which will make the variance calculation across several scenes faster. In our case, the variance calculator returns a partition latent variable set $\mathscr{L}_P$, which is a set of top $m$ latent variables that has the highest $AvgKL^i_{diff}$ across all images in $P$. Selecting an appropriate number of latent variables ($m$) for $\mathscr{L}_P$ is crucial, as we use it to select the latent variables for $\mathscr{L}_d$ and $\mathscr{L}_f$. The value for $m$ is chosen empirically, and the selection depends on the variances across the scenes of a partition. However, it is essential to note that selecting a small value for $m$ may not include all the informative latent variables required for detection

**Algorithm 5** Selecting Latent Variables for $\mathscr{L}_d$ and $\mathscr{L}_f$

**Parameter**: global list $G$
**Input**: data partitions $\mathscr{P} = \{P_1, P_2, ..., P_m\}$, number of latent variables $n$
**Output**: $\mathscr{L}_d$ and $\mathscr{L}_f$ for each partition

**for each** $P \in \mathscr{P}$ **do**
  **for each** $s \in P$ **do**
    **for** $l = 1;\ l <= len(s);\ l = l + 1$ **do**
      **for** $i = 0, 1, 2, ....., n$ **do**
        $KL^i(x_l) = D_{KL}(q_\phi(z_i|x_l)||\mathscr{N}(0,1))$
        $KL^i(x_{l+1}) = D_{KL}(q_\phi(z_i|x_{l+1})||\mathscr{N}(0,1))$
        $KL^i_l(diff) = |KL^i(x_{l+1}) - KL^i(x_l)|$
      **end for**
    **end for**
    **for** $i = 1, 2, ...., n$ **do**
      $AvgKL^i_{diff} = \frac{1}{len(s)-1} \sum_{l=1}^{len(s)-1} KL^i_l(diff)$
    **end for**
    Store $AvgKL^i_{diff}$ to $G$
  **end for**
  Use Welford's algorithm to select $\mathscr{L}_P$, a set of $m$ latent variables with high variance in $AvgKL^i_{diff}$
  $\mathscr{L}_f$ = max($\mathscr{L}_P$)
**end for**
$\mathscr{L}_d = \{\mathscr{L}_{P_1} \cup \mathscr{L}_{P_2} \cup ..... \cup \mathscr{L}_{P_m}\}$
**return** $\mathscr{L}_d$, $\{\mathscr{L}_{f_1}, \mathscr{L}_{f_2}, ..., \mathscr{L}_{f_m}\}$

and choosing a large value for $m$ may include uninformative latent variables that may reduce the detection accuracy and sensitivity.

Further, we choose the top latent variable(s) from $\mathscr{L}_P$ and use it as the reasoner ($\mathscr{L}_f$) for the partition. If the partition has variance in an independent image feature (e.g., brightness), then a single best latent variable with the most sensitivity can be used as the reasoner. However, if the feature is not independent, more than one latent variable needs to be used, and the size of $\mathscr{L}_P$ increases. Besides, if two features correlate, then a single latent variable may be sensitive to both features. If such a latent variable is used for reasoning at runtime, and if it shows variation, we attribute both the features to be responsible for the OOD.

Finally, these steps are repeated for all the partitions in $\mathscr{P}$, and the latent variables for $\mathscr{L}_d$ is formed by $\{\mathscr{L}_{P_1} \cup \mathscr{L}_{P_2} \cup \cdots \cup \mathscr{L}_{P_m}\}$. If the latent space is partially disentangled, then the top $m$ latent variables in each $\mathscr{L}_P$ will mostly be the same. Otherwise, the number of similar latent variables in each $\mathscr{L}_P$ will be small.

### 5.5.4 Runtime OOD Detection

At runtime, we use the trained $\beta$-VAE and the latent variable set $\mathscr{L}_d$ and $\mathscr{L}_f$ to detect OOD problems discussed in Section 5.3. Fig. 5.6 shows the pipeline for runtime OOD detection, and it works as follows. As a test image, $x_t$ is observed, and the encoder of the trained $\beta$-VAE is used to generate the latent space encoding. Then, the respective latent variables in ($\mathscr{L}_d$, $\mathscr{L}_f$) are sent to different processes to compute the

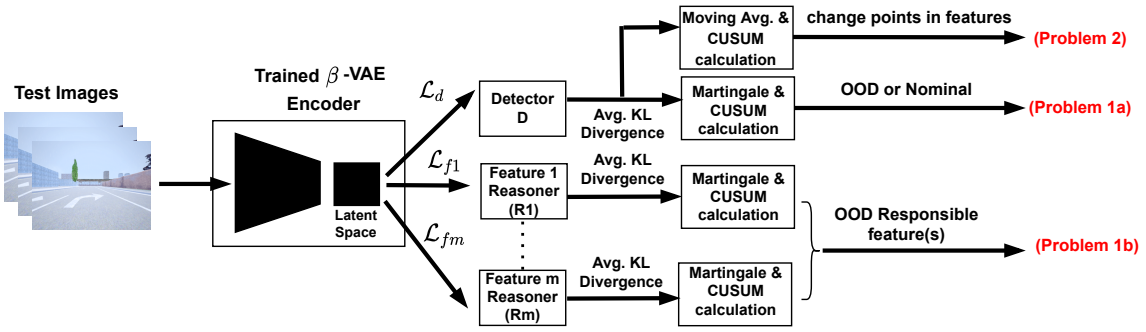



Figure 5.6: The trained $\beta$-VAE detector at runtime provides three outputs that are sent to the decision manager to select an appropriate controller or control action that can mitigate the OOD problems as discussed in Section 5.3.

average KL-divergence between the latent variables in $\mathscr{L}_d$ or latent variable(s) in $\mathscr{L}_f$ for the identified features. The KL-divergence is computed between each latent variables in $\mathscr{L}_d$ and $\mathscr{L}_f$, and the normal distribution $\mathscr{N}(0,1)$ as shown in Eq. (5.6).

$$\alpha_t = KL(x_t, \mathscr{N}(0,1)) = \frac{1}{L}\sum_{l=1}^{L} |D_{KL}(q(z_l|x_t)||\mathscr{N}(0,1))| \tag{5.6}$$

Where, $L$ is the number of selected latent distributions; for the detector, it is the number of latent variables in $\mathscr{L}_d$, and for each feature, it is the number of latent variables in $\mathscr{L}_f$.

To detect the first OOD problem (*Problem 1*), the KL-divergence is used as the non-conformity score to compute the ICP and martingale score as shown in Algorithm 6. However, as the martingale can grow large very rapidly, we use the log of martingale. Then, a CUSUM over the log of martingale is computed to identify when the martingale goes consistently high. The CUSUM value $S_t$ of the detector latent variables $\mathscr{L}_d$ is compared against a detector threshold ($\tau_d$) to detect if the image $x_t$ is OOD compared to the calibration set. Then, the CUSUM value $S_t$ of the reasoner latent variables $\mathscr{L}_f$ is compared against a reasoner threshold ($\tau_r$) to identify if the known feature(s) is responsible for the OOD as discussed in *Problem 1b* of Section 5.3. The thresholds are empirically tuned as a tradeoff between false positives and mean detection delay [281].

To detect the second OOD problem (*Problem 2*), the latent variables in $\mathscr{L}_d$ are used with sliding window moving average and CUSUM for change point detection. For this, the average KL-divergence of all the latent variables in $\mathscr{L}_d$ is computed using Eq. (5.6), and a moving average of the average KL-divergence ($A_{KL}$) is computed over a sliding window $[x_{t-M+1},\ldots,x_t]$ of previous $M$ images in the time series. $A_{KL}$ is used to compute the CUSUM value $S_t$, which is compared against a threshold $\tau_{cp}$ to detect changes.

Finally, the outputs of the detector (See Fig. 5.6) are sent to the decision manager to perform controller selection using the simplex strategy [61]. We use the detection result as the logic to arbitrate between the performant controller (LEC) and the safety controller (autopilot) of the simplex architecture (see Fig. 5.7).

**Algorithm 6** $\beta$-VAE based OOD detection using ICP

**Parameter**: sliding window length $M$, non-conformity measures of calibration set $\mathscr{C}$.
**Input**: image $x_t$ at time $t$, set of detector latent variables $\mathscr{L}_d$.
**Output**: martingale score $\log \mathscr{M}_t$ at time $t$

1: $\alpha_t = \sum_{\forall l \in \mathscr{L}_d} |D_{KL}(q(z_l|x_t)||\mathscr{N}(0,1))|$
2: $p_t = \frac{(|\{\forall \alpha \in \mathscr{C} | \alpha \geq \alpha_t\}|)}{(|\mathscr{C}|)}$
3: $M_t = \int_0^1 \left[\prod_{i=t-M+1}^{t} \varepsilon p_i^{\varepsilon-1}\right] d\varepsilon$
4: **return** $\log \mathscr{M}_t$

## 5.6 Experiments and Results

We evaluate our approach using an AV example in the CARLA simulator [111] and show the preliminary results from the real-world nuImages dataset [251]. The experiments[3] in this section were performed on a desktop with AMD Ryzen Threadripper 16-Core Processor, 4 NVIDIA Titan Xp GPU's and 128 GiB memory.

### 5.6.1 System Overview

Our first example system is an AV that must navigate different road segments in town1 of the CARLA simulator. The architecture of our AV, shown in Fig. 5.7, relies on a forward-looking camera for perception and a speedometer for measuring the system's speed. It uses the NVIDIA DAVE-II deep neural network (DNN) [4] as the primary controller and the simulator's inbuilt autopilot mode as the secondary safety controller. In addition, a trained $\beta$-VAE detector and reasoner are used in parallel to the two controllers. The detection results and the steering values from the two controllers are sent to a simplex decision manager, which selects the appropriate steering value for the system based on the detection result. If the detector returns the image to be OOD, the decision manager selects the autopilot controller to drive the AV. The sampling period used in the simulation is 1/13 seconds, and the vehicle moves at a constant speed of 0.5 m/s for all our experiments.

#### 5.6.1.1 Operating modes

Our AV has two operating modes: (a) *manual driving mode*, which uses CARLA's autopilot controller to drive around town1. The autopilot controller is not an ML component but uses hard-coded information from the simulator for safe navigation. We use this mode to collect the training set $\mathscr{T}$ and the test set ($\mathscr{T}_t$). These datasets are a collection of several CARLA scenes generated by a custom scenario description language (SDL) shown in Fig. 5.8; and (b) *autonomous mode*, which uses a trained NVIDIA DAVE-II LEC controller to drive the AV. In this setup, the $\beta$-VAE detector is used in parallel to the LEC controller to perform OOD detection.

[3] source code to replicate these experiments can be found at https://github.com/scope-lab-vu/Beta-VAE-OOD-Detector

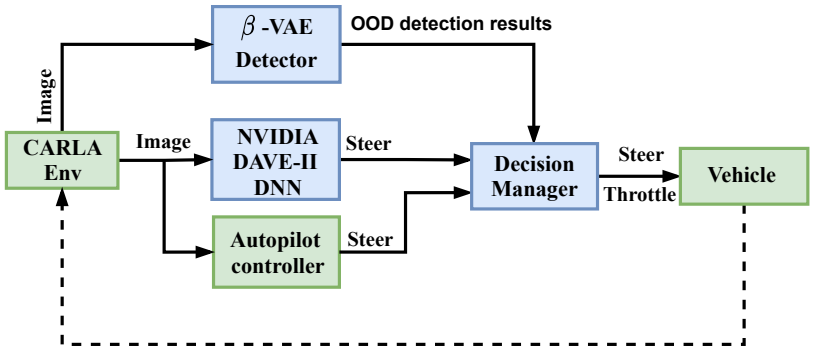



Figure 5.7: AV block diagram. The components in green come inbuilt with the CARLA simulator and the components in blue are designed for our example. While the simulator requires a GPU, the other components are run on one CPU core to emulate a resource-constrained setting.

```
scene CARLA {
  type int
  type distribution
  entity weather_parameters{
    sun_angle: int
    cloudiness: distribution
    precipitation: distribution }
  entity other_parameters{
    road_segment:int
    brightness: distribution }
    }
```

| Feature | Training Range | Nom | HP | HB | HPB | NR |
|---|---|---|---|---|---|---|
| **Sun Angle** | 90 | 90 | 90 | 90 | 90 | 90 |
| **Cloudiness** | [0,50] | 25 | 50 | 50 | 25 | 50 |
| **Precipitation** | [0,50] | 0 | **40** | **0** | **50** | **0** |
| | | | 60 | 0 | 75 | 0 |
| **Brightness** | [0,50] | 0 | **0** | **25** | **40** | **0** |
| | | | 0 | 60 | 60 | 0 |
| **Road** | 1,2,3 | 3 | 2 | 2 | 2 | 7 |

Figure 5.8: (left) A fragment of our SDL. (right) Scenes in CARLA simulation: Nominal scene (*Nom*) is generated by randomly sampling the features in their training ranges. High Precipitation (*HP*), High Brightness (*HB*), High Precipitation & Brightness (*HPB*) scene, and New Road (*NR*) are test scenes for which the feature values change from a training range value to a value outside the range at $t = 20$ seconds. The initial values of precipitation and brightness are highlighted in green.

#### 5.6.1.2 Data Generation

Domain-specific SDL such as Scenic [287] and MSDL [153] are available for probabilistic scene generation. However, they did not fit our need to generate partition variations. Hence, we have implemented a simple SDL in the textX [288] meta language (See Fig. 5.8), which is combined with a random sampler over the range of the simulator's features like sun altitude, cloudiness, precipitation, brightness, and road segments to generate different scenes.

**Train Scenes**: The training feature labels, and their values are shown in Fig. 5.8. These features were randomly sampled in the ranges shown in Fig. 5.8 to generate eight scenes of 750 images each that constituted the training set $\mathscr{T}$. Among these, two scenes had precipitation of 0%, the brightness of 0%, sun angle of 90°, cloudiness of 25%, and road segment of 1 and 2 for each scene, respectively. Three scenes had different precipitation values (precip=5%, precip=40%, precip=50%) while sun angle took a value of 90°, cloudiness took a value of 25%, brightness took a value of 0%, and 10%, and the road segments took a value of 1 and 2. The remaining three scenes had different brightness values (bright=9%, bright=25%, bright=40%),

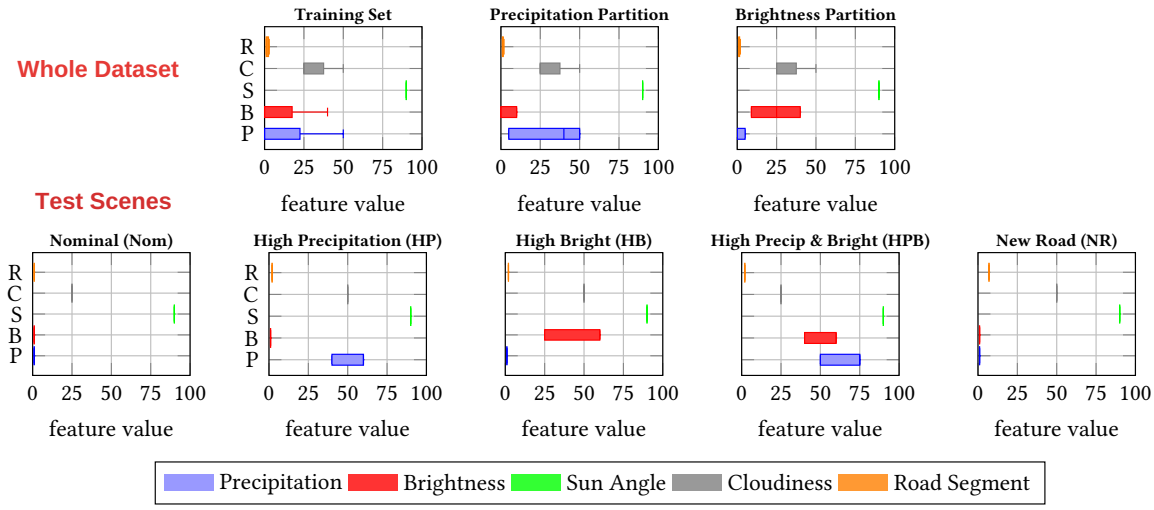



Figure 5.9: Plots representing the normalized values of sun angle (S), cloudiness (C), precipitation (P), brightness (B), and road segments (R) for the training set, partitions, and the test scenes. We use the test scene abbreviations *Nom* (Nominal), *HP* (High Precipitation), *HB* (High Brightness), *HPB* (High Precipitation & Brightness), and *NR* (New Road segment) for the rest of the paper. The *NR* scene has all the feature values like the *Nom* scene, but it uses the road segment 7 that is not in the training distribution.

precipitation took a value of 0% and 5%, and all the other parameters remained the same as the other scenes. We split 6000 images of $\mathscr{T}$ into 4000 images of $\mathscr{T}_P$ and 2000 images of $\mathscr{C}$ in the standard 2:1 ratio (page 222 of [289]) for ICP calculations.

**Test Scenes**: The test scenes included a nominal scene (*Nom*) and four OOD scenes as shown in Fig. 5.8. Each scene was 20 seconds long and had 260 images. (1) Nominal Scene (*Nom*) was an in-distribution scene generated from the training distribution. (2) High Precipitation (*HP*) scene was an OOD scene in which the precipitation was increased from 40% (in training range) to 60% (out of training range) at $t = 2$ seconds. (3) High Brightness (*HB*) scene was an OOD scene in which the brightness was increased from 25% (in training range) to 60% (out of training range) at $t = 2$ seconds. (4) High Precipitation & Brightness (*HPB*) scene was also an OOD scene in which both the precipitation and the brightness were increased out of the training range at $t = 2$ seconds. (5) New Road (*NR*) scene was an OOD scene with a new road segment (segment=7) that was not in the training range, but the other features remained within the training range. Fig. 5.9 shows the variance in the normalized values of the scene features for the training set, partitions, and test scenes.

### 5.6.2 $\beta$-VAE Detector

#### 5.6.2.1 Data Partitioning

Using our partitioning technique discussed in Section 5.5.1, we select three scenes with variance in precipitation values (precip=5%, precip=40%, precip=50%) as the precipitation partition $P1$, and the remaining three scenes with variance in brightness values (bright=9%, bright=25%, bright=40%) as the brightness partition $P2$.

#### 5.6.2.2 Latent Space Encoding

We applied the Bayesian optimization algorithm-based heuristic discussed in Section 5.5.2 to select and train the $\beta$-VAE with appropriate hyperparameters.

**Network Structure and Training**: We designed a $\beta$-VAE network that has four convolutional layers $32/64/128/256$ with $5x5$ filters and $2x2$ max-pooling followed by four fully connected layers with 2048, 1000, 250 and 50 neurons. A symmetric deconvolutional decoder structure is used as a decoder. This network, along with images in $\mathscr{T}$, was used in the Bayesian optimization algorithm discussed in Algorithm 4. For each iteration of the Bayesian optimization algorithm, the network was trained for 100 epochs using the Adam gradient-descent optimizer and a two-learning scheduler, that had an initial learning rate $\eta$ = 1 x $10^{-5}$ for 75 epochs, and subsequently fine-tuning $\eta$ = 1 x $10^{-6}$ for 25 epochs. A learning rate scheduler improves the model's accuracy and explores areas of lower loss. In addition, we had an early stopping mechanism to prevent the model from overfitting.

The algorithm also involved computing the MIG in every iteration. For computing it, we utilized the images and labels from partitions $P1$ and $P2$, which were generated in the previous step. To obtain robust MIG, we computed the latent variable entropy by randomly sampling 500 samples from each latent variable in the latent space. To back this, we also averaged the MIG across five iterations.

**Performance Comparison**: We compared the performance of Bayesian optimization, grid, and random search algorithms. The results of these algorithms are shown in Table 5.1. Random search and Bayesian optimization algorithm was run for 50 trials, while the grid search was run for 360 trials across all combinations of n $\in [30,200]$ and $\beta \in [1,5]$. In comparison, the Bayesian optimization algorithm achieved the highest MIG value of 0.0018 for $\beta = 1.4$ and $n = 30$ hyperparameters. It also took the shortest time of 837.05 minutes (early termination because optimal hyperparameters(s) were found) compared to the other algorithms.

| Algorithm | # of Iterations | Iterations to reach optimum | Search Time (min) | max MIG | Selected Parameter |
|---|---|---|---|---|---|
| Grid | 360 | 5 | 10924.55 | 0.0017 | 30,1.4 |
| Random | 50 | 40 | 1199.51 | 0.00032 | 40,1.5 |
| **BO** | **50** | **16** | **837.05** | **0.0018** | **30,1.4** |

Table 5.1: Comparing hyperparameter search algorithms. Bayesian optimization algorithm is compared against random and grid search algorithms.

| Partition | Latent Variable set $\mathscr{L}_P$ | | | |
|---|---|---|---|---|
| P1 | $L_2$**(0.09)** | $L_{25}$(0.06) | $L_0$(0.05) | $L_{29}$(0.02) |
| P2 | $L_{25}$**(0.16)** | $L_0$(0.09) | $L_{20}$(0.07) | $L_2$(0.07) |

Table 5.2: Latent variable mapping. Ordered List of latent variable set $\mathscr{L}_P$ for $P1$ and $P2$. The latent variable $L_2$ had highest KL-divergence variance across the scenes in $P1$. $L_{25}$ had the highest KL-divergence variance across the scenes in $P2$. $\mathscr{L}_d$ is chosen as the union of the two $\mathscr{L}_P$ sets. Chosen $\mathscr{L}_d = \{L_0, L_2, L_{20}, L_{25}, L_{29}\}$

#### 5.6.2.3 Latent Variable Mapping

We used the selected and trained $\beta$-VAE with the data partitions $P1$ and $P2$ to find latent variables for OOD detection ($\mathscr{L}_d$) and reasoning ($\mathscr{L}_f$) using Algorithm 5. First, we used the successive images in each scene to generate a latent variable set $\mathscr{L}$ and then computed a KL-divergence value. Second, we computed an average KL-divergence difference between corresponding latent variables of the two images. Third, we computed the average KL-divergence difference (using Eq. (5.5)) for each latent variable across all the subsequent images in a scene. We repeated these steps for all the scenes in both partitions. Finally, for each partition we identified a partition latent variable set $\mathscr{L}_P$ using the Welford's algorithm as discussed in Section 5.5.3. The number of latent variables $m$ in $\mathscr{L}_P$ requires selection based on the dataset. In this work, the value for $m$ is chosen by human judgment. For the CARLA dataset, we chose the value of $m$ to be 4.

Implementing these steps, we selected the partition latent variable sets $\mathscr{L}_{P1} = \{L_2, L_{25}, L_{20}, L_0, L_{29}\}$ for $P1$ and $\mathscr{L}_{P2} = \{L_{25}, L_0, L_{20}, L_2, L_{29}\}$ for $P2$. These latent variables had the highest KL-divergence variance values and hence were selected. The KL-divergence variance values of these latent variables are reported in Table 5.2. Then, the union of these two partition sets were used as the total detector $\mathscr{L}_d = \{L_0, L_2, L_{20}, L_{25}, L_{29}\}$. Fig. 5.10 shows the scatter plots of the latent distributions of the selected latent variables and 5 randomly selected latent variables ($L_1, L_3, L_5, L_{15}, L_{28}$) for the train images (yellow points) and the test images (green points), which are OOD with high brightness. The latent distributions of the selected latent variables highlighted in the red box form evident clusters between the train images and the test images, and these clusters have a good intra-cluster separation. But, for the other latent variables, the distributions are scattered and do not form clean clusters, and these clusters are not well separated.

Further, we chose one latent variable with the maximum variance in the partition latent variable sets, which

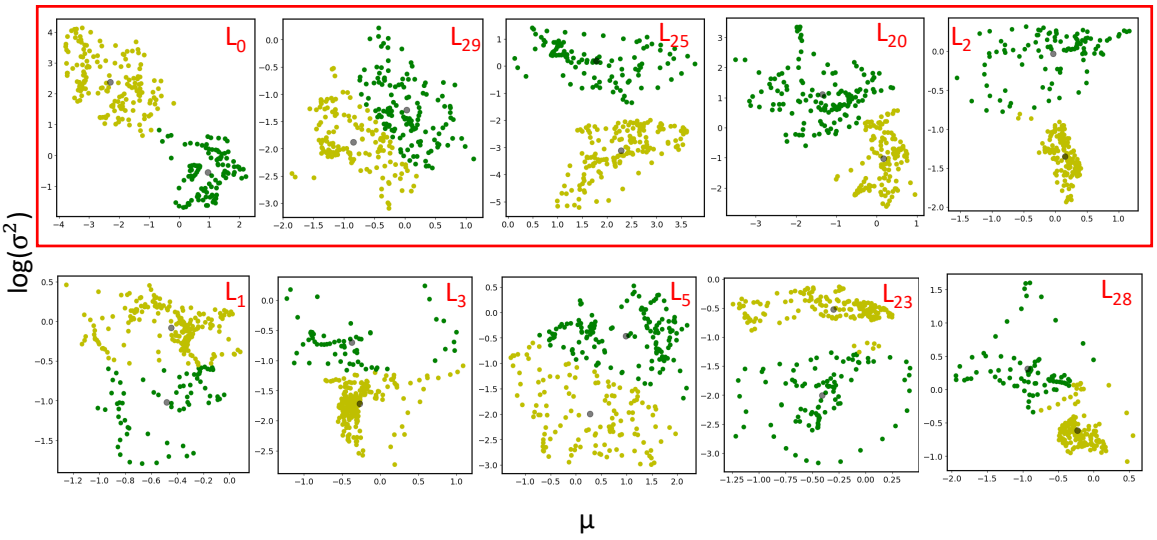



Figure 5.10: Scatter plots of individual latent variables. The latent distributions of the selected latent variables (highlighted in red) as compared to 5 other latent variables that were not selected. Yellow points represent the distributions of the train images, and green points represent the distributions of the test images that were OOD. The selected latent variables into a well-formed cluster, and there is a higher separation between the train and the test image clusters. The un-selected latent variables do not form clean clusters. Using the 5 most informative latent variables for detection resulted in better robustness and minimum sensitivity as compared to using all the latent variables as shown in Table 5.4. Plot axis: x-axis represents the mean of the latent distributions in the range $[-5,5]$, and the y-axis represents the log of variance in the range $[-5,5]$.

was used as the reasoner for that partition. We chose $L_2$ as the reasoner for the precipitation partition $P1$ and $L_{25}$ is used as the reasoner for the brightness $P2$. Our decision to choose only one latent variable for reasoning is backed by the fact that the dataset was synthetically generated, and the features were not highly correlated. However, real-world datasets may require more than one latent variable for reasoning.

### 5.6.3 OOD Detection Results from CARLA Simulation

#### 5.6.3.1 Evaluation Metrics

(1) *Precision (P)* is a fraction of the detector identified anomalies that are real anomalies. It is defined in terms of true positives (TP), false positives (FP), and false negatives (FN) as $TP \div (TP + FN)$. (2) *Recall (R)* is a fraction of all real anomalies that were identified by the detector. It is calculated as $TP \div (TP + FP)$. The FP and FN for the test scenes are shown in Table 5.3. (3) *F1-score* is a measure of the detector's accuracy that is computed as $2 \times (P \times R) \div (P + R)$. (4) *Execution Time* is computed as the time the detector receives an image to the time it computes the log of martingale. (5) *Average latency* is the number of frames between the detection and the occurrence of the anomaly. (6) *Memory Usage* is the memory utilized by the detector.

| Scene | FP | FN |
|---|---|---|
| *HP* | The image is not OOD due to high precipitation, but it is identified as OOD. | The image is OOD due to high rain, but is identified as in-distribution |
| *HB* | The image is not OOD due to high brightness, but is identified as OOD. | The image is OOD due to high brightness, but is identified as in-distribution. |
| *HPB* | The image is not OOD due to high precipitation and high brightness but, is identified as OOD. | The image is OOD due to high rain and high brightness, but is identified as in-distribution. |
| *NR* | The image is not OOD due to change in road segment, but is identified as OOD. | The image is OOD due to change in road segment, but is identified as in-distribution. |

Table 5.3: Definitions of false positives and false negatives for the *HP*, *HB*, *HPB*, and *NR* test scenes.

#### 5.6.3.2 Runtime OOD Detection

We evaluate the performance of the selected $\beta$-VAE network ($\beta = 1.4$ and $n = 30$) for the 5 test scenes described in Fig. 5.8. Additional hyperparameters used by the detectors and the reasoners are as follows. The martingale sliding window size $M = 20$, the CUSUM parameters for the detector are $\omega_d = 14$ and $\tau_d = 100$, and for the reasoners are $\omega_r = 18$ and $\tau_r = 130$. These hyperparameters were selected empirically based on the false-positive results from several trial runs. Fig. 5.11 summarizes the detector's performance for a short segment of the 5 test scenes. For the *HP*, *HB*, *HPB*, and *NR* scenes, the scene shifts from in-distribution to OOD at $t = 2$ seconds, and they are used to illustrate the detection and reasoning capability of our approach.

The *Nom* scene has all the feature values within the training distribution, and it is used to illustrate the detector's ability to identify in-distribution images. As seen in Fig. 5.11, the martingale of the detector and both the reasoners remain low throughout. In *HP* and *HB* scenes, the precipitation and the brightness feature values increase from the training distribution at $t = 2$ seconds. In the *HP* scene, the martingales of the detector and the reasoner for the precipitation feature increase above the threshold after $t = 2$ seconds. However, as seen martingale of the reasoner for brightness does not increase. So, we conclude that the precipitation feature is the reason for the OOD. In the *HB* scene, the martingales of the detector and the reasoner for the brightness feature increase above the threshold after $t = 2$ seconds. Also, the martingale of the reasoner for precipitation shows a slight variation but does not increase above the threshold. So, we conclude the brightness feature is the cause of the OOD. In the *HPB* scene, the precipitation and brightness feature increase to a value out of the training distribution at $t = 2$ seconds. The martingale of the detector and the feature reasoners increases above the threshold. So, we attribute both the features to be the cause of the OOD. The peaks in the steering plots at $t = 2$ seconds are when the DAVE-II DNN steering predictions get erroneous, and the decision manager arbitrates the control to the autopilot controller.

The *NR* scene is of interest for this work, as the scene has a new road segment with different background artifacts (e.g., buildings, traffic lights) that were not in the training distribution. But the precipitation and

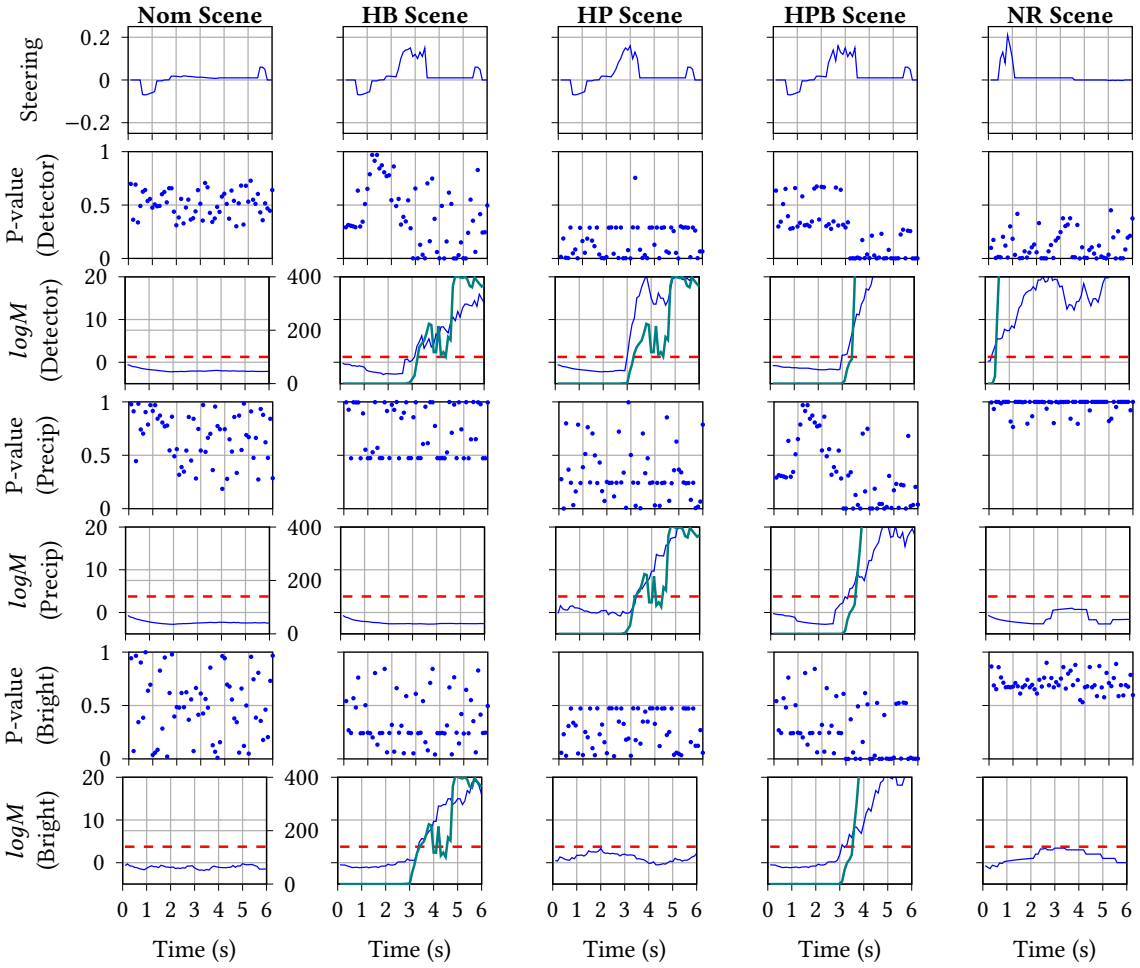



Figure 5.11: Performance of the $\beta$-VAE detector for the 5 test scenes generated in Section 5.6.1. The solid blue line represents the log of martingale, the solid green lines represent the CUSUM values, and the dotted red lines represent the threshold ($\tau$) for CUSUM comparison. Further, the left y-axis shows the log of martingale with a range of $[-5, 20]$, and the right y-axis shows the CUSUM value with a range of $[0, 400]$. The cusum threshold in the red dotted lines is plotted to the right y-axis.

brightness feature values were within the training distribution. When tested on this scene, the detector martingale instantly increases above the threshold, identifying the scene as OOD. However, the martingales of the reasoners for precipitation and brightness only show slight variations without increasing beyond the threshold. The result implies that the scene is OOD, but a feature other than brightness and precipitation has varied and is responsible for the OOD.

Further, Fig. 5.12 shows the plots of the capability of the $\beta$-VAE detector in identifying changes in the current test image as compared to the previous images in the time series (*problem2*). For these evaluations, we used the *Nom*, *HB*, and extended *HB* test scenes, which had a length of 50 seconds each. The *Nom* and *HB* scenes are the same as the previous setup, but in the extended *HB* scene, the brightness feature value abruptly

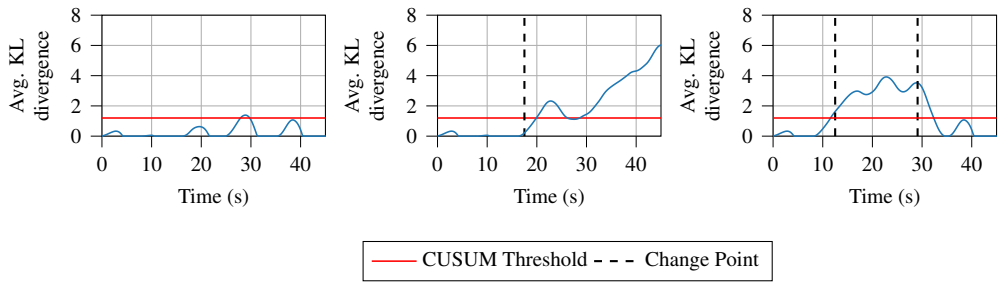



Figure 5.12: Moving Average and CUSUM for identifying feature changes in *Nom*, *HB*, and an extended *HB* test scenes. (Left) *Nom* scene - there is no change in the scene, so the average KL-divergence remains below $\tau$. (Center) *HB* scene - the brightness of the scene changes at 17.5 seconds, and the detector identified this. (Right) Extended *HB* scene - the brightness of the scenes increases for a short period between 9.5 seconds to 29.5 seconds, and the detector could identify both the changes.

increases for a brief period between 9.5 seconds and 29.5 seconds. For these test scenes, we performed the moving average calculation on a sliding window of $M = 20$ with the CUSUM parameters of $\omega_{cp} = 0.75$ and $\tau_{cp} = 1.2$. With these parameters, our detector could identify change points with a short latency of 11 frames, which translates to one second of inference time for the AV system.

#### 5.6.3.3 Evaluating the Design Approach

Table 5.4 illustrates how the proposed heuristics in each step of our approach (Section 5.5) results in achieving the best detector properties (text highlighted in green). The properties of interest are robustness, minimum sensitivity, and resource efficiency, defined in Section 5.3. We evaluate the robustness of the detectors on the in-distribution scene *Nom* and two OOD scenes *HPB* and *NR*. The *HP* and *HB* scenes had variations in either the precipitation value or the brightness value, but they did not have simultaneous variations. So, it was suitable to use them to measure the detector's minimum sensitivity towards these features. However, in the *HPB* scene, both these features varied simultaneously, so it was unsuitable for measuring the minimum sensitivity. The resource efficiency was measured across all the scenes. We have also compared the proposed heuristics to an alternate technique in each step. The comparisons are as follows (proposed techniques are underlined): (1) MIG vs. ELBO loss function (discussed in Section 5.4), (2) Bayesian optimization vs. Grid and Random Search, and (3) Selective vs. All latent variables for detection. Our evaluations are as follows.

If the dataset can be partitioned, either MIG or ELBO can be used as the objective function with the Bayesian optimization, grid, or random search algorithms. Since the dataset can be partitioned, the latent variable heuristic could be applied to select a subset of latent variables, which can be used for detection. As illustrated in Table 5.4, the optimization algorithm and objective function combinations resulted in 5 different

| Design-time steps of our approach | | | | Detector Properties | | |
|---|---|---|---|---|---|---|
| Partitioning | Latent Space Encoding | | Selected $\mathscr{L}_d$ for Detection | Robustness F1-score (%) | Minimum Sensitivity (%) | Execution Time (ms) |
| | Objective Function | Optimization Algorithm | | | | |
| Yes | MIG | BO (30,1.4) | [0,2,20,25,29] | 96.98 | 96 | 74.09 |
| | | Grid (30,1.4) | [0,2,20,25,29] | 96.98 | 96 | 74.09 |
| | | Random (40,1.5) | [0,17,6,8,7] | 80.75 | 35 | 79.15 |
| | ELBO | BO (30,1.0) | [0,1,6,21,23] | 95.83 | 63 | 75.39 |
| | | Grid (40,1.0) | [13,28,26,23,0] | 84.65 | 54 | 78.85 |
| | | Random (30,1.2) | [10,14,21,22,26] | 94.9 | 71 | 74.98 |
| No | ELBO | BO (30,1.0) | All 30 | 85.89 | 73 | 379.47 |
| | | Grid (40,1.0) | All 40 | 68.96 | 42 | 488.85 |
| | | Random (30,1.2) | All 30 | 89.73 | 53 | 396.89 |

Table 5.4: Design approach evaluation. Evaluations of how the heuristics of our design approach influence the detector properties discussed in Section 5.3. We evaluated Robustness on the *Nom*, *HPB*, and *NR* scenes. Minimum sensitivity was evaluated on the *HP* and *HB* scenes. Resource efficiency was measured across all the test scenes. The numbers in the optimization algorithm column indicate the selected hyperparameters. Text in green highlights the best detector properties achieved by our approach. Text in red highlights a high detection time.

$\beta$-VAE networks and 5 latent variables for $\mathscr{L}_d$. Among these, the Bayesian optimization and grid algorithms using MIG resulted in the best detector with the robustness of 96.98%, minimum sensitivity of 96%, and a detection time of 74.09 milliseconds. In comparison, the other detectors had low robustness, minimum sensitivity, and a similar detection time.

However, if the dataset cannot be partitioned, ELBO is the only objective function that can be used with Bayesian optimization, grid, or random search algorithms. However, the latent variable mapping heuristic cannot be applied without partitioning. So, all the latent variables for the chosen $\beta$-VAE had to be used for detection. Using all the latent variables for detection resulted in a less robust and sensitive detector that took an average of 400 milliseconds as shown in Table 5.4.

### 5.6.4 Detection Results from Competing Baselines

We compare the performance of the $\beta$-VAE detector to other state-of-the-art approaches using our AV example in CARLA. The approaches that we compare against are (1) Deep-SVDD one-class classifier; (2) VAE-based reconstruction classifier; (3) chain of one-class Deep-SVDD classifiers; and (4) chain of VAE-based reconstruction classifiers. The VAE network architecture is the same as that of $\beta$-VAE (described in Section 5.6) but uses the hyperparameters of $\beta = 1$ and $n = 1024$. The one-class Deep-SVDD network has four convolutional layers of 32/64/128/256 with 5$x$5 filters with LeakyReLu activation functions and 2$x$2 max-pooling, and one fully connected layer with 1568 units. These networks are also trained using a two-learning scheduler, with 100 epochs at a learning rate $\eta$ = 1 x $10^{-4}$, and 50 epochs at a learning rate $\eta$ = 1 x $10^{-5}$. Further, we combined two of these classifiers to form a chain of Deep-SVDD classifiers and a chain

of VAE classifiers. In each chain, one classifier is trained to classify images with variations in the values of the brightness feature, and the other is trained to classify images with variations in the values of the precipitation feature. These classifiers are combined using an OR operator.

For these evaluations, we set resource limits on the python software component (See Fig. 5.7) for evaluating the processing time and memory usage. We assigned a soft limit of one CPU core and a hard limit of four CPU cores on each component to mimic the settings of an NVIDIA Jetson TX2 board. Further, to measure memory usage, we used the psutil [290] cross-platform library.

#### 5.6.4.1 Comparing Runtime OOD Detection

The false-positive and false-negative definitions for the *HP*, *HB*, *HPB*, and *NR* test scenes are defined in Table 5.3. Based on these definitions, the precision and recall of the different detectors for the test scenes are shown in Fig. 5.13. In *HP* and *HB* scenes, a single feature (precipitation or brightness) value was varied, and the classifier that was not trained on the representative feature of that scene had low true positives. So, the precision and recall of these detectors are mostly zero. In the *HPB* scene shown in Fig. 5.13-c, both the features were varied, so a single one-class classifier was insufficient to identify both the feature variations. However, the one-class classifier chains with an OR logic had higher precision in detecting the feature variations in all these scenes. Similarly, the detection and OOD reasoning capability of the $\beta$-VAE detector could identify both the feature variations.

For the *NR* scene shown in Fig. 5.13-d, which had a new road segment with the precipitation and brightness values within the training distribution, both the one-class classifiers and their chains raise a false alarm. So, their precision towards detecting variations in these features is low (roughly 1%). In contrast, the $\beta$-VAE detector alongside the reasoner could identify that the OOD behavior was not because of the precipitation and brightness with a precision of 46%. These results imply that the $\beta$-VAE detector can precisely identify if the features of interest are responsible for the OOD. Whereas a similar reasoning inference cannot be achieved using the other approaches.

#### 5.6.4.2 Comparing Execution Time and Latency

**Execution Time**: As discussed earlier, we selected 5 latent variables for detection and one latent variable each for reasoning about the precipitation and brightness features. Two components that mainly contribute to the execution time of the $\beta$-VAE detector are: (1) the time taken by the $\beta$-VAE's encoder to generate the latent variables, and (2) the time taken by ICP and martingale for runtime detection. The average execution time using all 30 latent variables of the $\beta$-VAE was 400 milliseconds, which was drastically reduced to 74.09 milliseconds when the 5 selected latent variables were used. Also, the reasoner only took about 9 milliseconds

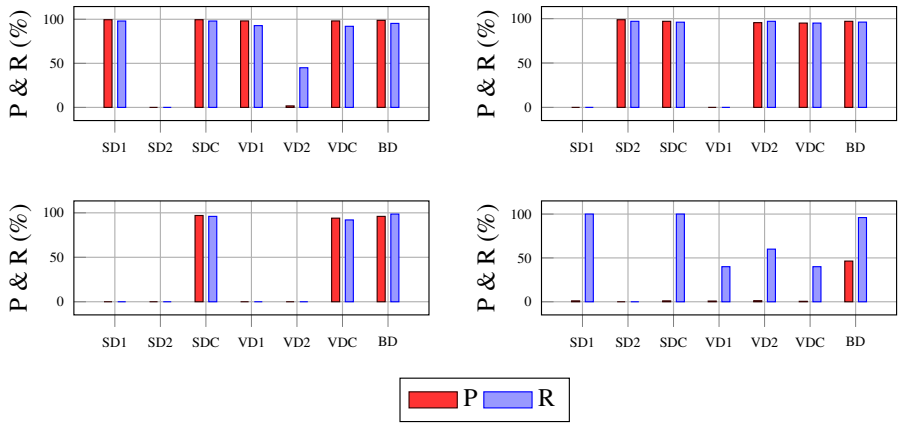



Figure 5.13: Precision and Recall. Evaluations on different scenes: (a) $HP$ - top left, (b) $HB$ - bottom left, (c) $HPB$ - top right and (d) $NR$ - bottom right. The detectors compared are: $SD1$ - Deep-SVDD Precipitation detector, $SD2$ - Deep-SVDD brightness detector, $SDC$ - Deep-SVDD detector chain, $VD1$ - VAE Precipitation detector, VD2 - VAE brightness detector, VDC - VAE detector chain, BD - $\beta$-VAE detector. These values were collected by running the detectors on each scene 20 times.

as it worked in parallel to the detector. In comparison, the Deep-SVDD classifiers took an average of 41 milliseconds for detection, and its chain took an average of 43.36 milliseconds, as shown in Fig. 5.14. Also, each of the VAE-based reconstruction classifiers took an average of 53 milliseconds for detection, and its chain took an average of 57 milliseconds. The Deep-SVDD and VAE-based reconstruction classifiers performed slightly faster than our detector.

**Detection Latency**: The detection latency in our context is the number of frames between the detection and the occurrence of the OOD. In our approach, the latency is dependent on the size of the martingale window, which is dependent on the CPS dynamics and the sampling period of the system as discussed in Section 5.4.4. In addition, the selection of the CUSUM threshold also impacts the latency. In these experiments, our AV traveled at a constant speed of 0.5 m/s, so we used a fixed window size of 20 images. Further, we empirically selected the CUSUM threshold to be 100. With this configuration, the $\beta$-VAE detector had an average latency of 12.6 frames, and the reasoners had an average latency of about 11.5 frames across the 4 test scenes. In comparison, the Deep-SVDD classifiers and their chain had an average latency of 11 frames and 11.21 frames, respectively. Also, each of the VAE-based reconstruction classifiers and their chain had an average latency of 13 frames and 12.9 frames, respectively.

To summarize, all the approaches had similar detection latency, with the Deep-SVDD classifier performing slightly better. The Deep-SVDD classifier and its chain had slightly shorter latency than our detector. However, the $\beta$-VAE detector had a lower latency than the VAE-based reconstruction classifier and its chain.

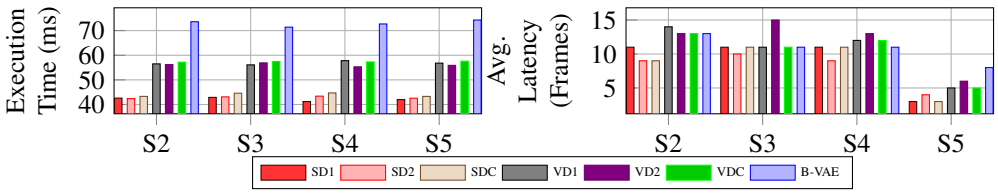



Figure 5.14: (Left) Execution Time in milliseconds and (Right) Detection latency in number of frames, of the different detectors for the $HP$, $HB$, $HPB$ and $NR$ scenes. The detectors are $SD1$ - Deep-SVDD Precipitation detector, $SD2$ - Deep-SVDD brightness detector, $SDC$ - Deep-SVDD detector chain, $VD1$ - VAE Precipitation detector, $VD2$ - VAE brightness detector, $VDC$ - VAE detector chain, $BD$ - $\beta$-VAE detector. These values were collected by running the detectors on each scene 20 times.

#### 5.6.4.3 Comparing Memory Usage

Our approach uses a single $\beta$-VAE network to perform detection and reasoning. Specifically, we only utilize the network's encoder instead of the encoder and the decoder. The average memory utilization of the $\beta$-VAE detector was 2.49 GB, and the reasoners were 0.23 GB. In comparison, the Deep-SVDD classifiers and their chain utilized an average memory of 3.2 GB and 6.4 GB, respectively. Also, the VAE-based reconstruction classifiers and their chain utilized an average memory of 3.6 GB and 7.2 GB, respectively. The classifier chains utilized higher memory because two DNNs were used for detection.

To summarize, our approach utilizes lesser memory because: (1) it only requires the encoder of a single $\beta$-VAE network for both detection and reasoning; and (2) it utilizes fewer latent variables for detection because it relies on the disentanglement concept. For the AV example, our detector only required 5 latent variables compared to 1024 latent variables of the VAE reconstruction classifier and 1568 activation functions in the embedding layer of the Deep-SVDD classifier.

### 5.6.5 OOD Detection Results from nuImages dataset

As the second example, we apply our detection approach to a small fragment of the nuImages [251] dataset. We report the preliminary results of our evaluation in this section.

**Dataset Overview**: The nuImages dataset is derived from the original nuScenes dataset [252]. The nuImages dataset provides image annotation labels, reduced label imbalance, and a larger number of similar scenes (e.g., a scene with the same road segment but different time-of-day, weather, and pedestrian density values), which makes it suitable for our work. The dataset has $93,000$ images collected at $2Hz$ and annotations for foreground objects such as vehicles, animals, humans, static obstacles, moving obstacles, etc. The foreground objects further have additional attributes about the weather, activity of a vehicle, traffic, number

of pedestrians, and pose of the pedestrians, among others. We demonstrate our OOD detection capability on scenes with different values of traffic and pedestrian density features. For these experiments, we trained the detector with different traffic and pedestrian values. We then used it to detect images in which either traffic or pedestrian features are absent (OOD condition).

**Data Partitioning**: The training set $\mathscr{T}$ had images from 4 scenes with varying traffic and pedestrian values. We partitioned $\mathscr{T}$ into two partitions. $P1$ had images with medium and high pedestrian density values in the range $[1,5]$ and more than 5, respectively. $P2$ had images with medium and high traffic values in the range $[1,5]$ and more than 5, respectively. The test scenes for this evaluation were nominal (*Nom*) and a pedestrian or traffic (*PoT*) scene. The *Nom* scene was in distribution, with 30 images in time series captured on a sunny day from a road segment in Singapore city with traffic and pedestrians in the training distribution range. The *PoT* scene had 70 images in time series captured from a similar road segment, and it either had traffic or pedestrians, but not both together. For most images in this scene, the traffic density took a value in the training range, but the pedestrian value was close to zero. Further, our test scenes were short as it was difficult to find longer image sequences that belonged to a sunny day and similar road segments.

**Detector Design**: We applied our approach discussed in Section 5.5 and we used the same $\beta$-VAE network structure as discussed in Section 5.6.2. Using this set up, we selected a $\beta$-VAE network with hyperparameters $n$=30 and $\beta$=1.1, that resulted in the maximum MIG of 0.0006. We then identified the detector latent variable set $\mathscr{L}_d$ to be $\{L_0, L_7, L_9, L_{22}\}$. Further, we selected latent variables $L_9$ and $L_{22}$ as the reasoners for the traffic density and pedestrian density partitions, respectively.

**OOD Detection**: For runtime OOD detection, we used a window size $M$=30 for martingale computation and $\omega$=2 and $\tau$=10 for reasoners and detector CUSUM calculations. The plots for detection and reasoning are shown in Fig. 5.15. The detector Martingale for the *Nom* test scene remained low, and all in-distribution images were detected as in-distribution. The martingale of the traffic density reasoner remained low for most images except for 2 images. We hypothesize that this could be because the vehicle and background blended well, confusing the reasoner. The martingale of the pedestrian reasoner remained low throughout the scene. For OOD scene *PoT*, the detector correctly identified 92% (65/70) of the images as OOD. However, the first 5 images were incorrectly detected to be in-distribution because of the detection latency of our window-based martingale approach. Also, as the traffic density was mostly zero throughout the scene, the martingale of traffic density reasoner is mostly flat throughout. But, for pedestrian reasoners, all the images were identified as OOD except the first 9 images. The false negatives are primarily because of the complexity of the operational scene and the detection latency. After these images, the martingale increases, and the CUSUM increases above its threshold. In summary, the reasoners worked reasonably well for these scenes but had a slow martingale growth because of several background attributes like sun glare, trees, and traffic lights.

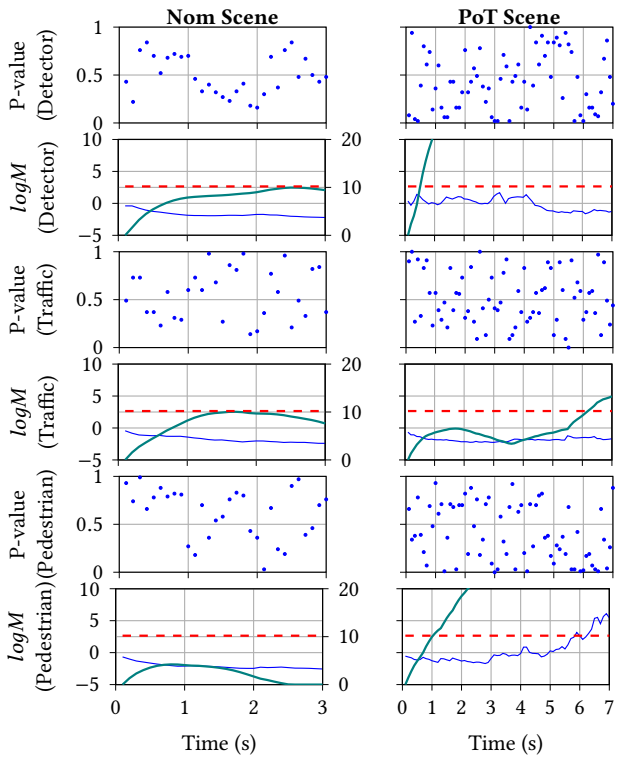



Figure 5.15: Runtime OOD detection on nuImages dataset. Performance of the $\beta$-VAE detector for the *Nom* (nominal) and *PoT* (pedestrian or traffic) scenes of the nuImages experiments. For the martingale plots, the solid blue line represents the log of martingale, the solid green lines represent the CUSUM values and the dotted red lines represent the threshold ($\tau$) for CUSUM comparison. Further, the left y-axis shows the log of martingale with range $[-5, 10]$, and the right y-axis shows the CUSUM value with range $[0, 20]$. The cusum threshold represented in the red dotted lines is plotted to the right y-axis.

**Challenges**: We discuss the several challenges of applying our approach to a real-world dataset. First, limited availability of time-series images. Although the nuImages dataset provides the notion of a scene, they contain time gaps, especially after partitioning them into train and test the dataset. Second, the labels provided for the dataset images are coarse grain. For example, in the nuImages dataset, the semantic label annotations are only limited to high-level foreground objects like pedestrians and cars, and extracting these labels requires significant pre-processing. Another challenge is the complexity of images in real-world datasets. The presence of excess background information such as trees, traffic signals, shadows, and reflections, among others, makes the real-world images complex and impacts the information in latent space. The other challenge is the absence of scenes in which the feature(s) gradually change their values. This makes it difficult to apply our latent variable mapping. Further, finding similar scenes with variations in the specific feature(s) of interest is difficult. For our experiments, we had to perform significant pre-processing to extract short image sequences to perform OOD detection and reasoning.

### 5.6.6 Discussion

With LECs being widely used in perception pipelines of automotive CPS, there has been an increased need for OOD detectors that can identify if the operational test image to the LEC is in conformance to the training set. Addressing this problem is challenging because these images have multiple feature labels, and a change in the value of one or more features can cause the image to be OOD. This problem is commonly solved using a multi-chained one-class classifier with each classifier trained on one feature label. However, as shown by our evaluation in Section 5.6.4, the chain gets computationally expensive with an increased number of image features. So, we have proposed a single $\beta$-VAE detector that is sensitive to variations in multiple features and computationally inexpensive compared to the classifier chains. For example, to perform detection on a real-world automotive dataset like nuScenes with 38 semantic labels, a multi-chained VAE-based reconstruction classifier (discussed in our experiments) would require training 38 different VAE DNNs as compared to a single $\beta$-VAE detector that is presented in this work. A memory projection based on the results in Section 5.6.4.3 is shown in Fig. 5.16. It shows that the multi-chain classifier will need a memory of 136.8 GB. In comparison, our approach requires a single network with one or a few latent variables for detection on each label, and this requires only 10.96 GB of memory.

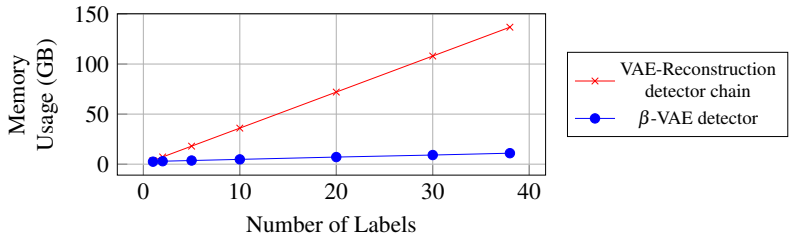



Figure 5.16: Predicted memory usage of a VAE-based reconstruction chain and a single $\beta$-VAE detector for the nuScenes images. If a VAE reconstruction detector is used for each of the 38 nuScenes labels, then the memory usage would linearly grow to 136.8 GB. However, our approach's $\beta$-VAE detector would only require 10.96 GB. These values were computed based on results in Section 5.6.4.3.

Another related problem motivated in this work is the OOD reasoning capability to identify the most responsible feature(s). The reasoning capability is desirable for the system to decide on the mitigation action it must perform. For example, if the detector can identify a high traffic feature to cause the OOD, then the system can switch to an alternate controller with lower autonomy. This reasoning capability cannot be achieved using the state-of-the-art chain of one-class classifiers. But the $\beta$-VAE detector designed and trained using our approach is sensitive to variations in multiple features. We have evaluated the capability of the $\beta$-VAE detector by applying it to several OOD scenes in CARLA simulation (see Fig. 5.11). Especially the *NR* scene with a new road segment and background artifacts not in the training set. In this scene, the multi-chained network

identified the scene as OOD because of the precipitation and brightness features. In comparison, our detector's OOD reasoning capability could identify with a precision of 46% that the cause was not precipitation or brightness.

In addition to the reasoning capability, a detector for a multi-labeled dataset should have minimum sensitivity (defined in Section 5.3) towards all the feature labels. High minimum sensitivity is needed to detect variations in all the feature labels of the training set images. In our approach, we hypothesize that using the most informative latent variables can provide good minimum sensitivity for the detector towards each feature label. We back this by our results in Table 5.4, which illustrates that a detector that used the 5 most informative latent variables had a high minimum sensitivity to variations in both the precipitation and brightness features (highlighted in green text in the Table). In comparison, the detectors that used all the latent variables for detection had lower minimum sensitivity, as illustrated in the Table.

Besides identifying if the input images are OOD to the training data, it is necessary to check if they have changed compared to the previous sequence of images in the time series. Such abrupt changes in the current input will increase the system's risk of consequence (e.g., collision) [117], so it is critical to detect them. This problem must be addressed but has not been given much importance in the OOD detection literature. In this work, we use the detector's latent variables in a moving window based on CUSUM for detecting abrupt changes in the features of the operational test images. We validated our approach across 3 different scenes (see Fig. 5.12) for which our detector could accurately identify feature variations with a short latency of 11 images.

Finally, in designing OOD detectors for CPSs, one needs to consider the system's dynamics. But this is often not given importance in the existing detection approaches. Only recently, Cai *et al.* [92] have proposed an OOD detection mechanism on time-series images that consider the system dynamics. As discussed in Section 5.5.4, we rely on their ICP and martingale framework approach for runtime detection over a short sliding window of images. The sliding window size needs to be adjusted based on the operational system dynamics like runtime sampling rate and the speed at which the system travels.

## 5.7 Related Work

There has been significant ongoing research to handle the brittleness and susceptibility of LECs. We have grouped the existing approaches into different classes and briefly discussed them below.

**Domain Adaptation** involves transferring knowledge between a labeled source domain and an unlabeled target domain [291]. The key idea is to learn domain invariant features of the training data by providing less importance to dataset biases. This is also referred to as transductive transfer learning and is used when the training and the target tasks remain the same while the domains are different. Altering the DNN structure

has been one of the approaches that have been used for feature transferability. One such example is the Deep Adaptation Network [292] that allows for feature transferability in the task-specific layers of DNN while reducing domain-related information. The other widely adopted approach is biasing the training objective to learn the domain invariant feature(s). For example, Heinze-Deml *et al.* [293] proposes a conditional variance penalty-based training loss function to learn domain invariant features. Although domain adaptation has been widely used, an assumption on the prior distributions of target domains is restrictive for practical applications like CPSs [294]. Further, negative knowledge transfer between the source and target domains has been a common problem [295].

**Confidence Estimation** involves estimating the confidence in the inputs to a DNN. There have been several approaches to estimating confidence. The first approach involves adding a confidence branch at the logits before the softmax layer [296]. The second approach involves using Bayesian Networks to represent uncertainties in the DNN [297, 298]. Ensembles of DNNs have been another approach used for estimating the confidence in the inputs [299, 300]. All these approaches mostly require making changes to the DNN or will require training model ensembles. Recently, Jha et al. [301] has proposed the attribution-based confidence (ABC) metric that does not require access to training data or does not need training model ensembles. It is computed by sampling in the neighborhood of high-dimensional data and then computing a score for its conformance. The metric looks robust and does not require access to the training data at runtime, which is an advantage compared to our approach. But the impact of input feature correlations and interactions on the attribution over features require future investigation [302].

**Identifying Distribution Shifts** involves identifying if the test observation has shifted from the training distribution. Probabilistic classifiers like generative adversarial networks and variational autoencoders have been widely adopted for identifying shifts in the test observations [259, 92, 95, 94]. Our work belongs to this class, and different approaches in this class are discussed in Section 5.7.1.

In contrast to these approaches, we formulate the problem as a multi-label OOD detection. In addition to identifying an OOD condition, we also find the feature(s) causing it.

### 5.7.1 Probabilistic One-class Classifiers

The OOD detectors, in general, learn the training data samples as members of a target class, and operational data that is not similar to those seen during training will exist outside the target class and is classified as OOD. Principal Component Analysis (PCA) [256] was one of the first one-class classifier approaches developed for unsupervised detection. It performs a linear transformation to map high-dimensional data into a lower-dimensional space for detection. A non-linear extension of PCA called kernel PCA has been introduced [303].

Kernel-based one-class classifiers have been the other popular approach for detection, and it includes the One-Class SVM (OC-SVM) [257] and Support Vector Data Descriptor (SVDD) [304]. Both OC-SVM and SVDD construct a boundary around most training data samples, and the operational data that lies outside the boundary is classified as OOD. However, the performance of these methods has been shown to be sub-optimal on complex, high-dimensional data, which has resulted in the use of deep learning models.

Discriminative deep learning models like multilayer perceptron [305] and Deep-SVDD [259] have shown robust detection capability on high-dimensional data such as images. Among them, Deep-SVDD is the most prominent model. It trains a neural network to learn a minimum volume hypersphere that encloses the network's representation of the training data samples [259]. The results from Cai *et al.* [92] have shown that a well-trained Deep-SVDD can detect OOD images with high precision. However, improper selection of network hyperparameters results in a hyper-sphere collapse problem of the Deep-SVDD network [306]. Recently, there has been growing interest in probabilistic classifiers like Generative Adversarial Network (GAN) [260], Autoencoders (AE) [307], and Variational Autoencoder (VAE) [261].

### 5.7.2 Adversarial Networks

GAN has been a prominent framework for learning generative models from complex and high-dimensional data distributions. A GAN has two competing networks called the generator and the discriminator. The generator is trained to generate fake data samples such that the discriminator would classify these samples as the actual input data. In the training process, the discriminator is optimized to correctly distinguish between the actual input data samples and fake data samples generated by the generator. This property of correctly classifying real and fake data is used for OOD detection [263, 308, 262, 264]. For example, GAN is used along with an adversarial training loss function to perform unsupervised OOD detection [262, 264]. However, their approach of implicitly modeling the data distributions requires additional optimization procedures to recover the latent representations and perform detection. To avoid this, Donahue *et al.* [309] propose the Bidirectional Generative Adversarial Network (BiGAN), which trains an additional encoder in parallel to the generator. In this way, the latent representations are readily available for detection. Despite being widely used, GANs have the following limitations [264, 265]: (1) training complexity because of the instability between the generator and discriminator networks, and (2) mode collapse problem that results in the generator only producing similar samples or, in the worst-case a single sample. These issues make the use of GAN difficult.

### 5.7.3 Autoencoders and Variational Autoencoders

Autoencoders [307] and Variational Autoencoders [261] are the other probabilistic models used for detection. The existing detection techniques using these networks can be classified into two approaches.

- **Reconstruction-based techniques** use the normalized difference between the input data (x) and the reconstructed data (x') generated by the AE or VAE to perform OOD detection. While a low reconstruction error value indicates the input data is in-distribution, a high value indicates the data is OOD. For example, the authors in [91, 92] have used the mean squared error between the original input image and the reconstructed image of a VAE to perform OOD detection. Reconstruction probability is presented as the probabilistic extension to the reconstruction error [94]. Here, a probabilistic anomaly score is computed using the stochastic latent distributions of the VAE. Despite being widely used because of its simplistic approach, the reconstruction approach can be error-prone when the OOD samples lie on the boundary of the training distribution (i.e., close to the learned manifold of the training distribution) [95].
- **Latent space-based techniques** use different distance and density-based metrics on the latent space generated by the VAE to detect OOD samples. For example, Vasilev *et al.* [96] propose several distance-based metrics such as Euclidean distance, Mahalanobis distance, and Bhattacharyya distance to measure the distance between the latent distributions of the input data samples and the operational data. Higher distances between the distributions indicate the operational data is OOD. In another work, Denouden *et al.* [95] use a combination of the reconstruction error and the latent space score to compute a combined OOD score. The Mahalanobis distance between the test image's latent distributions and the training images' mean vector is used as the latent space score.

**Disentangled Latent Representations**: Though prior detection work [95, 96] has successfully used the latent representations for OOD detection, there is a known problem that these representations are unstructured, entangled, and lack ease of understanding [266], which introduces challenges for robust detection. There has been ongoing work to disentangle and learn structured latent representations. Disentanglement is a state of the latent space where each latent variable is sensitive to changes in only one feature while being invariant to changes in the others [112]. Disentangled latent representations have been used in applications such as video predictions [270] and image-based OOD data [271]. Disentanglement is not a new concept, and there have been several approaches in the past to achieve it [112]. However, of particular interest to this research are the VAE-based approaches used for disentanglement. Recently, there have been several variants of VAE like FactorVAE [310], $\beta$-VAE [267], and $\beta$-TCVAE [278] designed to aid the disentanglement process.

The $\beta$-VAE network is the most widely used model because of its simplicity. It provides a $\beta$ hyperparameter

that controls the data flow from the input to the latent space. Appropriately tuning the $\beta$ parameter and the latent space size (n) can disentangle the latent space. Several recent research findings have used the $\beta$-VAE network for OOD detection. For example, Graydon *et al.*[311] have used the latent space of the $\beta$-VAE along with the Gaussian mixture models to identify OOD on cancer datasets. Also, the reconstruction error and the distances among latent distributions of a $\beta$-VAE are used together as the anomaly score in [312]. A well-trained $\beta$-VAE network has outperformed a classical VAE for OOD detection on MRI images [313].

### 5.7.4 Multi-Labeled OOD Detection

Since we are focused on OOD detection of multi-labeled images, we will briefly discuss the current work in multi-labeled classification or detection. Multi-label classification requires simultaneously checking for multiple labels, complicating the detection problem. Algorithm adaptation, problem transformation, and label embedding have been widely adopted approaches. Algorithm adaptation approaches [314] involve designing a custom algorithm for managing and performing detection directly on multiple labels. Problem transformation approaches involve transforming the multi-labeled detection problem into several single-labeled detection problems. Label embedding transforms [254] the input data labels into a latent space containing the input information and implicitly captured information about the label correlations. Multi-label classification is performed at lower computational costs with additional label mapping between the input and the latent space. We focus on the second approach that chains several on-class classifiers in this work. We have utilized this approach in one of our detection works, which will be discussed in the following sections.

Binary relevance (BR) [253] is the state-of-the-art problem transformation approach that splits the multi-labeled dataset into mutually exclusive partitions and trains a binary classifier (one-class classifier) for every generated partition. The trained binary classifiers are chained to perform detection on the actual multi-labeled dataset. However, the major problem with this approach is that it cannot capture the correlations among the labels [254], making it unsuitable for real-world datasets, which often have highly correlating features. For example, images from automotive datasets often have labels like time of day, weather, brightness, and clouds, which are all correlated. Training a one-class classifier on one feature label by ignoring the others can result in sub-optimal detection. To address this problem, Read, Jesse, *et al.* [255] extended the BR technique to a novel chain of classifiers that aimed at exploiting the cross-label dependencies. The classifiers are arranged such that the label prediction of one classifier is passed as the feature parameters to the next classifier in the chain. Unfortunately, this approach performs the detection sequentially on each data label, which is unsuitable for runtime use in systems with short inference times.

## 5.8 Conclusions and Future Work

This research work proposes a design workflow to generate a partially disentangled latent space and learn an approximate mapping between the latent variables and features for OOD detection and reasoning. We use a $\beta$-VAE network and tune its hyperparameter ($n$, $\beta$) using a Bayesian optimization heuristic to generate the disentangled latent space. Next, we perform a latent variable mapping to identify the most informative latent variables for detection and identify latent variable(s) sensitive to specific features and use them for reasoning the OOD problem. We evaluated our approach using an AV example in the CARLA simulation and illustrated the preliminary results from the nuImages dataset.

Our evaluation has shown that the detector designed using our approach has good robustness, minimum sensitivity, and low execution time. The detector could also detect OOD images and identify the most likely feature(s) causing it. The future extensions and applications of the proposed approach include: (1) improving the latent variable mapping heuristic to perform robust correspondence between the features and latent variables, (2) exploring alternate metrics such as Wasserstein distance for latent variable mapping, and (3) applying the approach to real-world datasets and research CPS testbeds like DeepNNCar [80], and (4) use the anomaly detection results to perform higher-level decision making such as controller selection or enactment of contingency plans.

To conclude, while detection provides a mechanism to identify the operational hazards (assumption violations because of dynamically changing operating conditions), the detection results are insufficient to ensure the system's safety. In addition to detection, it is also necessary to assess the additional risk introduced by the identified hazard on the system's operation. However, the assessment is currently performed as a post-consequence activity that involves analyzing a system accident to determine the cause and initiate corrective actions to prevent future errors. This design-time assessment activity is insufficient for the proposed framework, which demands a "proactive assessment" strategy that can utilize the historical accident information with runtime hazards identified by the detectors. Therefore, an open question that still needs to be addressed by this work is: *how can we use the OOD detection results in proactively assessing the system's safety at runtime?*

# Chapter 6

# Resonate: Assessing the Risk of Operating Complex CPS with LECs

## 6.1 Overview

In Chapter 5, we have made a clear case explaining the need for system risk assessment at runtime. Though risk assessment is a core activity of several cyber-physical system (CPS) domains and safety standards (e.g., ISO 26262 [41]), the assessment is performed as an offline post-consequence activity. For example, system risk management [31] and safety risk management [1] are standard approaches used in the aviation industry to quantify risk. These approaches calculate the system's risk at design time and use it for rule-based decision-making during operations [31] (the approach checks the system's state in applying the rules). Here, safety is established by showing that the risk posed to the system is mitigated or sufficiently alleviated to acceptable levels that it can tolerate. Since achieving perfect safety is impossible, the assessment often has to answer two questions: "how safe is safe enough?" and "What is an acceptable risk level?". To answer this, the concept of As Low As Reasonably Practicable (ALARP) [36] and As Safe As Reasonably Practicable (ASARP) [30] were introduced. Both these concepts rely on the notion of reasonable practicability. These are not hard thresholds, and they differ between systems based on the criticality of their operations.

Though widely adopted, the problem with this offline assessment is that it does not proactively utilize runtime information. The risk information computed at design time is used for the system's safety assessment at runtime. However, with the system's evolution on deployment at runtime, the design-time risk information may not be a correct safety indicator. That is, the assessment must involve using runtime information from monitors (anomaly detectors and out-of-distribution (OOD) detectors) to analyze the system's operational risk. Despite this need for proactive risk assessment, few attempts have been made toward this goal.

To address this need for proactive risk assessment, this chapter presents a framework called ReSonAte (Runtime Safety Evaluation in Autonomous Systems). The framework performs proactive assessment by first utilizing the design-time hazard analysis information to build a causal risk causation model called Bow-Tie Diagram. The model describes potential threats and hazards to the system and allows computing the probabilities of how these hazards may result in a severe consequence. Next, the threat probabilities are combined with information about the system's current state derived from system monitors (e.g., anomaly detectors, assurance monitors) and the operating environment (e.g., weather, traffic) to estimate the dynamic hazard rates of the system during operation. The feasibility of the proposed framework is demonstrated using

two CPS applications: an autonomous vehicle (AV) example in an urban simulated environment, and an unmanned underwater vehicle in simulation.

The work comprising this chapter was published in the 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).

- Charles Hartsell[1], **Shreyas Ramakrishna**[1], Abhishek Dubey, Daniel Stojcsics, Nagabhushan Mahadevan, and Gabor Karsai. "ReSonAte: A Runtime Risk Assessment Framework for Autonomous Systems." In Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 118-129. IEEE, 2021.

## 6.2 Introduction

**Problem Domain**: Autonomous CPSs are expected to handle uncertainties and self-manage the system operation in response to problems and increase in risk to system safety. This risk may arise due to distribution shifts [315], environmental context, or failure of software or hardware components [316]. System Risk Management [31] has been a well-known approach used to assess the system's operational risk. It involves design-time activities such as *hazard analysis* for identifying the system's potential hazards, *risk assessment* to identify the risk associated with the identified hazards, and a system-level *assurance case* [38] to argue the system's safety. However, the design-time hazard analysis and risk assessment information it uses is inadequate in highly dynamic situations at runtime. To better address the dynamic operating nature of CPSs, dynamic assurance approaches such as runtime certification [65], dynamic assurance cases [48], and modular safety certificates [206] have been proposed. These approaches extend the assurance case to include system monitors whose values are used to update the reasoning strategy at runtime. A prominent approach for utilizing the runtime information has been to design a discrete state-space model for the system, identity the risk associated with each possible state transition action, then perform the action with the least risk [222].

**State-of-the-art and Limitations**: The effectiveness of dynamic risk estimation has been demonstrated in the avionics [224] and medical [228] domains, but these techniques often encounter challenges with a state-space explosion which may limit their applicability to relatively low-complexity systems. Also, real-time CPSs often have strict timing deadlines in the order of tens of milliseconds requiring any dynamic risk assessment technique to be computationally lightweight. Besides, the dynamic risk assessment technique should also consider the uncertainty introduced by the LEC because of OOD data [93]. Assurance monitors [92, 93] are a type of OOD detector often used to tackle the OOD data problem. The output of these monitors should be considered when computing the dynamic risk.

[1]These authors have contributed equally. The entire BTD formalism was done by Charles Hartsell. The data generation mechanism, runtime monitors, and experimental evaluations was done by Shreyas Ramakrishna.

Recently, there has been a growing interest in the dynamic risk assessment of autonomous CPS. For example, the authors in [317] have used Dynamic Bayesian Networks to incorporate the broader effect of spatio-temporal risk gathered from road information on the system's operational risk. This paper introduces the Runtime Safety Evaluation in Autonomous Systems (ReSonAte) framework to perform a dynamic risk assessment of autonomous CPS. ReSonAte uses the design-time hazard analysis information to build Bow-Tie Diagram (BTD), which describes potential hazards to the system and how common events may escalate to consequences due to those hazards. The risk posed by these hazards can change dynamically since the frequency of events and effectiveness of hazard controls may vary based on the system's state and environment. To account for these dynamic events at runtime, ReSonAte uses design-time BTD models along with information about the system's current state derived from system monitors (e.g., anomaly detectors, assurance monitors, etc.) and the operating environment (e.g., weather, traffic, etc.) to estimate dynamic hazard rates. The estimated hazard rate can be used for high-level decision-making tasks at runtime to support the self-adaptation of CPSs.

**Contributions**: The specific contributions of this paper are the following. We present the ReSonAte framework and outline the dynamic risk estimation technique, which involves design-time measurement of the conditional relationships between hazard rates and the state of the system and environment. These conditional relationships are used at runtime with state observations from multiple sources to dynamically estimate system risk. Further, we describe a process that uses an extended BTD model for estimating the conditional relationships between the effectiveness of hazard control strategies and the system's state and environment. To improve scalability and reduce the amount of data required, this process considers each control strategy in isolation and composes several single-variate distributions into one complete multi-variate distribution for the control strategy in question. A key contribution is our scenario description language, which enables data collection by specifying prior distributions over threat conditions, environmental conditions, and initial conditions of the vehicle. We implement ReSonAte for an AV example in the CARLA simulator [111]. Through comprehensive simulations across 600 executions, we show a strong correlation between our risk estimates and eventual vehicular collisions. The dynamic risk calculations, on average, take only 0.3 milliseconds at runtime, in addition to the overhead introduced by the system monitors. Further, we exhibit ReSonAte's generalizability with preliminary results from an unmanned underwater vehicle (UUV) example.

**Outline**: The rest of this chapter is organized as follows. In Section 6.3, we present the related research. In Section 6.4, we present the ReSonAte framework and its operation. In Section 6.5, we demonstrate the utility of the proposed framework with an AV example in CARLA simulation and an underwater vehicle. Finally, we present our conclusion in Section 6.6.

## 6.3 Related Work

System health management [249, 70] with model-based reasoning has been used for self-adaptation of traditional CPSs. As discussed in [318], a pre-requisite in these approaches has been that the system has knowledge about itself, its objectives, and its operating environments, including the ability to estimate when the adaption should occur. One way to do this is to estimate the operational risk to the system at runtime. Estimating the runtime risk is more difficult in autonomous systems with LECs due to the black-box nature of the learning components and their susceptibility to distribution shifts and environmental changes.

The existing approaches for risk assessment can be broadly classified into pre-determined risk assessment and dynamic risk assessment approaches. While pre-determined risk assessment is meant to be a design-time strategy, its counterpart is more suited for runtime assessment [222]. Dynamic risk assessment has been used for safe reconfiguration in open adaptive systems such as automotive [222, 223], avionics [224], and medical-robotics [225]. Three main approaches to dynamic risk assessment are. The first approach involves designing a discrete state-space model for the system, identifying the risk associated with each possible state transition action, and acting with the least risk [222]. The second approach involves using different risk metrics at runtime. For example, time-to-collision and distance-to-collision are well-known risk metrics used in the automotive domain. If the time or distance to an object falls below a pre-selected threshold, the situation is considered risky, and a safe action is enforced [226]. However, the problem with this approach is that it is not so straightforward to define these metrics in some application domains like medical CPS [225].

Quantitative risk assessment using BTD models [58, 59, 60] is the third approach. Here, the causal structure of the BTD (See Fig. 6.2 in Section 6.4) is utilized in quantitatively computing the risk score. This requires computing the conditional probabilities of the Barrier events in the BTD from diverse data from the system's different operating conditions and failure modes. These probabilities are combined to compute the likelihood of a consequence. The likelihood is combined with the severity of the consequence to compute the risk score. Though widely used among the three approaches, there are several challenges in using this approach. *First*, BTD has mostly been used as a tool for visualizing the hazard propagation, and its static structure limits real-data updating, which is required for dynamic risk estimation[319]. *Second*, the conditional probability computation of BTD events needs to be derived from data. However, manual data collection and data uncertainty (e.g., non-availability or insufficiency of data) are the two major problems [97, 320]. Fuzzy set and evidence theory translate the developer's qualitative judgment of the BTD events into numerical quantities to compute quantitative risk scores [320, 321]. Bayesian approach [58, 59, 60] allows for both quantitatively computing the BTD event probabilities and updating these probabilities with new data. Though these approaches dynamically update the conditional probability of the BTD events with the system's latest

experience, they do not include runtime monitoring data to assess the operational risk.

In addition to these main approaches, there are several other approaches. A hierarchical self-management framework for online risk management is developed for a gas turbine aero-engine case study [224]. The different layers of the framework communicate to exchange runtime information from different monitors for synthesizing a dynamic safety case arguing the operational risk of the system. In another work [227], the authors have computed a risk score for different configurations of an adaptive system based on the associated services. The runtime score is computed as a product of the configuration's dynamic complexity and the severity level of the service, which are pre-determined during design time. [225] integrates runtime safety certificates and dynamic risk assessment to deal with the unpredictability in the cooperative behavior of collaborative medical CPS. The risk is computed by partitioning the situation space based on the risk of performing a specific behavior such as infusion or acceleration. [228] calculates a dynamic risk score using a Bayesian network based on a tree structure that finds the probability distribution nodes and relations. [75] uses reachability analysis to compute an autonomous system's dynamic risk. *First*, reachability analysis is used to find all the possible reachable states of the system. *Second*, for each reachable state, the probability of each hazard is combined with the severity of the hazard to compute the system's risk.

Although prior approaches have made BTDs suitable for quantitative risk estimation, the design-time hazard information used for risk estimation is inadequate for CPSs. The main focus of this work is to dynamically estimate risk by fusing design-time information captured in the BTDs with runtime information about the system and the environment. Additionally, we concentrate on generating large-scale simulation data for conditional probability estimation of the BTD events.

## 6.4 ReSonAte Framework

The goal of the ReSonAte framework is the dynamic estimation of *risk* based on runtime observations about the system's state and environment. We define risk as a product of the likelihood and severity of undesirable events, or *consequences*. Fig. 6.1 outlines the ReSonAte workflow, which is divided into design-time and runtime steps. Section 6.4.1 provide background information about each of these well-studied techniques. However, our BTD formalism described in Section 6.4.1 is distinct from existing formalisms with additional model attributes and restrictions for the ReSonAte framework. Section 6.4.2 through Section 6.4.4 introduce the risk calculation equations, the conditional probability estimation process, and the dynamic assurance case evaluation method of the ReSonAte framework respectively.

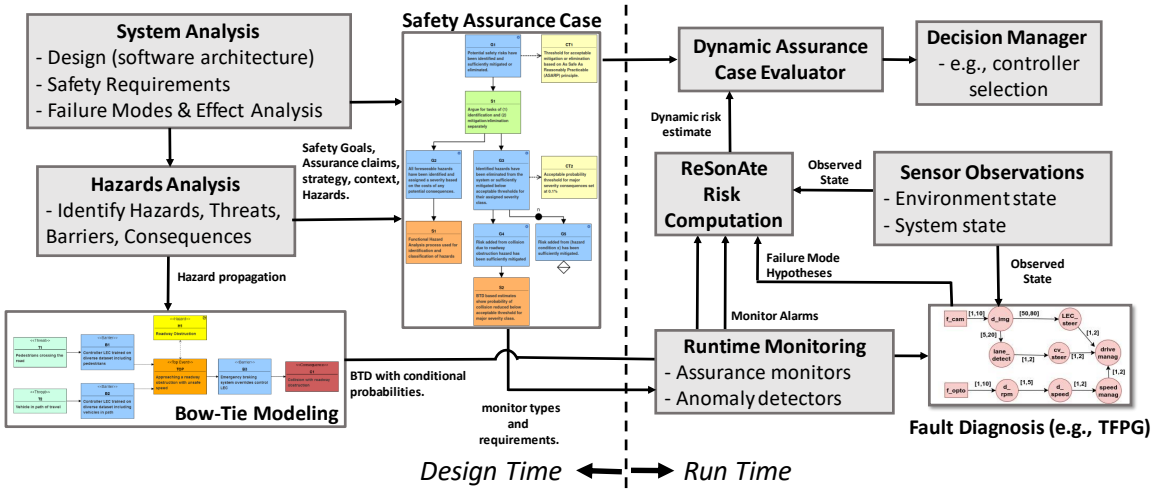



Figure 6.1: Overview of the steps involved in ReSonAte. Steps performed at design-time/run-time are shown on the left/right.

### 6.4.1 Bow-Tie Formalization & Extensions

Bow-Tie Diagrams are used in ReSonAte to describe hazard propagation paths and the control strategies used to prevent that propagation. BTDs are intended for describing linear propagation and do not have concepts for capturing non-linear causality or complex interactions between events. However, these linear models are sufficient for many hazards commonly encountered by CPSs, such as those demonstrated for the example systems described in Section 4.5. To perform dynamic risk estimation, the hazard models must also contain information about the expected rate of threat events and the success probability of barriers, which may be conditional on the current conditions. This section presents a BTD formalization that includes this conditional information not captured in existing BTD interpretations. Under our formalism, each node has an associated function conditional on the system's state and environment. Before constructing a BTD, we assume the following sets have been identified: all potential hazard classes $H$, all events $E$ relevant for hazard analysis, and barriers $B$ which may either *prevent* a loss of control or *recover* after such a loss has occurred, possible system failure modes $FM$, and event severity classes $SV$. We also assume a function $f_a$ which maps each severity class to the maximum acceptable rate of occurrence threshold $f_a : SV \to \mathbb{R}$ has been defined.

For risk calculation, we define the state of the system and the environment as $S = (F, e, m)$ where:

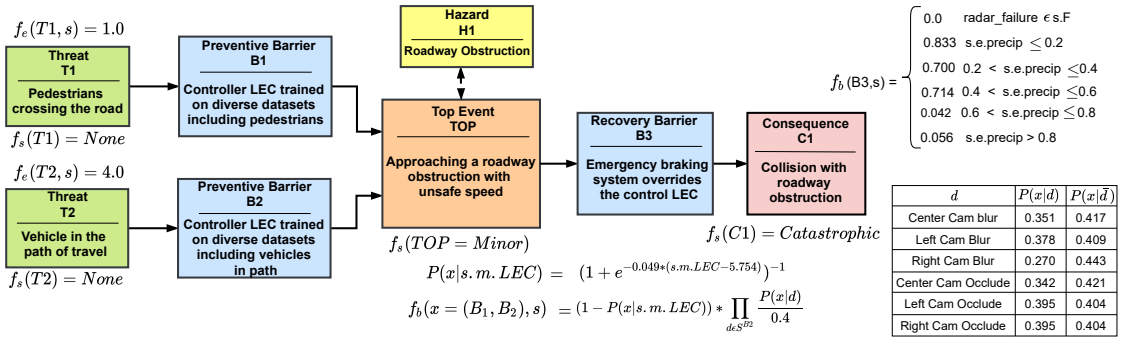
- $F$ is a set of failure modes currently present in the system. $F \subseteq FM$
- $e$ is an n-tuple describing the current environmental conditions where $n$ is the number of environmental parameters relevant for risk calculations, and each parameter may be continuous or discrete.
- $m$ is a k-tuple containing the outputs of runtime monitors in the system where $k$ is the number of such

monitors, and each monitor may be continuous or discrete-valued.

When modeling the environment, the system developer should choose an appropriate level of abstraction based on which environmental parameters are relevant for risk calculation. More environmental parameters may increase the model's fidelity but also increase the required effort when determining conditional probabilities for events and barriers. Considering every environmental parameter for all events and barriers is not required. Instead, each event or barrier may be conditional upon a smaller subset of environmental parameters that have the largest impact on that particular event or barrier. Formally, we define a BTD as a tuple $(N, C, f_t, h, f_d, f_b, f_e, f_s)$ where:

- $N$ is a set of nodes where each node represents either an event $e \in E$ or a barrier $b \in B$.
- $C$ represents a set of directed connections such that $C \subseteq N \times N$. For a given connection $c \in C$, $src(c)$ and $dst(c)$ represent the source and destination of $c$ respectively.
- The tuple $(N, C)$ is a directed acyclic graph representing the temporal ordering of events as well as the barriers which may break this ordering and prevent further propagation of events.
- The function $f_t$ gives the type of each node. $f_t : N \to \{event, barrier\}$. Using this function, we let $N_e = \{n \in N \mid f_t(n) = event\}$ and $N_b = \{n \in N \mid f_t(n) = barrier\}$. This gives the following properties: $N_e \subseteq E$, $N_b \subseteq B$, $N = N_e \cup N_b$, and $N_e \cap N_b = \emptyset$.
- $h \in H$ is the hazard class associated with the BTD.
- The function $f_d$ gives a textual description of each node. $f_d : N \to string$
- $f_b$ is a probability function conditional on the state of the system and environment defined for each barrier node. This function represents the probability that the barrier will successfully prevent further event propagation. $f_b : N_b \times S \to [0, 1]$.
- $f_e$ is a function conditional on the state of the system and environment defined for each event node. It gives the expected frequency of the event over a fixed period. The values of this function must be specified for *threat* events (i.e. root events with no preceding input events), but can be calculated for subsequent events in the diagram as discussed in Section 6.4.2. $f_e : N_e \times S \to [0, \text{inf})$.
- A function $f_s$ which maps each event to the appropriate severity class. $f_s : N_E \to SV$

BTDs are often constructed in a chained manner where a consequence event in one section of the BTD may serve as a threat or top event in a subsequent section of the same BTD. However, such a chained BTD can




| $d$ | $P(x\|d)$ | $P(x\|\bar{d})$ |
|---|---|---|
| Center Cam blur | 0.351 | 0.417 |
| Left Cam Blur | 0.378 | 0.409 |
| Right Cam Blur | 0.270 | 0.443 |
| Center Cam Occlude | 0.342 | 0.421 |
| Left Cam Occlude | 0.395 | 0.404 |
| Right Cam Occlude | 0.395 | 0.404 |

Figure 6.2: BTD for the "roadway obstruction" hazard of the AV example introduced in Section 6.5. Each block includes a brief description and the type of each node is denoted at the top of the block. The equations depicted are described in Section 6.4.1.

be broken into multiple, single-scope BTDs where each event has one unique type (i.e. threat, top event, or consequence). As a simplification, ReSonAte operates only on these single-scope BTDs which satisfy the additional restrictions listed below. Note that the symbol $\Rightarrow$ is used to denote precedence in the graph. That is, given two nodes $a, b \in N$, then $a \Rightarrow b$ states that $a$ precedes $b$ in the BTD, but this does not necessitate a direct edge such that $a \rightarrow b$. Instead, there may be any number of intermediate nodes $c_i \in N$ such that $a \rightarrow c_1 \rightarrow c_2 \rightarrow ... \rightarrow c_n \rightarrow b$.

- Exactly one event must be designated as the *top event*, denoted $e_{top}$.
- There must be at least one *threat*. i.e. $\exists\, t \in N_e \mid t \Rightarrow e_{top} \wedge \nexists\, n \in N \mid n \Rightarrow t$. We denote the set of all threat events as $N_t$.
- There must be at least one *consequence*. i.e. $\exists\, c \in N_e \mid e_{top} \Rightarrow c \wedge \nexists\, n \in N \mid c \Rightarrow n$. We denote the set of all consequence events as $N_c$.
- All events must be a threat, a top event, or a consequence. i.e. No intermediate events.
- All *barriers* must lie between a threat and the top event, or between the top event and a consequence. i.e. $\forall\, b \in N_b$ either $t \Rightarrow b \Rightarrow e_{top}$ or $e_{top} \Rightarrow b \Rightarrow c$
- No branching or joining of the graph is allowed, except for at the top event.

For the example BTD shown in Fig. 6.2, the type of each event is denoted on the top of the block and we can define the sets $N_e = \{T1, T2, TOP, C1\}$ and $N_b = \{B1, B2, B3\}$. For each event $e \in N_e$, the associated severity class is given by the function $f_s(e)$ which is shown under each event block. Each of the threats, $T1$ and $T2$, have been assigned to the "None" severity class because these events are considered to be common occurrences that do not result in any safety violation by themselves. The top event $TOP$ has been assigned to

the "Minor" severity class since this event can still be mitigated before a safety violation occurs but mitigation requires the automatic emergency braking system (AEBS) to override the primary LEC controller. Finally, the consequence $C1$ has been assigned to the "Catastrophic" severity class since this event is a safety violation and may result in significant damage to the system or environment.

The conditional functions $f_e$ and $f_b$ are shown near their respective nodes in Fig. 6.2. The functions $f_e(T1, \mathbf{s})$ and $f_e(T2, \mathbf{s})$ give the expected frequency of threats $T1$ and $T2$ in units of expected number of occurrences per minute, and their values were measured for our simulator configuration described in Section 4.5. The probability function for barriers $B1$ and $B2$, $f_b$(x = (B1,B2), **s**), shows how these barriers are dependent on both the continuous-valued output from the assurance monitor and on the binary state of other monitors. A sigmoid function, $P(x|\mathbf{s}.m.LEC)$, is used to capture the conditional relationship with the assurance monitor output. Finally, $f_b(B3, \mathbf{s})$ shows how barrier $B3$ is less likely to succeed as the precipitation increases and will not function in the case of a radar failure. Section 6.4.2 explains the generic process for conditional probability estimation and Section 6.5.1.4 provides more detail on the functions in this BTD.

### 6.4.2 Run-time Risk Computation

Each BTD includes functions describing the conditional frequency for all *threat* events and the conditional probability of success for all *barrier* nodes. However, the likelihood of each *consequence* is necessary to estimate the overall level of risk for the system. These probabilities can be calculated by propagating the initial threat rates through the BTD. When a particular event occurs, barrier nodes reduce the probability that the event will continue to propagate through the BTD based on the following equation:

$$R(e_2|s) = R(e_1|s)P(\overline{b_1} \wedge \overline{b_2} \wedge ... \wedge \overline{b_n}|s)$$
$$assume\ a \perp b\ \forall\ a,b \in \{b_1, b_2, ..., b_n\} \mid a \neq b$$
$$R(e_2|s) = R(e_1|s)[\Pi_{i=1}^{n} P(\overline{b_i}|s)] \tag{6.1}$$

Where, $e_1, e_2 \in N_e$ and $b_i \in N_b$ such that $e_1 \rightarrow b_1 \rightarrow b_2 \rightarrow ... \rightarrow b_n \rightarrow e_2$. $R(e_i|s)$ represents the frequency of event $e_i$ given state $s$. Eq. (6.1) makes the assumption that no barriers share any common failure modes and that the effectiveness of each barrier is independent from the outcome of other barriers. Similarly, $P(\overline{b_i}|s)$ represents the probability that barrier $b_i$ will fail to prevent event propagation given state $s$, i.e. $P(\overline{b_i}|s) = 1 - P(b_i|s)$. Letting $S$ represent the set of all states we are concerned with, then we can calculate the overall frequency of $e_2$ as $R(e_2) = \Sigma_{s \in S}[R(e_2|s)P(s)]$ where $P(s)$ is the probability of each particular state $s \in S$. As discussed in the BTD formalization in Section 6.4.1, no joining or splitting of paths is allowed in a BTD with the exception of the top event $e_{top}$. We treat the top event as a summation operation for all incoming edges, i.e.

any threat event may independently cause a top event if the associated barriers are unsuccessful. All outgoing edges from the top event are treated as independent causal chains, i.e. any potential consequence may occur from a top event, independent of other consequences in the BTD. The probability for the top event can be calculated as:

$$R(t^{top}) = \Sigma_{s \in S}[R(t|s)P(s)[\Pi_{i=1}^{n} P(\overline{b_i}|s)]]$$
$$R(e_{top}) = \Sigma_{t \in N_t} R(t^{top}) \tag{6.2}$$

where $R(e_{top})$ is the frequency for the top event and $P(t^{top})$ represents the contribution of each threat $t$ to this rate after passing through any intermediate prevention barriers $b_i \in N_b$ such that $t_i \rightarrow b_1 \rightarrow b_2 \rightarrow ... \rightarrow b_n \rightarrow e_{top}$. Finally, the probability of each consequence can be calculated with:

$$R(c_i) = R(e_{top})\Sigma_{s \in S}[P(s)[\Pi_{i=1}^{n} P(\overline{b_i}|s)]] \tag{6.3}$$

Where, $R(c_i)$ is the frequency of consequence $c_i \in N_c$ after passing through any recovery barriers $b_i \in N_b$ such that $e_{top} \rightarrow b_1 \rightarrow b_2 \rightarrow ... \rightarrow b_n \rightarrow c_i$.

If the state is not known uniquely, then each potential state $\mathbf{s} \in S$ must be enumerated and a probability function $P(\mathbf{s})$ must be assigned such that $\Sigma_{\mathbf{s} \in S} P(\mathbf{s}) = 1$. When only the expected distributions of the state variables are known at design time, this state probability function is typically calculated as a product of the probability mass functions (or probability density functions for continuous variables) of each state variable. Recall that for ReSonAte, the state $S$ is restricted to only those variables that have a conditional impact on the functions contained in the BTDs. Thus, not all state variables describing the system must be considered in this calculation. When the system is deployed at runtime, observations about the current state can be used to refine the set of prior probabilities $P(s)$ and dynamically calculate current risk values. The equations outlined here use a discrete treatment of probability, but continuous distributions can also be used where an appropriate integration replaces summation operations. If the state can be identified uniquely to a particular state $\mathbf{s}_0 \in S$, then we can assign $P(\mathbf{s}_0) = 1$ and simplify the risk equations. For example, we could calculate the rate of occurrence for events $TOP$ and $C1$ which are part of the BTD shown in Fig. 6.2 using the following equations:

$$R(TOP|s_0) = B.f_e(T1, s_0) * (1 - B.f_b(B1, s_0))$$
$$+ B.f_e(T2, s_0) * (1 - B.f_b(B2, s_0))$$

$$R(C1|s_0) = R(TOP|s_0) * [1 - B.f_b(B3, s_0)] \tag{6.4}$$

### 6.4.3 Estimating Conditional Relationships

#### 6.4.3.1 Conditional Relationships

In ReSonAte, both the rate of occurrence of events and the effectiveness of barriers may be conditionally dependent on the state including system failure modes, runtime monitor values, and environmental conditions. For each of these categories, it is necessary to identify the factors which should be examined for their impact on this conditional relationship. A failure analysis should be performed using an appropriate failure modeling language such as the timed failure propagation graph (TFPG). Similarly, the environmental parameters relevant to the system must be identified and the operating environment defined in terms of bounds and expected distributions of these parameters. We assume the environmental conditions are known uniquely and provided to ReSonAte. Monitor values that may impact the conditional relationships should also be identified.

For some nodes in a BTD, it may be possible to analytically derive the appropriate conditional relationship with each state variable, but often this relationship must be inferred from data. In this section, we consider a generic threat to the top event chain with a single barrier described as $t \rightarrow b \rightarrow e_{top}$. The contribution to the rate of the top event $e_{top}$ from this singular threat $t$ with a single barrier $b$ is shown in Eq. (6.5). Normally, multiple threats can lead to the top event and the contribution of all threats must be considered as described in Eq. (6.2). However, if all other threat conditions can be eliminated, then the top event may only occur as a result of threat $t$. In this case, the probability of success for barrier $b$ can be calculated as a function of the ratio of frequencies between the top event and threat $t$ shown in Eq. (6.6). This approach can also be used for threats with multiple associated barriers by considering each barrier in isolation since individual barriers are assumed to be independent in Eq. (6.1).

$$R(e^t_{top}|\mathbf{s}) = R(t|\mathbf{s}) * (1 - P(b|\mathbf{s})) \tag{6.5}$$

$$R(t_j|\mathbf{s}) = 0 \; \forall \; t_j \in N_t \mid t_j \neq t \rightarrow R(e_{top}|\mathbf{s}) = R(e^t_{top}|\mathbf{s})$$

$$P(b|\mathbf{s}) = 1 - [R(e_{top}|\mathbf{s})/R(t|\mathbf{s})] \tag{6.6}$$

We isolate individual threats in simulation using a custom SDL, described in Section 6.4.3.2, to generate scenarios where only the one threat of interest is allowed to occur and all other possible threats are eliminated. Each time the threat $t$ is encountered, the top event $e_{top}$ may either occur or not occur. If the top event does not occur, then the associated barrier $b$ was successful - i.e. prevented hazard propagation along this path. Otherwise, the barrier was unsuccessful. Since the occurrence of a threat is a discrete event that results in a boolean outcome (i.e. top event does/does not occur), the barrier effectiveness can be modeled as a conditional Binomial distribution where the probability of barrier success is dependent on the ratio of top event frequency

to threat frequency as shown in Eq. (6.6).

For each barrier $b$ in the BTD, any state variables which are likely to impact the conditional frequency or probability of that node should be identified, and we denote this reduced set of state variables as $S^b$. For discrete state variables (e.g., presence or absence of a particular fault condition, urban/rural/suburban environment, etc.), this ratio can be estimated using Laplace's rule of succession [322] as shown in Eq. (6.7) where $n_{s_i=a}$ is the number of scenes where the state variable $s_i$ had the desired value $a$ and $k_{s_i=a}$ is the number of such scenes where the top event also occurred. This equation can be applied for each of the possible values of the state variable $s_i$ to estimate the discrete probability distribution $P(b|s_i)$, and the process can then be repeated for each relevant state variable $s_i \in S^c$. Eq. (6.8) can be used to fuse the estimated distributions of each individual state variable into one multivariate distribution $P(b|\mathbf{s})$. Similar to Naive Bayes classifiers, this equation assumes each of the state variables $s_i$ are mutually independent conditional on the success of barrier $b$. If a stronger assumption is used that the state variables in $S^b$ are mutually independent, then the term $\Pi_{j=0}^{m}[P(s_j)]P(s_0 s_1 ... s_m)^{-1}$ reduces to 1 as was the case for all of the probabilities estimated in our examples. For each continuous state variable $s_j$ (e.g. output of an assurance monitor), maximum likelihood estimation was used in place of Eq. (6.7) to estimate $P(b|s_j)$. Similarly, Eq. (6.8) can be revised for continuous values by replacing the probability mass functions $P(s_j)$ with probability density functions $p(s_j)$.

$$P(b|s_i = a) = 1 - \frac{k_{s_i=a} + 1}{n_{s_i=a} + 2} \tag{6.7}$$

$$P(b|\mathbf{s}) = \frac{P(s_0, s_1, ..., s_m|b)P(b)}{P(s_0, s_1, ..., s_m)}$$

$$assume\ s_j \perp s_k \mid b\ \forall\ s_j, s_k \in S^b \mid s_j \neq s_k$$

$$P(b|s_0, s_1, ..., s_m) = \frac{P(b)\Pi_{j=0}^{m} P(s_j|b)}{P(s_0, s_1, ..., s_m)}$$

$$P(b|s_0, s_1, ..., s_m) = \frac{\Pi_{j=0}^{m} P(b|s_j)P(s_j)}{P(b)^{m-1}P(s_0 s_1 ... s_m)} \tag{6.8}$$

While the conditional probability estimation process is outlined here for prevention barriers, it may be modified for recovery barriers by replacing occurrences of any threats $t$ with the top event $e_{top}$ and replacing occurrences of the top event with each consequence of interest.

#### 6.4.3.2 Scenario Description Language

Description and generation of scenarios that cover the full range of expected operating environments is an important aspect of the design of CPS. As discussed in the previous chapter, several domain-specific scenario description languages (SDLs) such as Scenic [287] and MSDL [153] with probabilistic scene generation capabilities are available. While these languages have powerful scene generation capabilities, they are targeted

```
scene sample {
    type string
    type int
    entity town_description{
        id:string
        map:string  }
    entity weather_description{
        cloudiness: uniform
        precipitation: uniform
        precipitation_deposits: uniform  }
    entity uniform{
        low: int
        high: int  }
    }
```

Figure 6.3: This listing shows a fragment of a CARLA scene description that was generated using our SDL written in textX meta language.

specifically at the automotive domain. We have developed a simplified SDL using the textX [288] meta language to generate varied scenes for multiple domains including our AV and UUV systems.

A fragment of the scene description for the CARLA AV example is shown in Fig. 6.3. A scene S = $\{e_1,e_2,...,e_i\}$ is described as a collection of entities (or set points), with each entity representing information either about the ego vehicle (e.g., type, route, etc.), or the operating environment (e.g., weather, obstacle). Further, each of these entities has parameters whose value can be sampled using techniques such as Markov chain Monte Carlo to generate different scenes in the simulation space. The larger the number of sampling, the wider is the simulation space coverage. For the AV example, our scene was defined as S = {town_description, weather_description, av_route}. While the parameters of town_description and av_route remained fixed, the parameters of weather_description such as cloud, precipitation, and precipitation deposits were randomly sampled to take a value in [0,100]. We generated 46 different CARLA scenes by randomly sampling the weather parameters, a few of which were used for estimating the conditional relationships.

Currently, we perform unbiased sampling to generate each scenario, then use the resulting unbiased dataset for the probability calculations described in Section 6.4.3. This approach proved sufficient for the example systems described in Section 6.5. However, these systems are prototypes where consequences occur relatively often. For more refined production systems, consequences do not typically occur under nominal operation but instead are often the result of rare combinations of adverse operating conditions and/or system failure modes. This is an example of the long tails problem where the probability of observing this undesired system behavior is low if unbiased random sampling is used. In future work, guided sampling of the state space (e.g., [323]) can be used to better observe rare events and perform conditional probability estimation.

Figure 6.4: Screenshots from each autonomous system simulation. The left figure shows an image from the forward-looking camera of the AV as it navigates through the city. The right figure shows a top-down view of the UUV where the vehicle (trajectory shown as a green line) is inspecting a pipeline (thick blue line) until an obstacle (grey box) is detected by the forward-looking sonar and the vehicle performs an avoidance maneuver.

### 6.4.4 Dynamic Assurance Case Evaluation

System Risk Management [31] is a common technique used in the system safety assurance process which involves identification of potential hazards, analysis of the risks posed by those hazards, and reduction of these risks to acceptable levels. The amount of risk remaining after risk control strategies have been implemented is known as the *residual risk*. An appropriate assurance argument pattern, such as the As Low As Reasonably Practicable (ALARP) pattern outlined by Kelly [176], is often used to document the means used for risk reduction and show that the estimated levels of residual risk are within tolerable bounds. With traditional SRM techniques, residual risk estimates are static results of design-time analysis techniques. However, using the risk estimated by ReSonAte the residual risk for each hazard can be updated dynamically at run-time and the associated goal in the assurance argument can be invalidated if the risk exceeds a predefined threshold. When this risk threshold is violated, contingency plans can be enacted to place the system in a safe state. For example, stopping the AV or surfacing the UUV are simple contingency actions used for the example systems described in Section 6.5. The risk scores produced by ReSonAte may also be used in assurance case adaptation techniques such as Dynamic Safety Cases [48] or ENTRUST [53].

## 6.5 Evaluation

We evaluate ReSonAte using an AV example in the CARLA simulator [111] and show its generalizability with preliminary results from an UUV example [82]. Fig. 6.4 shows the screenshots from these demonstration platforms. The experiments[2] in this section were performed on a desktop with AMD Ryzen Threadripper 16-Core Processor, 4 NVIDIA Titan Xp GPU's and 128 GiB memory.

[2]source code to replicate the CARLA AV experiments can be found at: https://github.com/scope-lab-vu/Resonate

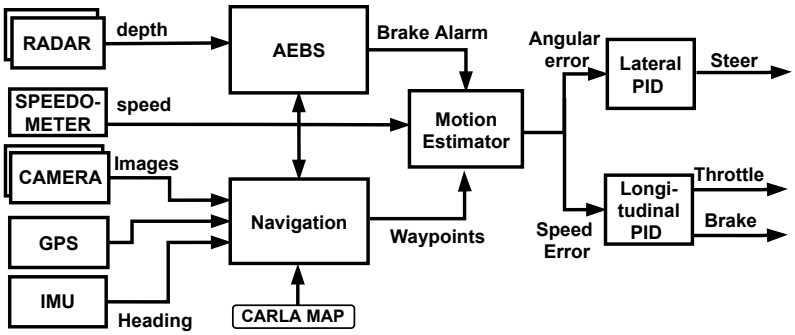



Figure 6.5: A block diagram of our AV example in CARLA simulation.

### 6.5.1 Autonomous Ground Vehicle

#### 6.5.1.1 System Overview

Our first example system is an autonomous car that must safely navigate through an urban environment while avoiding collisions with pedestrians and other vehicles in a variety of environmental and component failure conditions. The architecture of our AV, shown in Fig. 6.5, relies on a total of nine sensors including two forward-looking radars, three forward-looking cameras, a global positioning system (GPS) receiver, an inertial measurement unit (IMU), and a speedometer. The "navigation" LEC, adapted from previous work [5], produces waypoints for the desired position and velocity of the vehicle at a sub-meter granularity using a neural network for image processing along with higher-level information about the desired route provided by a map. These waypoints are passed to the "motion estimator" which computes throttle and steering angle error between the current and desired waypoints. The motion estimator also serves as a supervisory controller which will override the primary navigation component if an alarm is sent by the AEBS safeguard component. This AEBS component will raise an alarm when the vehicle's safe stopping distance, estimated based on the current vehicle speed, exceeds the distance returned by the radars indicating that the AV is approaching an object at an unsafe speed. Finally, the output of the motion estimator is sent to two PID controllers which generate appropriate steering, throttle, and brake control signals.

#### 6.5.1.2 System Analysis

We start with the analysis of the AV system and its operating environment (e.g., weather, traffic, etc). As the AV primarily uses a perception LEC, parameters such as weather conditions (e.g., cloud spread, precipitation level, and precipitation deposit level), high brightness, and camera-related faults such as blur and occlusion influenced the control actions generated by the LEC controller. To detect the camera faults and adverse operating conditions, we have designed several monitors. Each of the three cameras is equipped with

OpenCV-based blur and occlusion detectors to detect image distortions. The blur detector uses the variance of the laplacian [324] to quantify the level of blur in the image where a high variance indicates that the image is not blurred, while a low value ($<$30) indicates the image is blurred. The occlusion detector is designed to detect continuous black pixels in the images, with the hypothesis that an occluded image will have a higher percentage of connected black pixels. In this work, a large value ($>$15%) of connected black pixels indicated an occlusion. The blur detector has an F1 score of 99% and the blur detector has an F1 score of 97% in detecting the respective anomalies.

Additionally, we leverage our previous work [93] to design a reconstruction based $\beta$-VAE assurance monitor for identifying changes in the operating scenes such as high brightness. The $\beta$-VAE network has four convolutional layers $32/64/128/256$ with (5x5) filters and (2x2) max-pooling followed by four fully connected layers with 2048, 1000, and 250 neurons. A symmetric deconvolutional decoder structure is used as a decoder, and the network uses hyperparameters of $\beta$=1.2 and a latent space of size=100. This network is trained for 150 epochs on 6000 images from CARLA scenes of both clear and rainy scenarios. The reconstruction mean squared error of the $\beta$-VAE is used with Inductive Conformal Prediction [325] and power martingale [280] to compute a martingale value. The assurance monitor has an F1 score of 98% in detecting operating scenes with high brightness. Further, TFPG models for the identified camera faults were constructed, but the TFPG reasoning engine was not used since the available monitors were sufficient to uniquely isolate fault conditions without any additional diagnostic procedures.

#### 6.5.1.3 Hazard Analysis and BTD Modeling

For our AV example, we consider a single hazard of a potential collision with roadway obstructions. The potential threats for this hazard were identified to be pedestrians crossing the road (T1), and other vehicles in the AV's path of travel (T2). The top event was defined as a condition where the AV is approaching a roadway obstruction at an unsafe speed. The primary LEC is trained to safely navigate in the presence of either of these threats and served as the first hazard control strategy for both threat conditions, denoted by prevention barriers B1 and B2. Finally, the AEBS served as a secondary hazard control strategy denoted by the recovery barrier B3. The BTD shown in Fig. 6.2 was constructed based on these identified events and control strategies. For full-scale systems, the BTD construction process will usually result in the identification of a large number of events and barriers modeled across many BTDs. For our example, we have restricted our analysis to these few events and barriers contained in a single BTD.

#### 6.5.1.4 Conditional Relationships

Here we consider the barrier node B2 in Fig. 6.2 as an example for the probability calculation technique described in Section 6.4.3. Barrier B2 is part of the event chain $T2 \rightarrow B2 \rightarrow TOP$ and describes the primary control system's ability to recognize when it is approaching a slow-moving or stopped vehicle in our lane of travel (T2) too quickly and slow down before control of the roadway obstruction hazard (H1) is lost. To isolate barrier B2, simulation scenes were generated using our SDL where T2 was the only threat condition present but all other system and environmental parameters were free to vary. It is important for these scenes to cover the range of expected system states and environmental conditions since the resulting dataset is used to estimate the conditional probability of success for barrier B2. A total of 300 simulation scenarios were used for estimating the probability of barrier B2. As a convenience for our example, the threat (T2) was set to occur once during each scene, regardless of other state parameters, which allows the denominator of this ratio to be set as $R(T2|\mathbf{s}) = 1$ occurrence per scene.

As perception LEC is the primary controller, the success rate of barriers B1 and B2 will be dependent on the image quality. The image quality will in turn be dependent on image blurriness, occlusion, and environmental conditions. While the level of blur and occlusion for each camera is a configurable simulation parameter, our example system is not provided this information and instead relies on blur and occlusion detectors for each of the three cameras as discussed in Section 6.5.1.2. Each detector provides a boolean output indicating if the level of blur or occlusion exceeds a fixed threshold. The primary LEC is also susceptible to OOD data, and an assurance monitor was trained for this LEC to detect such conditions.

This results in barrier $B2$ being dependent on a total of 7 state-variables described as the set $S^{B2}$. For the 6 boolean variables, Laplace's rule of succession shown in Eq. (6.7) was applied to the simulation dataset resulting in probability table in the lower-right section of Fig. 6.2. A sigmoid function was chosen to model the conditional relationship with the continuous assurance monitor output. Maximum likelihood estimation was used to produce the function $P(x|\mathbf{s}.m.LEC)$ shown in Fig. 6.2 where $\mathbf{s}.m.LEC$ represents the output of the LEC assurance monitor. Each of these single variable functions was combined into the multivariate conditional probability distribution $f_b(B2, \mathbf{s})$ using Eq. (6.8). Note that the LEC was observed to be similarly effective in identifying pedestrians and vehicles, and a simplifying assumption was made to use the same conditional probability function for both barriers B1 and B2 given by function $f_b$(x = (B1,B2), $\mathbf{s}$) in Fig. 6.2.

The AEBS described by barrier B3 is independent of the camera images and relies on the forward-looking radar. The level of noise in our simulated radar sensor increased with increasing precipitation levels, indicating that the effectiveness of barrier B3 would likely decrease as precipitation increased. Also, failure of the radar sensor may occur on a random basis which reduces the effectiveness of the AEBS. A similar conditional

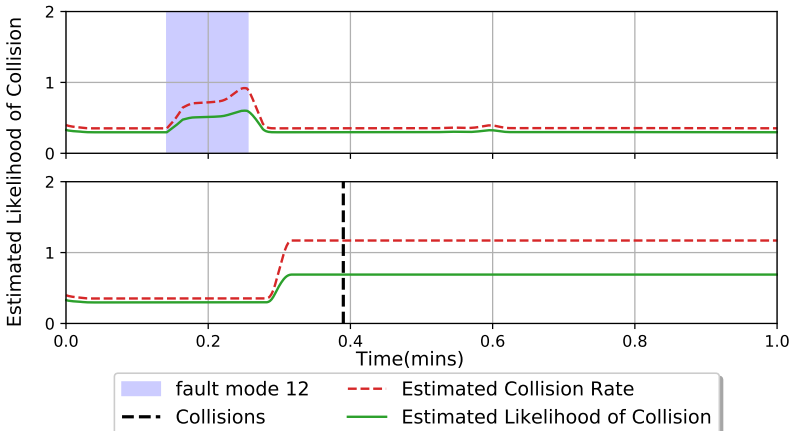



Figure 6.6: ReSonAte estimated collision rate and the likelihood of a collision for 2 validation scenes. (Top) Scene1 - nominal scene with good weather and an intermittent occlusion fault for left and center cameras. (Bottom) Scene2 - initially nominal scene until 27 seconds when image brightness is increased. A collision occurs at 38 seconds denoted by the vertical dotted line.

estimation process as used for B2 was applied here to calculate the function $f_b(B3, \mathbf{s})$ shown in Fig. 6.2.

#### 6.5.1.5 Results

To validate the ReSonAte framework, the AV was tasked to navigate 46 different validation scenes that were generated using our SDL. In these scenes, the weather_description parameters of cloud (c), precipitation (p), and precipitation deposits (d) were varied in the range [0,100]. Other adversities were synthetically introduced using OpenCV including increased image brightness, camera occlusion (15%-30% black pixels), and camera blur (using 10x10 Gaussian filters). During each simulation, the ReSonAte's risk calculations continuously estimate the hazard (or collision) rate $h(t)$ based on changing environmental conditions, the presence of faults, and outputs from the runtime monitors. The frequency of the risk estimation can be selected either based on the vehicle's speed, environmental changes, or available compute resources. In our experiments, we estimate the risk of the system every inference cycle.

Fig. 6.6 shows the estimated collision rate and the likelihood of collision of the AV across 2 validation scenes. The occurrence of a collision can be described as a random variable following a Poisson distribution where the collision rate is the expected value $\lambda$. The likelihood of collision can then be computed as $(1 - e^{-\lambda \cdot t})$, where $\lambda$ is the estimated collision rate and $t$ is the operation time (fixed to 1 minute in our experiments).

Fig. 6.7 shows the estimated average collision rate plotted against the observed collisions for 6 validation scenes described in the figure caption. The estimated average collision rate is calculated as $\frac{\int_{T_1}^{T_2} h(t)\,dt}{T_2 - T_1}$, where, h(t) is the estimated collision rate, $T_1 = 0$ and $T_2 = 1$ minute for our simulations. A moving average is used to smooth the estimated collision rate and a window size of 20 was selected to balance the desired smoothing

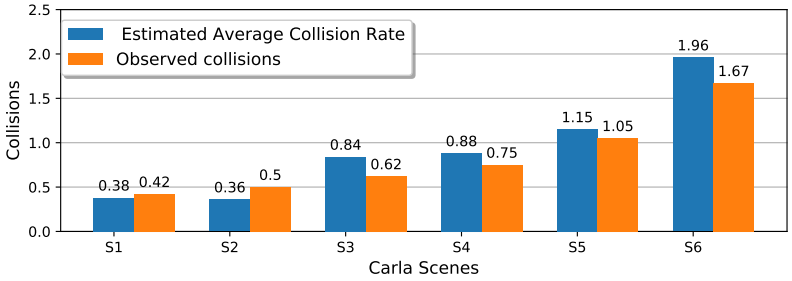
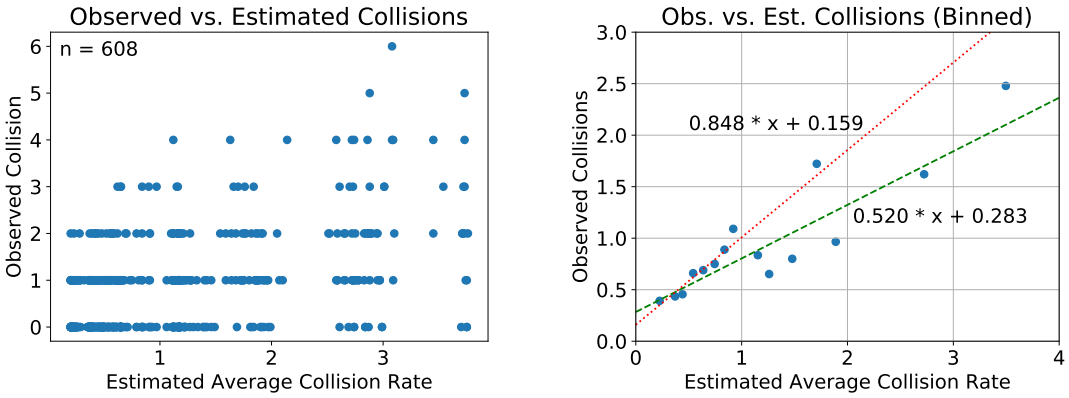



Figure 6.7: ReSonAte estimated average collision rate vs. observed collisions compared across 6 validation scenes. The results are averaged across 20 simulation runs for each scene. Each subsequent scene represents increasingly adverse weather and component failure conditions.




(a) Validation data scatter plot.

(b) Binned validation data.

Figure 6.8: Results from validation scenes for the ReSonAte framework. Each data point shown in Fig. 6.8a represents the outcome of one simulated scene with the actual number of collisions observed plotted against the estimated average collision rate. These same data points have been divided into bins and averaged in Fig. 6.8b, along with two least-squares fit trend lines.

against the delay incurred by the moving average. An overall trend in the plot shows a strong correlation between the actual and the estimated collisions. Also, a visible trend is that the collision rate changes with the weather patterns, increasing for adverse weather conditions (S5-S6). However, the estimated risk tends to be slightly overestimated when the collision rate is greater than 1.0.

Further, each of the 608 data points shown in Fig. 6.8a represents the outcome of one simulated scene with the actual number of collisions plotted against the average collision rate estimated by ReSonAte. Since the occurrence of a collision is a probabilistic event, there is significant variation in the actual number of collisions observed in each scene. Using our dynamic rate calculation approach, the rate parameter $\lambda$ of the Poisson distribution changes for each scene as shown on the x-axis of Fig. 6.8a. Maximum likelihood analysis was used to compare our dynamic approach against a static, design-time collision rate estimate where $\lambda$ is fixed for all scenes. The observed average collision rate across all scenes was found to be 0.829 collisions per minute. Using this static $\lambda$ value gave a log-likelihood of -740.7 while the dynamically updated $\lambda$ resulted

| System Configuration | GPU | | CPU | | Execution Time (s) |
|---|---|---|---|---|---|
| | Util (%) | Mem (%) | Util (%) | Mem (%) | |
| **LEC only** | 14.3 | 47.2 | 17.6 | 10.4 | 0.024 |
| **LEC + Runtime Monitors** | 14.7 | 86.5 | 18.5 | 10.5 | 0.217 |
| **LEC + Monitors + ReSonAte** | 16.1 | 87.0 | 18.7 | 10.4 | 0.218 |

Table 6.1: Resource requirements and execution times for different configurations of the system. Average values computed across 20 simulation runs.

in a log-likelihood of -709.8 for a likelihood ratio of 30.9 in favor of our dynamic approach. Note that the static collision rate used here is a posterior estimate calculated from the true number of collisions observed. Any static risk estimate made without this post facto knowledge would result in a lower likelihood value and further increase the gap between the dynamic and static approaches.

To better show the correlation between estimated and actual collisions, the same data points have been divided into bins and averaged in Fig. 6.8b along with two least-squares fit trend lines. The dashed green trend line shows a linear fit to the complete data set while the dotted red line shows a linear fit to only those data points where the average estimated collision rate was less than or equal to 1.0. Both trend lines show a strong positive correlation between the estimated collision rate and the number of observed collisions, but the dotted red line more closely resembles the desired 1-to-1 correspondence between estimated and actual collisions (i.e. slope of 1). These results indicate that our dynamic risk calculation tends to overestimate when the estimated collision rate is greater than 1.0.

Table 6.1 shows the resource requirements and execution times for the AV system with different configurations. As seen from the shaded columns, the system monitors, particularly the resource-intensive $\beta$-VAE monitor, increases the GPU memory used by $\sim 40\%$ and execution time by $\sim 0.2$ seconds. However, the ReSonAte risk calculations require minimal computational resources taking only 0.3 milliseconds.

### 6.5.2 Unmanned Underwater Vehicle

In this section, we discuss the primary results of applying ReSonAte to a UUV testbed based on the BlueROV2 [82] vehicle. The testbed is built on the ROS middleware [326] and simulated using the Gazebo simulator environment [151] with the UUV Simulator [327] extensions.

#### 6.5.2.1 System Overview and BTD Modeling

In this example, the UUV was tasked to track a pipeline while avoiding static obstacles (e.g., plants, rocks, etc.) as shown in Fig. 6.4. The UUV is equipped with six thrusters, a forward-looking sonar (FLS),

2 side-looking sonars (SLS), an IMU, a GPS, an altimeter, an odometer, and a pressure sensor. The vehicle has several ROS nodes for performing pipe tracking, obstacle avoidance, degradation detection, contingency planning, and control. Additional contingency management features such as thruster reallocation, return-to-home, and resurfacing are available. The UUV uses the SLS along with the FLS, odometry, and altimeter to generate the HSD commands for pipe tracking and obstacle avoidance. The degradation detector is a LEC that employs a feed-forward neural network to detect possible thruster degradation. This LEC sends information to the contingency manager including the identifier of the degraded thruster, level of degradation, and a value measuring confidence in the predictions. The contingency manager may then perform a thruster reallocation (i.e. adjustment of the vehicle control law) if necessary to adapt to any degradation. A ROS node implementation of ReSonAte was used to dynamically estimate and publish the likelihood of a collision.

#### 6.5.2.2 Hazard Analysis and BTD Modeling

Using the system requirements and its operating conditions we identified UUV operating in presence of static obstacles as the hazard condition which could result in the UUV's collision. The static obstacle which appears at a distance less than the desired separation distance of 30 meters is considered to be a threat in this example. Further, the static obstacle appearing at a distance less than a minimum separation distance of 5 meters is considered to be a TOP event for the BTD.

#### 6.5.2.3 Conditional Relationships

The probability estimation method described in Section 6.4.3 was used to compute the conditional probabilities for the BTD. We used data from 350 simulation scenarios for the conditional probability calculations. These simulations were generated by varying parameters such as the obstacle sizes (cubes with lengths of 0.5,1,2,5, and 10 meters), the obstacle spawn distance (value in range the range of 5 to 30 meters), and random thruster1 failures that were synthetically introduced by varying the efficiency in the range 0 to 60.

#### 6.5.2.4 Results

Fig. 6.9 shows the ReSonAte estimated likelihood of collision. The efficiency of thruster 1 degrades to 60% at 45 seconds, and the estimated likelihood of a collision increases to 0.25. Soon after, the contingency manager performs a thruster reallocation to improve the UUV's stability, the likelihood of collision decreases to 0.15. We are currently validating the estimated likelihood across large simulation runs.

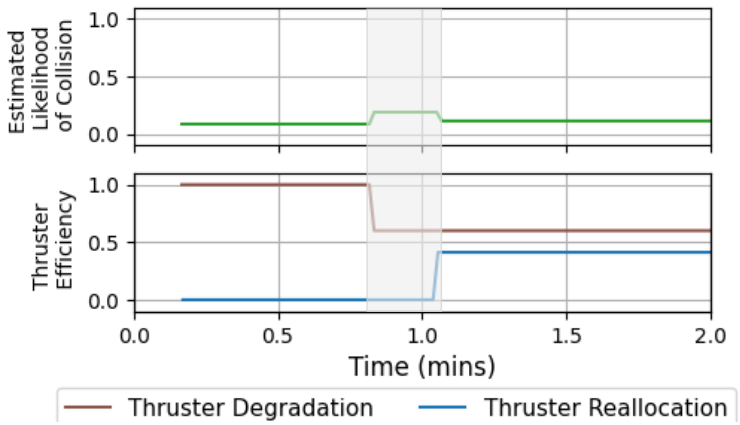



Figure 6.9: ReSonAte estimated likelihood of a collision for the BlueROV2 example. As seen in the gray shaded region, the likelihood of collision increases when thruster degradation occurs, then reduces after thruster control reallocation.

## 6.6 Conclusions and Future Work

ReSonAte captures design-time information about system hazard propagation, control strategies, and potential consequences using BTDs, then uses the information contained in these models to calculate risk at runtime. The frequency of threat events and the effectiveness of hazard control strategies influence the estimated risk and may be conditionally dependent on the system's state and operating environment. A technique for measuring these conditional relationships in simulation using a custom SDL was demonstrated. ReSonAte was then applied to an AV example in the CARLA simulator to assess the risk of collision dynamically. A strong correlation was found between the estimated likelihood of a collision and the observed collisions. Additionally, ReSonAte's risk calculations require minimal computational resources, making it suitable for resource-constrained and real-time CPSs. Future extensions and applications for ReSonAte include: (1) dynamic estimation of event severity in addition to event likelihood, (2) inclusion of state uncertainty into risk calculations to produce confidence bounds on risk estimates, (3) forecasting future risk based on expected changes to system or environment, (4) use of estimated risk for higher-level decision making such as controller switching or enactment of contingency plans.

To conclude, while the detector and risk assessment components discussed in Chapter 5 and the current chapter identifies the occurrence of the system hazard and associates it with a risk level, they do not provide a mechanism to mitigate the identified hazard. Therefore, to complete the proposed dynamic safety assurance approach, we need an adaptive mitigation strategy that can select a suitable action to mitigate the identified problem. An open question that we must address is: *how can we design an adaptive decision strategy that selects a suitable mitigation action to mitigate the system's risk?*

# Chapter 7

# Mitigating the Risk: Blended Simplex Strategy

## 7.1 Overview

The decisive step of our dynamic safety assurance framework is the mitigation component needed for selecting a suitable control action that alleviates or mitigates the risk posed to the system. Techniques like controller redundant architectures (simplex architecture [62] and controller sandboxing [98]) and system health management techniques [69, 2, 70] have been used to mitigate faults in complex cyber-physical systems (CPSs) like F-16 aircraft [99], unmanned aerial vehicles [100], and remote-controlled cars [101]. Especially, the simplex architecture has been widely utilized in assuring the safety of CPSs with unverified controllers. These architectures have become even more relevant in the context of autonomous CPSs with LECs, which are hard to verify and test.

The architecture augments a safety controller and a decision logic for systems with unverified and unsafe controllers. The logic arbitrates the control to the safety controller when the system is on the verge of entering an unsafe state. In this regard, the logic is designed with a safety objective using traditional verification techniques such as linear matrix inequality [63] and reachability analysis [98]. While these approaches have shown promising results in several applications [99, 100, 101, 102], there are two major challenges in using them. First, it is not easy to design an appropriate decision logic to maintain the system's safety and mission-critical objectives. In case proper care is not taken in designing the logic, it will be overly conservative in always selecting the safety controller. This conservative nature will affect the system's mission-critical objectives. Second, switching from one controller to the other is often performed instantaneously based on the logic's decisions, which affects the system's stability.

To handle these challenges, this chapter presents an extension to the conventional simplex architecture called the "weighted simplex strategy", which performs a weighted blending of the two controller's actions. The idea of blending is to avoid the immediate control transition problem of the conventional simplex approach. The decision logic in this approach needs to find the optimal weights rather than an optimal controller. The weight selection problem is formulated as Markov Decision Process, which is solved using reinforcement learning (RL). Further, to reduce conservatism, the RL-based logic is trained with a two-objective reward function that includes the system's safety and mission-critical objectives. The feasibility of the proposed strategy is demonstrated using the DeepNNCar [80] autonomous vehicle testbed.

The work comprising this chapter has been published in the Journal of Systems Architecture [120]. This builds on [80] which was published in the 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing.

- **Shreyas Ramakrishna**, Charles Hartsell, Matthew P. Burruss, Gabor Karsai, and Abhishek Dubey. "Dynamic-weighted simplex strategy for learning enabled cyber physical systems." Journal of systems architecture 111 (2020): 101760.
- **Shreyas Ramakrishna**, Abhishek Dubey, Matthew P. Burruss, Charles Hartsell, Nagabhushan Mahadevan, Saideep Nannapaneni, Aron Laszka, and Gabor Karsai. "Augmenting learning components for safety in resource constrained autonomous robots." In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 108-117. IEEE, 2019.

## 7.2 Introduction

**Problem Domain**: Autonomous cyber-physical systems (CPSs) have increasingly started relying on learning enabled components (LECs) as part of the control loop, performing varied perception-based autonomy tasks. These data-driven components are trained using machine learning approaches like machine learning (ML) [10], and reinforcement learning (RL) [11]. These approaches have given these components the capability to continuously learn and work in unfamiliar environments. Recently, these components have illustrated exceptional performance in a variety of challenging problems such as urban driving [4], object detection and tracking [328], and image segmentation [329]. However, recent accidents such as Tesla's autopilot crashes [23] and the fatal Uber self-driving car accident [24] demonstrate that these systems can still fail, often with devastating consequences.

While several design-time approaches like the assurance case [38], verification and validation [32, 33], and testing [34, 35], have been available for conventional CPSs, applying them to autonomous systems is difficult because of the following reasons. First, the black-box nature of LECs limits testing coverage. Pei, Kexin, *et al.* [26] discuss the limitations of performing exhaustive testing of LECs and introduce a white-box framework called DeepXplore that can perform limited testing. Secondly, the existing verification tools are limited by the complexity of the neural network and the non-linearity of the activation functions. The authors in [29] discuss the limitations of performing verification techniques on deep neural networks (DNNs). Third, these components learn from data and perform well only when the test observations resemble the training data. The authors in [26], [330] have shown how subtle changes (or adversaries) in the test image confuse the model and result in erroneous predictions. Recently, the DARPA Assured Autonomy project [331] has been focusing on designing tools and techniques for the safety assurance of autonomous systems.

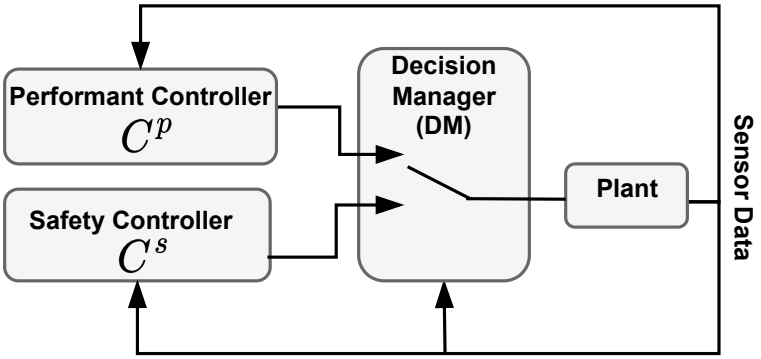



Figure 7.1: The simplex architecture combines an unverified performant controller ($C^p$) with a safe baseline controller ($C^s$) and a decision manager (DM) with a logic that arbitrates the system control among the controllers to maintain the system's safety objective.

**State-of-the-art and Limitations**: The problem of assuring the safety of CPSs has been tackled using controller-redundant architectures such as the simplex architecture [61] and controller sandboxing [98]. These architectures (see Fig. 7.1) improve the system's safety by augmenting a safety controller ($C^s$) and a decision manager (DM) to systems with high-performant and unverifiable controller ($C^p$). The DM holds a decision logic that arbitrates the control between the two controllers based on safety criteria. The primary advantage of these architectures is that they only require verifying the $C^s$ and the DM, thus alleviating the requirement of verifying the $C^p$ (e.g., LEC), which is sometimes difficult or impossible. Indeed, such techniques have been successfully used in unmanned aerial vehicles [100], remote-controlled cars [101], and industrial infrastructures [102].

However, the key challenges in using the simplex architectures for CPS applications are (1) **designing effective decision logic**: designing safe logic by always using the $C^s$ to make the decisions is trivial. However, that would make the system too defensive without utilizing the $C^p$. Therefore, the architecture should utilize the $C^p$ as much as possible to avoid conservatively using the $C^s$. That is, it is important to design decision logic that balances the safety and performance of the system. (2) **sudden transitions**: the instantaneous transitions (see Fig. 7.7) from the $C^p$ to the $C^s$ drastically degrades the system performance. For example, Bak, Stanley, *et al.* [332] have discussed the applicability of the simplex architecture in heart pacemakers. The authors discuss how sudden transitions between simplex controllers are dangerous in safety-critical applications. For a pacemaker case study, they illustrate how sudden jumps between two discrete heart rates (65 to 120 beats per minute) from two simplex controllers can make the patients dizzy and uncomfortable. Similarly, in automotive applications, the sudden changes in the control (e.g., speed) will be perceived as jerks (high rate of change of acceleration) and can be uncomfortable for passengers (e.g., Toyota sudden acceleration problem [333]).

Table 7.1: Notation Lookup Map

| Symbol | Description |
|---|---|
| $\theta_L$ | Steering PWM value of DeepNNCar using LEC controller |
| $\theta_C$ | Steering PWM value of DeepNNCar using OpenCV controller |
| $\theta_D$ | Steering PWM value using dynamic-weighted simplex strategy |
| $\theta_F$ | Steering PWM value using fixed-weighted simplex strategy |
| $W_L$ | Ensemble weight given to LEC controller |
| $W_C$ | Ensemble weight given to OpenCV controller |
| $W_{SET}$ | Ensemble weights $\{W_L, W_C\}$ computed by dynamic-weighted simplex strategy |
| $T_R$ | Inference pipeline time of DeepNNCar |
| $v_t$ | Current speed PWM value of DeepNNCar |

**Contributions:** To address these challenges we perform a weighted blending of the simplex control actions. We refer to this weighted extension of the conventional simplex architecture as the "weighted simplex strategy". Instead of selecting a single controller action, this strategy computes a weighted ensemble of all the controllers' actions. Such blending mechanisms have been shown to improve performance or accuracy in model ensembles [334] and model adaptive predictive control [335]. For the simplex architecture, we hypothesize that blending the controllers' actions could optimally balance the system's performance and soft constraint violations while avoiding abrupt transitions. Specifically, this strategy aims to optimize performance while reducing safety violations. So, it can only be used for systems that can tolerate soft constraint violations (i.e., constraints that can be violated but will incur a penalty). We summarize our contributions below.

- We discuss the design of a resource-constrained remote-controlled car called the DeepNNCar that will be used as a case study to test the proposed extensions to the conventional simplex architecture.
- We discuss the "weighted simplex strategy" and apply it to the DeepNNCar platform. We formulate the weight selection problem as an MDP and solve it using reinforcement learning. The RL-based logic is trained on both the system's safety and mission-critical objectives.
- We design a middleware framework to deploy the weighted simplex strategy on the DeepNNCar platform. The framework has a resource manager that monitors the computational resources of the system to mitigate any overload introduced by the complex computations of the proposed strategy.

**Outline**: The outline of this paper is as follows: in Section 7.3, we discuss the DeepNNCar platform and other background topics required to understand this work. Section 7.4 introduces the weighted simplex strategy and the setup to compute dynamic weights. In Section 7.5, we design a resource manager for managing the system's resource utilization. Section 7.6 describes the system integration. Section 7.7 evaluates the weighted simplex strategy and the resource manager for the DeepNNCar platform. In Section 7.8, we present related research and conclude the paper in Section 7.9. The notations used in the paper are described in Table 7.1.

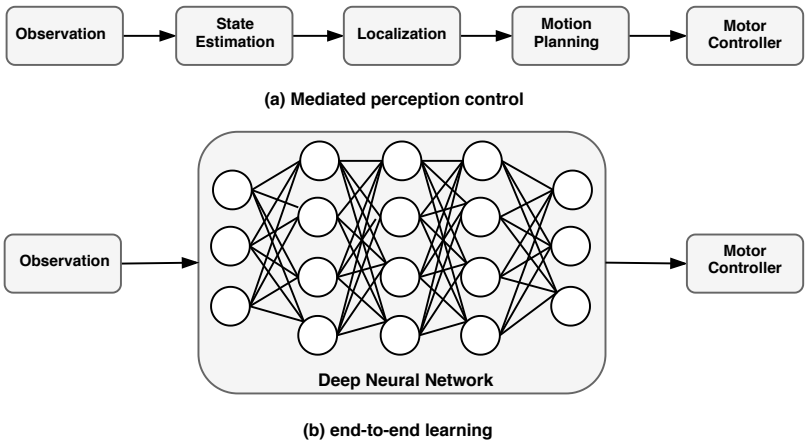



Figure 7.2: Perception-based control approaches in robotics [3].

## 7.3 Background

In this section, we discuss a few key concepts that are required to understand our methodology discussed in the later sections of this paper. Readers who are familiar can skip this section.

### 7.3.1 Control Approaches in Autonomous Systems

Perception-based control paradigms in autonomous systems can be classified into the mediated perception approach and end-to-end (e2e) learning approach, as shown in Fig. 7.2.

The mediated perception approach [336, 337] decomposes the problem into multiple sub-goals to form a processing pipeline for performing autonomous driving (see Fig. 7.2). Different sensing components (e.g., cameras, lidar, and radar) observe and learn the operating environment. The sensor readings are sent to the state estimation, localization, and motion planning components to interpret high-level information about the operating environment (e.g., lanes, objects, cars, and traffic lights) and learn a plan to navigate the environment. Finally, the motor control component uses the high-level information interpreted by the prior components to compute the low-level control actions for the system.

As discussed by Mantegazza, Dario, *et al.* [338], this approach has the following advantages: (1) transparent internal modes of operation for testing and debugging, (2) robust decision-making capabilities due to multiple dedicated algorithms, and (3) a high degree of freedom for the designer to select and fine-tune the internal stage algorithms. However, the multiple stages of operation and the need to create and maintain a large code base for simple tasks (e.g., indoor navigation) make it expensive for use in small-scale systems.

In contrast, the e2e learning [339] approach simplifies the control process by using a single supervised ML [340] component (e.g., DNNs) that directly computes the control actions from sensory data. Unlike

traditional software, where the execution logic is derived from analytical models, DNNs learn the underlying relationship between the sensory data and control actions from training data. This approach has recently gained considerable attention because of its conceptual simplicity and computational efficiency. It has been used in complex tasks such as obstacle avoidance [341] and autonomous driving [18, 4, 342].

However, some limitations of this approach are: (1) ML components are trained with data to learn the mapping between the sensor data and control actions. So, the quality and quantity of the training data highly influence how the approach works. For example, out-of-distribution (OOD) data samples have been shown to affect the performance of these components [92]. (2) training the ML component is challenging as several hyperparameters are to be tuned. Improper tuning of these parameters can lead to underfitting or overfitting problems.

### 7.3.2 Simplex Architecture

The simplex architecture is built on analytic redundancy, i.e., redundant components with different designs and implementations but similar interfaces [61]. Fig. 7.1 illustrates the structure of the simplex architecture, where the analytic redundancy is achieved using two controllers with the same interfaces but with different implementations and specifications. A performant controller ($C^p$) is combined with a reliable and safety controller ($C^s$) and a decision manager (DM). The decision logic of the DM is encoded to guarantee the system's safety, i.e., the logic will transfer the control from the $C^p$ to the $C^s$ when the $C^p$ decisions start moving the system toward unsafe states. The simplex combination of controller outputs can be represented as the weighted combination of the two controller outputs and can be written as in Eq. (7.1):

$$C_{SA} = W_1 \cdot C_P + W_2 \cdot C_S \tag{7.1}$$

Where $C_{SA}$ is the system's control output, $C_P$, $C_S$ represents the controller outputs of the $C^p$ and the $C^s$ respectively. $W_1$, $W_2$ represents the ensemble weights, and they sum up to 1. For the conventional simplex architecture, the weights are $W_1$=1 and $W_2$=0 in most scenarios as the $C^p$ is operating it. However, when the $C^p$ is on the verge of jeopardizing the system's safety, the control transfers to the $C^s$, making the weights $W_1$=0 and $W_2$=1. The blue line in the Fig. 7.7 shows the control transition between the two controllers of the simplex architecture. As seen, the control remains with the $C^p$ up to a certain point, and when the DM decides to switch, there is a sudden (or instantaneous) transition of the control to the $C^s$.

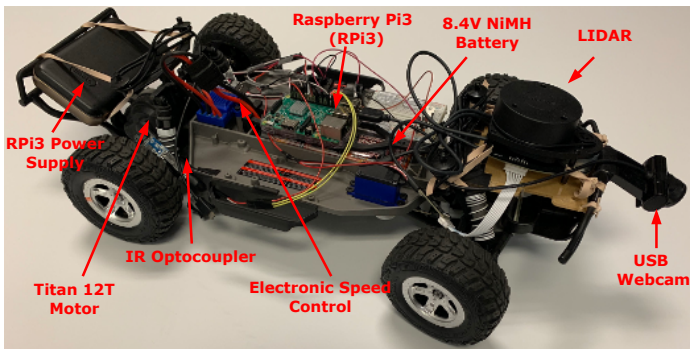



Figure 7.3: DeepNNCar is a resource-constrained remote-controlled car that is designed to perform end-to-end learning based autonomous driving. The hardware components and operating modes of the vehicle are discussed in Section 7.3.3.

### 7.3.3 DeepNNCar autonomous driving platform

To understand the methodology discussed in later sections, we first introduce the DeepNNCar[1] (in Fig. 7.3) platform and its operation. The vehicle is operated on indoor tracks shown in Fig. 7.4 with two objectives. The first objective of the vehicle is to operate within the tracks and reduce the out-of-track occurrences (minimize the soft constraint violations). The second objective is to operate at the highest possible speed on the different segments[2] of the track. For example, we have set a maximum attainable speed of 0.65 m/s for the straight segment and 0.25 m/s for the curved segments. Next, the platform is built using the chassis of the Traxxas slash 2WD 1/10 scale remote-controlled car. It has two onboard motors: (a) a servomotor for steering control and (b) a Titan 12T 550 motor for the motive force. These motors are powered using a single 8.4 V NiMH battery. A Raspberry Pi 3 (RPi3) is the onboard computational unit that performs all the required computations and interfaces with the sensors. RPi3 reserves two GPIO pins to generate pulse width modulation (PWM) signals to control the vehicle's motors. The duty cycle of a PWM signal is the proportion of time the signal remains in the high state (or logical 1) over the total time it takes to complete one cycle. The PWM signal has a duty cycle component and a frequency component.

For the DeepNNCar, the duty cycle is the percentage of a digital square pulse with a period of 10 milliseconds (frequency = 100 Hz). Varying the duty cycle percentage allows us to control the two onboard motors of the vehicle. For the servomotor, varying the duty cycle $\in$ [10%, 20%] results in a continuous steering angle $\in$ [-30°, 30°]. Similarly, for the titan motor, varying the duty cycle $\in$ [15.58%, 15.70%] results in a speed $\in$ [0, 1] m/s. Throughout this work, we use the notations ($v$) for speed and ($\theta$) for steering. The speed is referred to in terms of m/s and steering in degrees. However, to control the testbed, the steering and speed are varied as duty cycle values.

[1] Build instructions, source code, and videos of DeepNNCar can be found at https://github.com/scope-lab-vu/deep-nn-car
[2] We have classified different parts of the track into straight and curved segments.


Figure 7.4: We collected data from tracks 1 and 2 to train the LEC and perform exploration for the RL-based decision logic of the dynamic-weighted simplex strategy. Track 3 was used to test the performance and accuracy of the trained LEC and the dynamic-weighted simplex strategy. The tracks were built indoors in our laboratory using 10' x 12' blue tarps. The videos of DeepNNCar performing on these tracks can be found in https://github.com/scope-lab-vu/deep-nn-car

.

#### 7.3.3.1 Vehicle Setup

A USB webcam is attached to the RPi3 to capture images at 30 frames per second (FPS) with a resolution of 320 $\times$ 240 RGB pixels. A slot-type IR opto-coupler is attached to the chassis near the rear wheel that counts the wheel's revolutions. The speed of the vehicle is calculated based on the frequency of revolutions. The captured RGB images and the speed values are recorded on the computational unit and later utilized for training the vehicle's controllers.

Next, the vehicle (server) is used along with a desktop (client) to provide a server-client setup for data collection, monitoring, and runtime diagnostics. An Xbox controller is connected to the desktop via Bluetooth and communicates with the vehicle using TCP messages. The vehicle operates in three modes: (1) data collection mode to manually drive the vehicle to collect training data, (2) autonomous driving mode to autonomously drive the vehicle using a LEC or simplex strategies discussed in the later sections, and (3) live stream tracking mode to stream runtime images captured by the camera to the client for runtime tracking.

#### 7.3.3.2 Controllers

The vehicle is primarily driven by a LEC, and it also has a safety controller built using the traditional image processing based lane following algorithm. These controllers consume images from the forward-looking camera to compute the steering and speed control actions. A human supervisor sets the initial speed for the vehicle, which these controllers then vary during the vehicle's operation. These controllers are discussed in more detail in the rest of this section.

**LEC**: The vehicle performs autonomous driving using e2e learning. We use a modified version of NVIDIA's DAVE-II DNN that takes in the image and the current speed reading $v$ as inputs to predict the

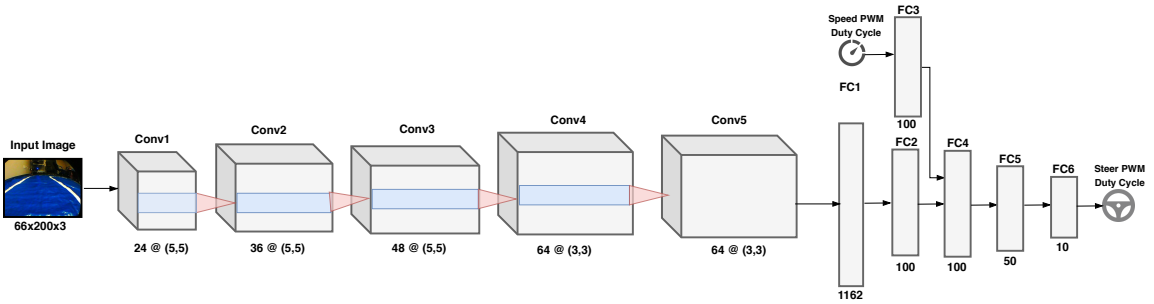



Figure 7.5: The modified DAVE-II model takes images and speed as the inputs to predict the speed and steering duty cycles. This modified model also takes in the speed as the inputs, which is not considered by the original NVIDIA DAVE-II model [4]. The model has five convolutional layers and six fully connected layers. *Conv* represents the convolutional layers, and the sizes of the filters are mentioned below. *FC* represents the fully connected layers, and the number of neurons is mentioned below.

steering control action ($\theta_L$). Our DNN model (in Fig. 7.5) has five convolutional layers and six fully connected layers. This is an extension to the original DAVE-II model [4] that consumed only images as input to predict the steering action. We modified the model architecture for two reasons. First, the steering and speed actions have some correlation that needs to be learned by the model. For example, a human driver would reduce the speed while turning the vehicle. Second, the quality of the captured image deteriorates with increased speed. Again, the model needs this additional information for accurately predicting the steering values.

**OpenCV controller**: The OpenCV controller is designed using classical image processing algorithms, and it performs two tasks: (1) detect and count the track lanes in images, and (2) associate discrete steering ($\theta_C$) values based on the lanes detected. The output of these tasks is the number and kind (left or right) of the detected lanes, the lane segments ($\hat{M}$), the discrete steering $\theta_C$ (in degrees) values, and a stop signal alarm when the vehicle moves out of the track. To perform these tasks, we apply a classical image-processing based lane detection (LD) algorithm, which performs the following steps:

- Gaussian blur and white masking: A 3x3 Gaussian kernel is convolved across the image to reduce noise. Next, all pixels except those within a specified range (e.g., [215, 255]) are masked, thus differentiating the track lanes from the foreground.
- Canny edge detection [343]: The algorithm first computes a gradient of pixel intensities. An upper and lower threshold of these gradients is defined at compile time. A comparison of the pixel gradients to these thresholds, in addition to hysteresis (suppress all weak and unconnected edges), can determine if a pixel is an edge or not. The edges reveal the boundary of the lanes.
- Region of interest (ROI) selection: The image is divided into two similar 30x66 regions of interest to capture the left and right lanes, respectively.
- Hough line transform [344]: A Hough line transform is applied to each ROI to detect the existence of a lane

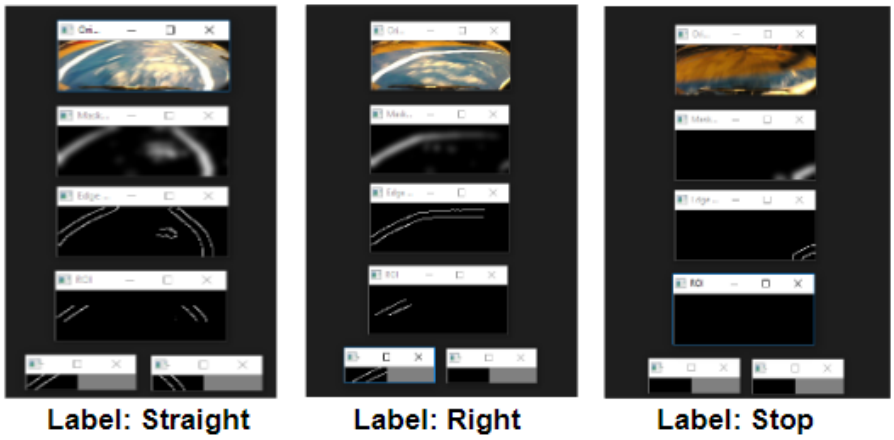



Figure 7.6: Image transformations as it passes through the different steps of the LD algorithm. Based on the lanes detected, the OpenCV controller assigns a track segment and then associates a discrete steering value. During the live stream mode, real-time images captured from the vehicle are relayed to the client, which runs the LD algorithm continuously to get this visualization.

based on the results of the Canny edge detection algorithm. Using this information, we determine a label for the track segment.

Fig. 7.6 shows the image transformation as it passes through the different steps of the LD algorithm. As seen, if two lanes are detected, the algorithm classifies the track segment as straight. If only the left lane is detected, the algorithm classifies the track segment as right. If only the right lane is detected, the algorithm classifies the track segment as left. Finally, once the segments are detected, the algorithm associates a discrete steering angle $\theta_C$, and its corresponding duty cycle (similar to [345]): if two lanes are detected, $\theta_C = 0°$ (corresponds to a duty cycle of 15%); if the right lane is detected, $\theta_C = -30°$ (corresponds to a duty cycle of 10%); and if the left lane is detected, $\theta_C = 30°$ (corresponds to a duty cycle of 20%).

## 7.4 Weighted Simplex Strategy

The conventional simplex architecture has two problems that we want to address in this work, and they are (1) non-availability of a reverse switch from the $C^s$ to $C^p$ controllers, which affects the system's performance, and (2) instantaneous switching between the controllers, which affects the system's stability. To address these problems, we combine the controller's outputs using weights instead of switching between the controllers. This means the weights of the controller's outputs in Eq. (7.1) are no longer restricted to taking only values of 0 or 1 but can take a value in the range of [0,1]. However, the combination of the ensemble weights must sum to 1. Such blending approaches have been popularly used in model ensembles [334] and model adaptive predictive control [335]. We call this approach the "weighted simplex strategy".

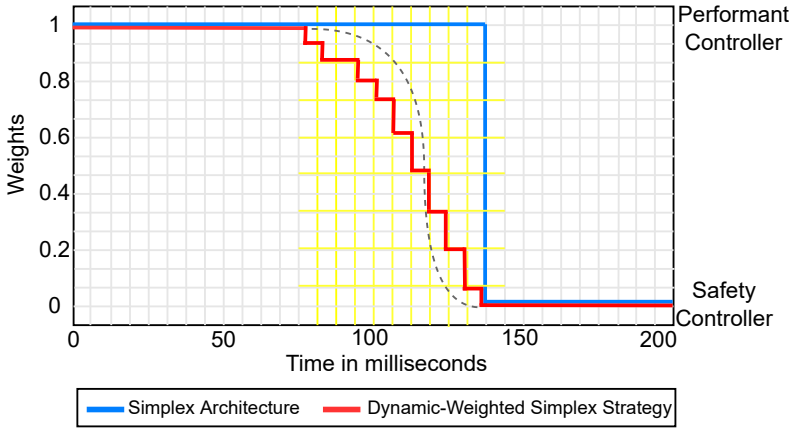



Figure 7.7: The weighted simplex strategy works like the conventional simplex architecture in high-performance and safe regions. However, the targeted spectrum of this strategy is the transition region represented by the yellow grid, where the $C^p$ actions start drifting the system towards the unsafe state. Here, instead of instantaneous transition, the dynamic weights of the weighted simplex strategy provide a mechanism for a smoother transition between the controller outputs. The discrete weight adjustment in the dynamic-weighted simplex strategy (red line) takes a staircase function approximating a smoother curve (dotted gray line).

As shown in Fig. 7.7, the weighted simplex strategy works like the conventional simplex architecture in high-performance and safe regions. The targeted spectrum of the proposed strategy is the arbitration region, which is represented by the yellow grid in Fig. 7.7. In this region, the conventional simplex architecture performs an instantaneous transition from the $C^p$ to the $C^s$. In contrast, the weighted simplex strategy's ensemble weights provide a smoother transition between the controllers. However, one challenge in using this strategy is the calculation of appropriate ensemble weights.

We present two approaches to select the ensemble weights: (a) **dynamic-weighted simplex strategy (DW)**, which dynamically computes the ensemble weights for blending the controllers. We model the sequential decision-making problem of selecting optimal weights as an MDP [346] and solve it using RL. (b) **fixed-weighted simplex strategy (FW)**, which uses fixed ensemble weights, manually tuned through experiments. We demonstrate these approaches for combining the steering control of the DeepNNCar platform. We are combining the steering actions because the LEC erroneously predicts the steering actions at high speeds around the curved segments of the track, resulting in increased out-of-track occurrences (see Fig. 7.10). The weighted simplex combination of the steering action is computed using Eq. (7.2).

$$\theta_D = W_L \cdot \theta_L + W_C \cdot \theta_C \tag{7.2}$$

Where $\theta_D$ represents the duty cycle corresponding to steering computed while using dynamic weights, $W_L$ is the ensemble weight of $C^p$, $W_C$ is the ensemble weight of $C^s$, $\theta_L$ is the steering computed by $C^p$, and $\theta_C$ is the

steering calculated by $C^s$. In the rest of this section, we discuss how these weights are computed using the two simplex approaches.

### 7.4.1 Dynamic-weighted simplex strategy

This section discusses the MDP formulation, and the Q-learning algorithm [347] set up to learn a policy that selects an optimal action (ensemble weights) for a given state of the MDP.

**Decision Epoch** We denote the decision epoch by $d$. The MDP we formulate is a discrete-time process, where an action must be chosen at each decision epoch. Note that the overall environment can evolve in-between decision epochs, (e.g., the track segments evolve in continuous time). However, as in prior work [348], the decision-maker can only intervene at discrete points in time. In principle, this discretization is not a challenge since we could always discretize time fine enough depending on the specific dynamics of the CPS at hand. In our use case, we make the decision once every inference cycle (100 milliseconds).

An MDP is a mathematical framework for modeling a sequential decision-making process in a stochastic environment. It is defined by a 4-tuple $\langle S, A, T, R \rangle$, where $S$ is the set of states, which capture information relevant for decision-making, $A$ is a set of actions that can be performed from a state, $T$ is the state-action transition model, and $R$ is a reward function that decides the incentive each action gets from a state.

**State**: We denote the finite set of states by $s$. The state includes the ensemble weights ($W_L$, $W_C$), the current speed of the vehicle ($v$), and the steering actions $\theta_L$ and $\theta_C$ predicted by the performant and safety controllers. The steering values are included as internal state information for locating the vehicle's position on the different segments of the track. The state $s_t$ of the vehicle at time t is: ($W_{L(t)}$, $W_{C(t)}$, $v_{(t)}$, $\theta_L$, $\theta_C$). The state transition happens when the weights $W_L$ changes by $\pm\ \delta W_L$ and $W_C$ changes by $\pm\ \delta W_C$, or the speed $v$ changes by $\pm\ \delta v$. The steering values are internal information and do not transition the state. These transitions are illustrated in Eq. (7.3).

$$\begin{bmatrix} W_{L(t+1)} \\ W_{C(t+1)} \\ v_{(t+1)} \end{bmatrix} = \begin{bmatrix} W_{L(t)} \\ W_{C(t)} \\ v_{(t)} \end{bmatrix} \pm \begin{bmatrix} \delta W_L \\ \delta W_C \\ \delta v \end{bmatrix} \tag{7.3}$$

Further, to limit the dimension of our state-space, we reduce the number of states in our MDP by discretizing the elements of the state. Each weight $W_L, W_C \in [0,1]$ is divided into 21 elements ($W_C$ = 1-$W_L$, as the two weights sum up to 1) in increments of 0.05. Also, our tracks were small, and we could not maintain the vehicle within the track at a speed $> 0.65$ m/s. Thus, we restricted the speed $\in$ the range of $[0, 0.65]$, the corresponding duty cycle in the range of $v \in [15.58\%, 15.62\%]$, and divided them into 41 elements in increments of 0.001.

| Action Space | $\uparrow \nu_t$ by 0.001 | $\downarrow \nu_t$ by 0.001 | NOP |
|---|---|---|---|
| $\uparrow W_L$ by 0.05 | (0.95,0.05,15.591, 16,15) | (0.95,0.05,15.589, 16,15) | (0.95,0.05,15.590, 16,15) |
| $\downarrow W_L$ by 0.05 | (0.85,0.15,15.591, 16,15) | (0.85,0.15,15.589, 16,15) | (0.85,0.15,15.590, 16,15) |
| NOP | (0.90,0.10,15.591, 16,15) | (0.90,0.10,15.589, 16,15) | (0.90,0.10,15.590, 16,15) |

Table 7.2: Action space for a given MDP state ($W_L = 0.90$, $W_C = 0.10$, $\nu_t = 15.590\%$, $\theta_L = 16\%$, $\theta_C$=15%). $W_C$ is computed as (1-$W_L$). Similar action combinations are generated for other states. NOP: means no operation. This is a sample action space when the car is in the straight segment of the track. So, the steering values remain almost the same in the next achievable states too.

**Action**: We denote the set of all actions by $A$. For each state $s \in S$, the vehicle performs an action $a \in A$, which results in a reward, $r : S \times A \rightarrow \mathbb{R}$, as the vehicle transitions from the current state to a new state $s \rightarrow s' \in S$. For example, the possible actions for the vehicle when starting from the state $s$ = ($W_L = 0.90$, $W_C = 0.10$, $\nu_t = 15.590\%$, $\theta_L = 16\%$, $\theta_C$=15%) is shown in Table 7.2. There are three possible actions ($\delta W_L$, $\delta W_C$ and $\delta \nu$) in each state. Thus, the current state can transition to one among 9 possible states as shown in the table.

**Transition Model**: We denote the state transition model by $T$. The state transition model $T(s'|s,a)$ describes the probability of traversing to the next state $s' \in \mathscr{S}$, given the current state $s \in \mathscr{S}$ and an action $a \in \mathscr{A}$. We refrain from discussing the mathematical model of the state transition probabilities, as our solution approach only needs a generative model of the environment (a simple model estimating the vehicle's position on the track) and not the explicit transition model itself.

**Reward**: We denote the reward function by $r(s,a)$. Since our goal is to reduce the vehicle's safety constraint violations while improving its performance, we incorporate them in designing the reward function. For our setup, we chose speed as the performance measure and the deviation of the vehicle from the center of the track ($\hat{t}$) to be the safety measure. The calculated reward is expressed in Eq. (7.4):

$$r(s_t, a_t) = \nu_t \cdot (1 - \hat{t}) \tag{7.4}$$

Where $\nu_t$ is the vehicle's current speed, and $\hat{t}$ is a scalar quantity calculated based on the deviation of the vehicle from the center of the track. The measure $\hat{t}$ is calculated based on the lane segment information given by the LD algorithm. If the algorithm detects both lanes in the captured image, we infer that the vehicle is at the center of the track, and we assign $\hat{t} = 0$. If the algorithm detects only one lane, we assume the vehicle has deviated from the center and assign $\hat{t} = 1/2$. Finally, if no lanes are seen, we assume the vehicle has moved out of the track, and we assign a hefty penalty $\hat{t} = 10$.

Our reward structure aims to provide a high positive reward for actions that will maintain the vehicle within the track boundaries. In contrast, a negative reward is awarded to actions that move the vehicle off the

track. Further, incorporating the speed information in calculating the reward enables the RL algorithm to learn the highest achievable speeds that the vehicle can achieve without exiting the track.

### 7.4.2 The learning algorithm

Typically, the reinforcement learning setup is composed of an agent (DeepNNCar vehicle) interacting with an environment (indoor tracks on which the car is intended to operate) [349]. The agent senses the state of the environment and selects an appropriate action to perform. The agent's action will cause a transition to a new state of the environment. The agent's physical dynamics govern this transition. The environment then calculates a reward for the action performed by the agent. A positive reward is provided for desirable actions (e.g., actions that keep the vehicle within the track), and a negative reward is provided for undesirable actions (e.g., actions that move the vehicle off the track). This interaction between the environment and the agent continues until the termination condition (set by the designer). The overall goal of the agent in this interaction is to learn a policy $\pi : s \rightarrow a$ that selects which action is optimal for a given state of the environment. For this work, the overall goal of the algorithm is to learn a policy that selects the optimal ensemble weights for a given state of the environment.

We use Q-learning [347], a model-free RL algorithm that learns the value of an action in each state of the MDP. The advantage of being model-free is that it does not require a model of the system's operating environment, which is often unavailable or difficult to model (e.g., open-world navigation) [350]. For each state-action pair $(s,a)$, the algorithm learns the "quality" $Q(s_t,a_t)$ (Q-function), which is a measurement of the quality of the immediate reward $r(s_t,a_t)$ from being in state $s_t$ and taking action $a_t$ discounted by the maximum expected future award in the new state denoted $Q'(s_{t+1},a_{t+1})$. The new Q-state, $Q'(s_{t+1},a_{t+1})$, is obtained by selecting an action which results in the maximum Q-value as shown in the Eq. (7.5) below.

$$Q'(s_{t+1},a_{t+1}) = \max_{a_k \in A} Q(s_{t+1},a_k) \tag{7.5}$$

The Q-state is updated using the Bellman equation, which takes the current state and action as inputs along with the parameters $\alpha \in [0,1]$ and $\gamma \in [0,1]$. $\alpha$ controls the learning rate of the algorithm, while $\gamma$ represents the discount factor that balances the importance of future benefits over immediate benefits (i.e., decreasing $\gamma$ increases priority on obtaining immediate rewards).

$$Q_{new}(s_t,a_t) = Q(s_t,a_t) + \alpha[r(s_t,a_t) + \gamma \cdot \max Q'(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \tag{7.6}$$

The new Q-states and the Q-value calculated from the Q-learning algorithm are stored in a Q-table, a look-up table that holds the state-action pairs $Q(s_t,a_t)$ and the associated reward $r(s_t,a_t)$. Finally, the policy likely

learned by the algorithm is to pick the action with the highest action value in each state.

Finally, the learning algorithm typically operates in two phases. In the exploration phase, a policy is implicitly learned from the Q-function, as $\pi(a_t|s_t) = \delta(a_t = argmaxQ(s_t, a_t))$, where the Q-function is stored in a Q-table as discussed above. It is important to note that the amount of exploration impacts the quality of the learned policy. Therefore, sufficiently exploring the state space is important. In the exploitation phase, the agent uses the learned policy to select an optimal action with the highest action value from the Q-table. In this work, we do not perform additional online exploration during the exploitation phase.

### 7.4.3 Fixed-weighted simplex strategy

The fixed-weighted simplex strategy uses static weights for the different segments of the tracks. The overall idea of this approach is to perform rule-based weights selection [351, 352] for different track segments (i.e., fix optimal weights for the straight and curved track segments). Rule-based decision-making has been applied to several applications of autonomous driving [353, 354]. Typically, the approach involves manually selecting the decision parameter and checking the performance of the system under test. Then, rules are formed around the value(s) of the decision parameters to control the system's performance.

We applied a similar approach to find the fixed weights around the different segments of the track. We manually selected a weight combination and then observed its effect on the vehicle's safety and performance objectives. However, it was difficult to define rules for our setup (it was difficult to accurately find the track segment in which the car operates). So, through extensive manual tuning across tracks 1 and 2 (see Fig. 7.4), we found the optimal weights to be $W_L$=0.8, $W_C$=0.2.

Further, to implement this strategy for the target platform, we use the concept of "Arguing Machines", introduced by Fridman *et al.* [355] for safe autonomous driving. The concept combined an autonomous system with two or more unsafe and unverified controllers (LECs) with a human supervisor as the decision logic to perform decision-making during uncertain situations. When the difference (or argument) between the control actions predicted by the two controllers exceeds a pre-defined threshold ($\tau_{SW}$), the system's control is transferred to the human supervisor to mediate the argument. We aim to integrate the arguing machine concept with the fixed-weighted decision logic. Suppose the computed steering difference between the $C^p$ and the $C^s$ is greater than $\tau_{SW}$. In that case, the selected fixed weights are used to compute the vehicle's steering control. However, if the difference is lower than $\tau_{SW}$, then the predicted steering of the $C^p$ is chosen to drive the vehicle. We also vary the vehicle's speed based on the arguments. If the difference among the controller predictions is greater than $\tau_{SW}$, then $v_t$ is decreased. However, if the difference among the controller predictions is lesser than $\tau_{SW}$, then $v_t$ is increased as it suggests consensus among the controllers' predictions.

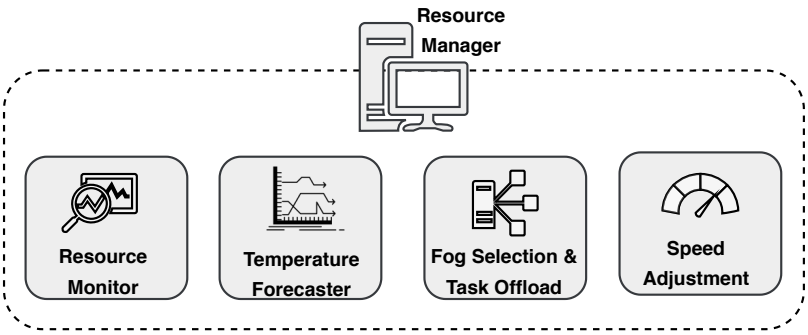



Figure 7.8: The resource manager is responsible for resource monitoring, temperature forecasting, fog selection and task offload, and speed adjustment.

## 7.5 Resource Management

The redundant controller operations of the weighted simplex strategy require a significant computational resource, which is not often available in small-scale autonomous vehicles such as those used in hospitals [356], warehouses [357], and laboratory research testbeds [81], [358]. For our target platform, the proposed strategy increases the power consumption, CPU utilization, and temperature of the RPi3 (onboard computing unit) beyond its configured soft limit (70°C). This varies the computing unit's clock speed and operating voltage [359], affecting the vehicle's performance. To address this problem, we offload some of the non-critical tasks of the vehicle to other nearby devices. In this work, we use the computation offloading approach to keep the development costs low and utilize the wireless communication capability that enables computation on other devices. For this, we use the onboard WiFi capability of RPi3 and set up a wireless communication with other edge devices [3] or fog devices [4] and offload some tasks.

For resource management, we designed a resource manager (RM) shown in Fig. 7.8, which performs: (1) continuous monitoring of resource state (temperature and CPU Utilization), (2) forecasting the temperature of the RPi3 based on current temperature and CPU utilization, (3) selection of an optimal fog device and offloading non-critical tasks to the selected device and (4) adjustment of the vehicle's speed to support the decisions of the simplex strategy.

### 7.5.1 Resource Monitoring and Forecasting

For resource-constrained computation platforms (e.g., RPi and NVIDIA Jetson), it is essential to continuously monitor system utilization, including CPU, memory, network, and disk. Variations in these utilization levels could result in degraded computational performance of the system. For our platform, we found the temperature and CPU utilization of RPi3 as the critical parameters that required monitoring because they affected

[3] Devices that have a similar computational capacity as the onboard RPi3.
[4] Devices that have higher computational capacity than the onboard RPi3.

the vehicle's performance. To monitor these parameters, we run the RM in the background continuously and measure the utilization once every 30 second using the psutil [360] library.

Further, forecasting the future system utilization based on the current workload is vital for resource-constrained computation platforms. When used with the resource monitor, the forecasting model can aid resource management by allowing time to enact mitigation strategies. For example, in our experiments, forecasting the RPi3 temperature based on the current workload helped proactively prepare for task offloading (see Fig. 7.15). For this, we designed and trained a neural network-based forecasting model. The model specifics are discussed in Section 7.7.

### 7.5.2 Fog Device Selection and Task Offloading

To offload the non-critical tasks, the RM contains an offloading routine that continuously performs a latency ping every 10 seconds to all the fog devices in the vicinity of the vehicle. We have a selection routine that runs for 30 seconds before selecting an optimal fog device. Within this 30 seconds, the routine pings the fog devices three times and computes an average ping time. The fog device with the lowest average latency is selected for task offloading.

Once the temperature forecasting model predicts the temperature to exceed 70°C and an optimal fog device has been selected, the RM prepares to offload the tasks. In our context, it is not feasible to transfer the entire task codebase to the fog device at runtime, so we assume these devices have a pre-installed copy of the codebase. During runtime offloading, the RM deactivates the code on the RPi3 and activates the same code on the fog device. The message transferring between the RPi3 and the fog device is managed using ZeroMQ (ZMQ) [361]. A similar task activation exchange is performed when the predicted temperature falls below 70°C. The forecasting model and the task offloading routine work synchronously at runtime to plan the offloading of the computations.

### 7.5.3 Vehicle Speed Adjustment

During the task offload process, the inference time $T_R$ (discussed in Section 7.6) increases because of the additional network overhead in the wireless communication channel. This change in the inference times affects the achievable speeds of the vehicle. To incorporate the impact of varying times on the vehicle's physical dynamics, the RM computes the tolerable maximum speed for a given inference time. The maximum speed $v_{MAX}$ is calculated using the safe distance ($d_S$), which is the closest distance to the track during turning when the vehicle can still safely act to avoid going off the track. The $d_S$ is a track-specific quantity which was found to be 0.09m for our tracks (see Fig. 7.4). Therefore, any decision before reaching this distance will give the

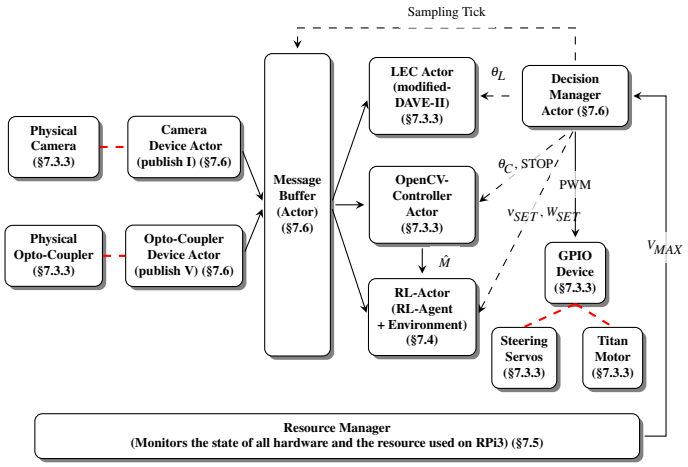



Figure 7.9: A block diagram of DeepNNCar along with actors. There are asynchronous interactions among various actors, so we use different messaging patterns. The request-reply communications are shown with dotted lines, the publish-subscribe communications are shown in solid lines, and the red dotted lines indicate the hardware connections. Also, refer to the listed section numbers for a detailed description of the components.

vehicle a good turning radius. However, any decision taken after this distance will leave the vehicle with insufficient space to turn, resulting in a safety violation. $v_{MAX}$ is computed as: $v_{MAX} = \frac{d_S}{T_R}$, where $T_R$ is the inference pipeline time. This speed is converted to the corresponding duty cycle value and applied to the RPi3 to control the vehicle's speed. During task offloading, the decision manager actor (DMA) must wait longer for a reply from the offloaded component due to the latency overhead, which increases the $T_R$.

## 7.6 System Integration

We use an actor-based design [362] to integrate the components discussed in the sections above. As discussed in [363], an actor is a self-constrained and restartable process with its execution thread and communicates synchronously or asynchronously with other actors. The actors of DeepNNCar and the data flow between them are shown in Fig. 7.9. Communication between actors is done with various ZMQ messaging patterns. The camera provides new images at 30 Hz, and the IR opto-coupler continuously measures the rotations per minute data to compute the vehicle's speed. Then, the camera device actor[5] and the opto-coupler device actor periodically publishes the images and current speed to all subscriber actors. The LEC actor, the OpenCV controller actor, and the RL actor are aperiodic consumers (see [68]). They do not consume sensor values until prompted by the DMA.

The interactions between the periodic publishers and aperiodic consumers are handled with the help of a message buffer actor (MBA), which has one buffer queue to store the published data (both images and current speed) along with a sequence label. The data in the MBA gets updated according to the sampling period of the

[5] A device actor converts hardware sensor information into topics that can be published and subscribed to (see [364]).

sensors. However, this data cannot be published until the MBA has received a sampling tick and a request from the DMA to publish the data of a certain label. Once the MBA gets this request, it publishes the image and current-speed messages to all subscribed actors. Using this data, the LEC actor predicts $\theta_L$, the OpenCV controller computes $\theta_C$, track position $\hat{M}$, and $STOP$ (a command issued if the vehicle goes out of the track), and the RL-Actor computes $W_L$, $W_C$ and $v_{SET}$.

**Decision Manager Actor (DMA)**: This is the key component in the architecture that controls and initiates the entire message exchange process in every inference cycle. The DMA issues requests for the sequence label and data $\theta_L$, $\theta_C$ from the controllers, and for $W_L$, $W_C$, and $v_{SET}$ from the RL-Actor. Once the controller and the RL-Actor have finished computation, they respond to the DMA with their label and values. The DMA then matches the labels and computes $\theta_D$ using Eq. (7.2) before feeding $\theta_D$ and $v_{SET}$ to the GPIO device actor, which controls the two motors.

The DMA starts a new cycle, which continues until it is terminated by the STOP signal from the OpenCV controller or manually by the user. The tasks performed between two sampling ticks of the DMA is one control cycle of the system, and the time taken to perform one control cycle is referred to as the inference pipeline time $T_R$. $T_R$ varies for every control cycle, but the average inference time for the dynamic-weighted simplex strategy is experimentally found to be 130 milliseconds (in Fig. 7.13).

## 7.7 Evaluation

### 7.7.1 Experimental setup

**Tracks**: We built three different indoor tracks shown in Fig. 7.4. These tracks were made in our laboratory using 10' x 12' tarps under controlled lighting conditions (higher lighting intensities create reflections on the tarp, causing the LD algorithm to fail). Each track had different geometric shapes and turns. All the experiments were performed using the tracks in the same wrinkled conditions. The LEC was trained on the images collected from tracks 1 and 2 and then tested on track 3.

**Baselines**: We evaluate the performance of the proposed dynamic-weighted simplex strategy (DW) against the following baseline controller configurations. (1) LEC, (2) OpenCV controller, and (3) fixed-weighted simplex strategy (FW).

#### 7.7.1.1 LEC and OpenCV controllers

**LEC**: We trained the LEC with 6000 labeled images collected from tracks 1 and 2 (in Fig. 7.4). We used the adam optimizer at a learning rate of 0.0001, with a decay factor of 0.1 every 50 epochs. We trained the estimator for 250 epochs and selected the model with the lowest validation error. We tested the trained model

| Track Segment | Precision (%) | Recall (%) |
|---|---|---|
| Straight | 97.73 | 87.78 |
| Curved | 90.78 | 93.05 |

Table 7.3: The precision and recall values to evaluate the performance of the LD algorithm in different segments of the track. We manually iterated through 3000 images and compared the actual lane segments to those predicted by the LD algorithm.

using a test set with 1000 images. We then used mean squares error (MSE) [365] as a metric to quantify the distance between the actual and predicted steering values. Once the MSE was within our acceptable threshold (0.1, as found through diverse experiments), we deployed the model on the vehicle for autonomous driving.

**OpenCV controller**: The LD algorithm was tested with a dataset of 3000 images and correctly labeled the track regions with an accuracy of 89.6%. Table 7.3 summarizes the accuracy of the LD algorithm in correctly classifying the lanes into straight or curved (left and right) segments. To generate this plot, we manually iterated through the 3000 images and noted the actual lane segment visualized by a human supervisor and the lane segment classified by the LD algorithm. Using this, we gathered the number of correct lane predictions and misclassifications to generate the data for the precision-recall graph. Using the precision and recall values, we also computed the F1 score for the straight and curved segments, which were 92.4% and 91.8%, respectively.

#### 7.7.1.2 Dynamic-weighted simplex configuration

**Parameters**: We explored the vehicle for 1000 steps across the track, which converts to 30 rounds across the tracks. These steps were sufficient because our state-space was small (approximately 900 states). However, for a large state space, we recommend longer exploration. During these runs, we set a learning rate $\alpha = 0.1$, and a discount factor $\gamma = 0.4$. We found these values to be the most feasible parameter for our experiments. Next, for all these exploration runs, we start with initial weights of $W_L$=0.5, and $W_C$=0.5, and we start from various parts of the tracks. Our idea with varying these parameters is to curate a robust Q-table with a larger percentage of the state-action pairs explored.

**Exploration and Exploitation**: Table 7.4 shows the ensemble weights learned by the algorithm through extensive exploration across the tracks. The column entries show the discretized steering duty cycle values the LEC could take ($\theta_L \in$ [10%,20%]). The row entries show the discretized steering duty cycle values the OpenCV controller could take ($\theta_C \in$ [10%,20%]). Next, we can observe that the RL algorithm learned different ensemble weights for different track segments. We observed that in the straight track segments, the weights quickly move to $W_L$=0.95, and $W_C$=0.05, and in curved track segments, the weights hover around

| | $\theta_L$= 10% (Left) | $\theta_L$= 11% | $\theta_L$= 12% | $\theta_L$= 13% | $\theta_L$= 14% | $\theta_L$= 15% (Straight) | $\theta_L$= 16% | $\theta_L$= 17% | $\theta_L$= 18% | $\theta_L$= 19% | $\theta_L$=20% (Right) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\theta_C$ = 20% (Right) | - | - | - | - | $W_L$=0.95 $W_C$=0.05 | $W_L$=0.95 $W_C$=0.05 | $W_L$=0.95 $W_C$=0.05 | $W_L$=0.85 $W_C$=0.15 | $W_L$=0.80 $W_C$=0.20 | $W_L$=0.80 $W_C$=0.20 | $W_L$=0.80 $W_C$=0.20 |
| $\theta_C$= 15% (Straight) | - | - | - | - | $W_L$=0.95 $W_C$=0.05 | $W_L$=0.95 $W_C$=0.05 | $W_L$=0.90 $W_C$=0.10 | - | - | - | - |
| $\theta_C$= 10% (Left) | - | - | - | $W_L$=0.80 $W_C$=0.20 | - | - | - | $W_L$=0.90 $W_C$=0.10 | - | - | - |

Table 7.4: The variations in the ensemble weights with the change in the steering PWM duty cycle of the LEC and OpenCV controller. The rows indicate the discretized steering PWM duty cycle of the OpenCV controller $\theta_C \in$ [10%,15%,30%]. The columns indicate the steering PWM duty cycle of the LEC controller $\theta_L \in$ [10%,20%]. These steering values are discretized in steps of 1%. The blocks with "-" indicate the unexplored region. Also, it is evident from the top right corner of the table that $W_C$ starts to increase as the vehicle turns in the right segment.

$W_L$=0.8, and $W_C$=0.2. We can also observe several steering angle combinations are unexplored (marked with a "-"). This is because the exploration tracks did not have left turns or extreme right turns.

The insufficient exploration was a problem when we deployed the learned Q-table to exploit a new track (not used during exploration) with a steep right turn. Though the algorithm always selected to turn right, its action (steering angle) was insufficient, resulting in an out-of-track occurrence of the vehicle. To overcome this problem, we started from the previously learned Q-table and re-explored track 3. This addressed our issues with the steep right turns on track 3.

| **Fog1 Latency (ms)** | **Fog2 Latency (ms)** | **Fog1 avg Latency (ms)** | **Fog2 avg Latency (ms)** | **Selected Fog Device** |
|---|---|---|---|---|
| 13.0<br>11.4<br>11.2 | 10.8<br>11.7<br>14.0 | 11.87 | 12.17 | Fog1 |
| 11.7<br>11.9<br>10.1 | 11.4<br>10.6<br>10.3 | 11.23 | 10.77 | Fog2 |

Table 7.5: Fog device selection. The selection is based on the average of three latency pings. The fog device with the lowest latency is selected for offloading. The experimental testbed for the fog device selection experiments had Fog1 device (laptop), Fog2 device (desktop), and each of these were connected to different WiFi networks.

#### 7.7.1.3 Resource Manager

The manager performs a device selection routine discussed in Section 7.5 to select the most reliable fog device that can be used for offloading the onboard tasks. To evaluate the selection routine, we identified two fog devices (a laptop with an Intel 4-core processor and a desktop with AMD Ryzen Threadripper 16-core processor) in our lab that can be used for task offloading. These devices used a common WiFi connection to communicate with the DeepNNCar. Table 7.5 shows the latency test results of the two fog devices. For this, the vehicle performed a ping to these devices once every 10 second. Then, the device with a low average latency

after three pings was selected to perform the offloading. We do not conduct online discovery of available devices in the current setup but assume that we have prior knowledge of the available fog devices.

Next, we designed a neural network-based forecasting model to proactively forecast the temperature of the onboard computing unit. The model has two layers with 20 and 40 neurons in the first and second layers, respectively. To collect data, we set up an experiment in which we continuously monitored temperature, CPU utilization, and offload status (indicated if the task was performed on the RPi3 or offloaded to a fog device). The dataset for training the forecasting model consisted of three hours of resource monitoring data. We trained the model for 200 epochs using adam optimizer and a learning rate of 0.0001. The model's performance and accuracy are discussed in Section 7.7 and illustrated in Fig. 7.14. We deployed the trained model on the vehicle at runtime to forecast the computing unit's temperature.

### 7.7.2 Results

We evaluated the baseline configurations with the DeepNNCar platform by running them for ten laps across the three tracks shown in Fig. 7.4.

#### 7.7.2.1 Comparing LEC and OpenCV controllers

We evaluated the performance of these controllers based on their highest achievable speeds and the number of out-of-track occurrences. During the testing regiment, the vehicle drives ten laps around the track for both the trained LEC and the OpenCV controller. The track was divided into straight and curved (left and right) segments, and performance results were compiled for each segment. A human supervisor manually noted the out-of-track occurrences. In the straight segment, the LEC performed well with very few (typically 2 or 3) out-of-track occurrences up to a speed of 0.55 m/s. The OpenCV controller typically had one or no out-of-track occurrences up to 0.35 m/s. Above 0.35 m/s, the OpenCV controller had higher chances of leading the vehicle out of track than the LEC.

The out-of-track occurrences of the different controllers in the curved segment of the track are shown in Fig. 7.10. The LEC has very few (typically three occurrences) out-of-track occurrences up to a speed of 0.45 m/s, and gradually as the speed increases, the out-of-track occurrences increase. In comparison, the OpenCV controller performed well up to a speed of 0.35 m/s with only one out-of-track occurrence. Again, above 0.35 m/s, the OpenCV controller started making higher wrong predictions leading the vehicle out of track. At 0.65 m/s, the OpenCV controller had 10 out-of-track occurrences compared to 8 occurrences of the LEC.

To summarize, these results show that one controller performs better than the other in specific tack parts. This implies that we can reduce the number of out-of-track occurrences by intelligently blending the two

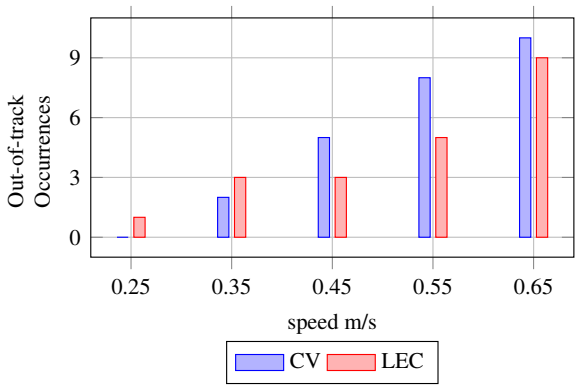



Figure 7.10: The out-of-track occurrences for different speeds in the curved segments of the track. From the figure, CV: OpenCV controller, LEC: modified Dave-II model. The horizontal axis shows the different speeds of the vehicle during the experiment. The data is collected by running DeepNNCar with each controller independently around the track for ten laps. A human supervisor manually noted down the out-of-track occurrences.

controller actions. In the next section, we evaluate the proposed weighted simplex strategy.

#### 7.7.2.2 Comparing Weighted Simplex Configurations

Fig. 7.11 shows the number of out-of-track occurrences performed by the different controllers at different speeds. We varied the speed between [0.25 - 0.65] m/s and ran all the controller configurations separately for ten laps around track 1. From Fig. 7.11, it is evident that, at low speeds ($<$ 0.35) m/s, both variants of the weighted simplex strategy have lower out-of-track occurrences compared to the independent controllers. At higher speeds (0.65 m/s), the dynamic-weighted simplex strategy outperforms all other controllers, reducing the number of out-of-track occurrences to three.

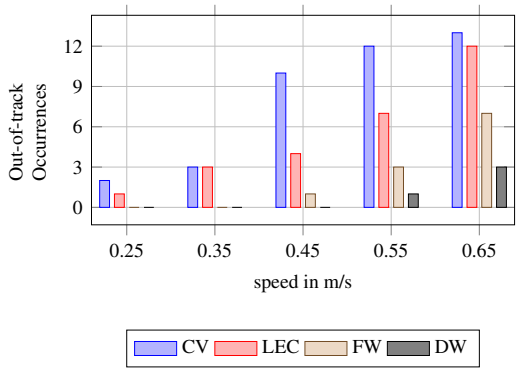



Figure 7.11: The out-of-track occurrences by different controllers for different speeds. From the figure, CV: OpenCV controller, LEC: modified Dave-II model, FW: fixed-weighted simplex strategy, and DW: dynamic-weighted simplex strategy.

Fig. 7.12 shows the maximum speed (represented as anomalies in the figure) and the average speed

performance of the different controllers. To collect data, we independently ran the vehicle with all the controller configurations for ten laps around track 1. The LEC and OpenCV controllers operate with the same average speeds of 0.29 - 0.39 m/s. This is because these controllers have a small number of out-of-track occurrences (see Fig. 7.11) in this speed range. However, the two weighted simplex strategies operate in a higher speed range because they perform significantly fewer out-of-track occurrences for speeds ($< 0.4$) m/s. Notably, the dynamic-weighted simplex strategy works faster with lower out-of-track occurrences than the other controllers.

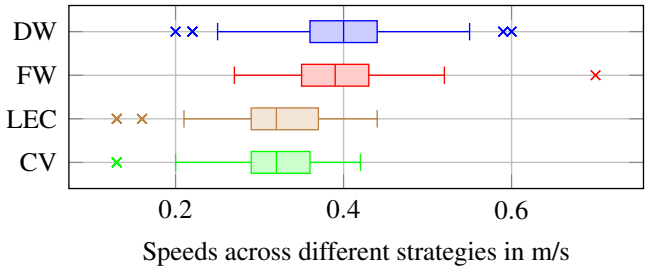



Figure 7.12: Speeds in meters per second of CV: OpenCV controller, LEC: modified Dave-II model, FW: fixed-weighted simplex strategy, and (d) DW: dynamic-weighted simplex strategy.

Fig. 7.13 illustrates the inference times of the different controllers. It is evident from the figure that the dual controller operation and the decision computation steps of the weighted simplex strategy increase the inference times of the vehicle. The dynamic-weighted simplex strategy has an average inference time of 130 milliseconds, and the fixed-weighted simplex strategy has an average of 120 milliseconds. In contrast, both independent controllers have shorter inference times of about 80 milliseconds.

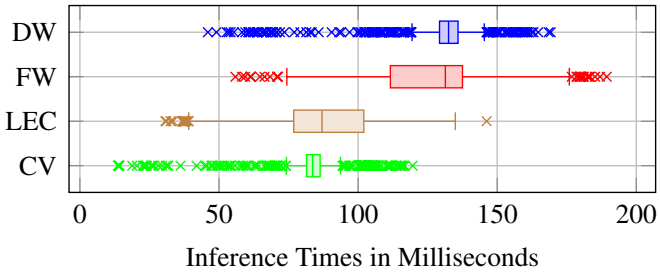



Figure 7.13: Inference times in milliseconds of CV: OpenCV controller, LEC: modified Dave-II model, FW: fixed-weighted simplex strategy, and (d) DW: dynamic-weighted simplex strategy. The variance in the inference times is because of the computational load and excessive temperature of the vehicle's onboard computing unit.

To illustrate how the ensemble weights are varied by our approach, we divided track 1 into three segments: Straight, Near Curved, and In Curved. Then as the vehicle ran using the two weighted simplex strategies, we

| Weighted Simplex Strategy | Straight Segment | Near Curved Segment | In Curved Segment |
|---|---|---|---|
| Dynamic Weights (WL, WC) | 0.95, 0.05 | 0.85, 0.15 | 0.80, 0.20 |
| Fixed Weights (WL, WC) | 0.80, 0.20 | 0.80, 0.20 | 0.80, 0.20 |

Table 7.6: Comparing the ensemble weights. For the dynamic-weighted simplex strategy the weights were dynamically updated by the Q-learning algorithm. For the fixed-weighted simplex strategy we have a fixed weight for all the track segments, these weights were manually tuned by a human supervisor.

recorded the ensemble weights. We later classified the weights into three segments. From Table 7.6 we see the weights used in the fixed-weighted strategy remain constant across the three-track segments. However, in the dynamic-weighted strategy, the weights change dynamically and are different for the three different track segments. The dynamic weights in each segment are not always the same as shown in Table 7.6, but they vary $\pm$ 0.05. To simplify the table, we have listed a single value that occurred most often.

To summarize, Fig. 7.11 shows the introduced dynamic-weighted simplex strategy can operate at higher speeds with lower out-of-track occurrences. However, this comes with a penalty of increased inference times (see Fig. 7.13). From these results, it can be inferred that dynamic blending of the ensemble weights helps reduce soft constraint violations while achieving higher performance.

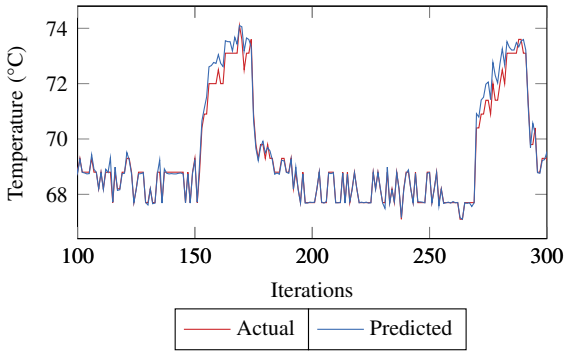



Figure 7.14: The result of the DNN temperature predictions vs. the actual temperature values. The DNN forecasts the temperature of the computing unit based on the current state (temperature, CPU utilization). We see the DNN predictions closely match the actual values. The graph shows a subset of iterations (total 10000 iterations).

#### 7.7.2.3 Evaluating the impact of task offloading

Fig. 7.14 shows the the actual and the forecasted temperatures of the computing unit. We can observe that the forecasted temperature closely matches the actual temperature of the computing unit. The forecasting model had a Mean Absolute Percentage Error of 0.16% across 1000 predictions.

To evaluate the performance of the task offloader, we continuously offloaded the task to one of the available fog devices (laptop or desktop). We then noted the temperature variations when the tasks stay onboard and when they get offloaded. The fluctuating blue lines in Fig. 7.15 show the temperature variations when the task was moved on and off the vehicle. The red lines indicate if the tasks were executed on or off the vehicle. As observed, the temperature drops below the threshold (70°C) when the tasks get offloaded.

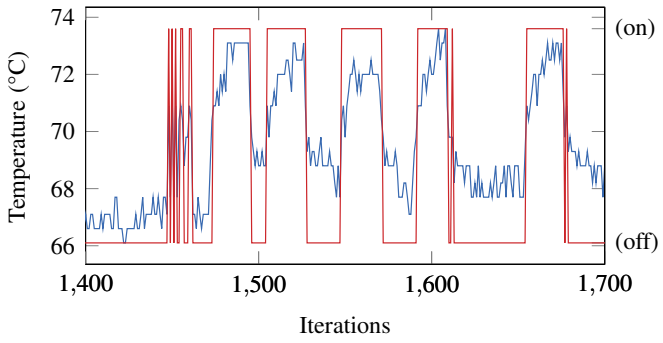



Figure 7.15: The effect of offloading the tasks in response to elevated temperature per iteration of the inference pipeline. The trigger to offload the task is 70°C. The blue line shows the temperature in Celsius. The red line shows when the tasks were offloaded to the fog (on=on fog, off=off fog). The graph shows a subset of iterations (total 10000 iterations).

Fig. 7.16 shows that the DeepNNCar can operate at higher speeds ($\sim$ 0.42 m/s) when all tasks are performed onboard. As tasks are offloaded to fog devices, the operating speed slightly decreases to 0.34 m/s. This is because of the continuous speed adjustment to compensate for the increased inference times (due to increased latency overhead).

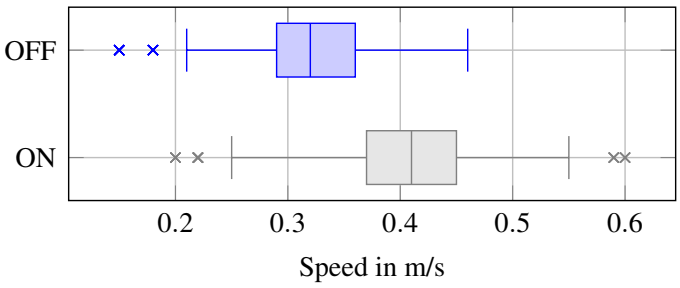



Figure 7.16: Speed readjustment during offload (m/s). ON: dynamic-weighted simplex strategy with all tasks executed onboard, and OFF: dynamic-weighted simplex strategy with RL task offloaded (Q-table was offloaded).

Fig. 7.15 shows task offloading performed to maintain the RPi3's temperature below the threshold (70°C). During offloading, the inference times increase because of the network overhead in sending the computations to the fog devices. The inference time comparison of the vehicle when tasks get offloaded vs. not offloaded is

shown in Fig. 7.17. The dynamic-weighted simplex strategy with all tasks performed onboard has a lower inference time ($\sim$ 130 milliseconds) compared to the inference time ($\sim$ 140 milliseconds) with RL tasks offloaded (Q-table was offloaded).

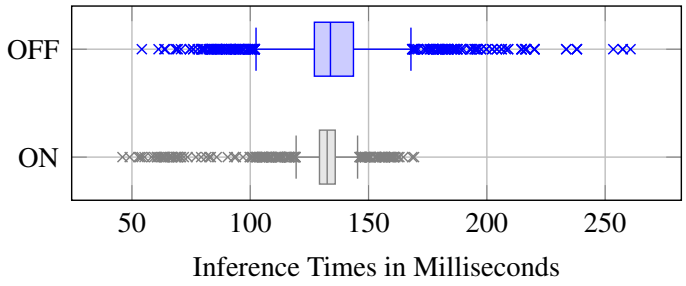



Figure 7.17: Inference times on task offload. ON: dynamic-weighted simplex strategy with all tasks executed onboard, and OFF: dynamic-weighted simplex strategy with RL task offloaded (Q-table was offloaded). The inference times increase with the increase in the RPi3's temperature.

The higher number of anomalies in this graph is because inference times vary based on the computational load and the onboard temperature of the RPi3. As the RPi3 overheats and gets overloaded, the chance of different inference times is higher. In addition, while offloading, the wireless exchange of messages is performed over Vanderbilt University WiFi. The network traffic variations in the WiFi add to the latency overhead.

To summarize, it is evident from Fig. 7.15 that the resource manager tries to keep a tight check on the RPi3 temperature by offloading specific non-critical tasks onto available fog devices. However, to balance the increase in inference times during offloading (see Fig. 7.17), the resource manager must penalize or adjust the vehicle's speed. The average speed during offload is 0.34 m/s, compared to 0.42 m/s without the offload. Since the primary concern of this paper is to reduce the soft constraint violations, the slight penalization in the performance (speed) is acceptable. For small-scale CPSs with limited computational capacity, we recommend using a resource manager and an optimal dynamic-weighted blending strategy, as discussed in this work.

## 7.8 Related Work

This section provides an overview of the existing literature on the dimensions of decision logic for simplex architecture, middleware framework for small-scale CPS platforms, and autonomous robot testbeds. A complete survey of papers in these areas is beyond the scope of this paper.

### 7.8.1 Decision Logic for Simplex Architectures

Linear matrix inequality (LMI) [63] and reachability analysis [98] have been the two popularly used verification-based decision logic. In LMI, the system's operational constraints and approximated linear dynamics are set as a convex optimization problem, which produces a positive-definite matrix. The matrix is constructed to represent an ellipsoid in the state space. The controller's switching depends on the primary controller's capability of maintaining the system within the ellipsoid. Reachability analysis involves computing a set of reachable states that the system can reach over a finite time horizon. The switching happens if the primary controller's actions lead the system towards an unsafe reachable state. Johnson, Taylor T *et al.* [110] present a real-time reachability algorithm that uses the offline LMI results with online reachability analysis. A zero-level set of barrier certificates [103] that separates the unsafe region from all the possible system trajectories starting from a set of initial conditions is presented as a decision logic for hybrid systems [366]. A zero-level set of barrier certificates [103] that separates the unsafe region from all the possible system trajectories starting from a set of initial conditions is used as the decision logic for two-hybrid system applications, including a water tank and a vehicle controller [366].

Instead of using a verification-based decision logic, Phan, Dung *et al.* [367] use a compositional proof technique called Assume-Guarantee (A-G) contracts for making the simplex decisions for a quick bot ground rover. The performant controller drives the system until this setup validates the agreement. The control is transferred to the baseline controller on the non-validity of the contract. Rule-based decision logic is designed for an unmanned aerial vehicle case study [100]. Here, the set of conditions under which the advanced controller has previously failed are listed, and this information is used for controller arbitration at runtime. All these approaches have only considered faults in the software components of CPSs. To assess both software and physical faults of CPSs, Wang *et al.* [368] propose an extended version of simplex called the $\mathscr{L}_1$-Simplex. To cover physical faults, an $\mathscr{L}_1$ adaptive controller is used as the advanced controller [369] because of its predictable system behavior in faulty situations. This controller is also used to compute the stability envelope used by the decision logic for controller selection.

Recently, simplex architecture has been integrated into the inference pipelines of autonomous CPS. Neural Simplex Architecture (NSA) [109] modifies the conventional simplex design in two ways. (1) an adaptation module is augmented to the decision manager for retraining the controller online with the previously performed unsafe action, and (2) a reverse switching logic is used to switch back from the baseline controller to the advanced controller, a feature not allowed in the conventional architecture. This approach is demonstrated for a system with an RL controller. Adaptive Simplex Architecture (ASA) [370] uses a LEC as the performant controller. It uses a simple acceptance test as the decision logic. If the system with the LEC component fails

the acceptance test, the safety controller controls the system.

### 7.8.2 Middleware Framework for Small Scale Robots

There are several middleware frameworks designed for managing the functionalities of small-scale robots. Seiger, Ronny, *et al.* [371] have developed a Robot Operating System (ROS) [326] based high-level programming framework to control the functionalities of a domestic robot. ROS commands are used through the framework to control the robot in manual, semi-autonomous, and fully-autonomous modes. The authors in [372] have introduced a middleware platform for distributed applications involving robots, sensors, and the cloud. The framework uses AIOLOS [373], a distributed software framework that allows the developer to design software on multiple components (sensors, robots) without having them develop the inter-component communication. The F1/10 platform [81] is a small-scale remote-controlled race car that uses a ROS-based communication framework to control the steer and speed control actions. In [374], the authors have discussed an architecture for intelligent manufacturing units on shop floors. The key focus of the middleware layer is to relay the data collected from the robotic sensors to the server for analysis. [375] discusses a multi-agent cloud-based framework for managing collaborative robots. The sensor data collected by a swarm of robots is sent to a cloud-based server, which computes the actions that need to be taken by the robots.

### 7.8.3 Autonomous Robot Testbeds

MIT's RaceCar [358] and University of Pennsylvania's F1/10 [81] are the two popular autonomous racing platforms built on Traxxas 1/10 scale remote-controlled car with an NVIDIA's Jetson TX1 on-board computational unit. These vehicles use cameras, inertial measurement units (IMUs), and expensive lidar ($1,775) systems for performing simultaneous localization and mapping. In contrast, our target platform (DeepNNCar) performs e2e learning using data from a limited array of sensors (camera, optocoupler, and lidar ($100)). DeepPicar [376] is another remote-controlled car platform that uses a smaller 1/24 scale chassis. Like DeepNNCar, this platform uses an RPi3 as the computational unit and performs e2e learning-based autonomous driving using NVIDIA's DAVE-II model. This platform is relatively inexpensive ($70) but has a considerably smaller chassis and uses discrete steering actuation, unlike DeepNNCar, which performs continuous steering. Donkey car [377] is an open-source autonomous car platform for small-scale remote-controlled cars. These cars are built on a 1/16 or 1/10 scale chassis and use RPi3 as the computational unit and a wide-angle camera as the primary sensor. This car performs discrete control actions compared to our platform.

## 7.9 Conclusions and Future Work

This chapter discusses the two key challenges of using the conventional simplex architecture for CPSs. They are: (1) designing an effective decision logic that is less conservative and (2) instantaneous transition between the simplex controllers that affect the system's stability. To address these challenges, we presented the dynamic-weighted simplex strategy that computes dynamic simplex weights according to the segments of the track. We also introduced a middleware framework that integrates the proposed dynamic-weighted strategy and provides a resource manager for monitoring the onboard computational unit. Finally, we introduced the DeepNNCar platform to deploy and test the proposed simplex strategy. We performed several experiments with the target platforms to evaluate the performance of the dynamic-weighted simplex strategy in reducing the soft constraint violations (out-of-track occurrences) and improving the performance (speed).

Our results show the dynamic-weighted simplex strategy works well at higher speeds ($\sim$ 0.4 m/s) with low out-of-track occurrences compared to the LEC and the OpenCV controller. While this is impressive, we identify two limitations of our approach: First, the approach is only suitable for systems that can tolerate soft constraint violations. However, safety-critical systems operating in factories, warehouses, and hospitals are often safety-critical, requiring hard safety assurance guarantees. Therefore, we cannot directly apply the proposed simplex strategy for such systems. Second, while the decision logic adaptively selects the ensemble weights according to the system's state, it is trained offline and does not include runtime information from the dynamic assurance components, which is essential for optimally balancing these objectives at runtime. Therefore, an open question that must be addressed is: *how can we design an adaptive mitigation strategy for safety-critical systems requiring absolute safety guarantees?*

# Chapter 8

# Mitigating the Risk: Dynamic Simplex Strategy

## 8.1 Overview

As discussed, the proposed dynamic assurance framework utilizes the simplex architecture to select a suitable action that mitigates the increased risk introduced by one or more hazards. The weighted simplex strategy (WSS) presented in the previous chapter (see Chapter 7) aimed at addressing two issues with the simplex decision logic, they are: (1) reduce conservatism by balancing the performance and safety objectives of the system. For this, WSS utilized a reinforcement learning (RL) logic with a two-objective reward function focused on both safety and performance and (2) avoiding instantaneous controller transition when the logic decides to switch. Rather than controller arbitration, WSS performs a weighted blending of the performant controller and the safety controller) to achieve a smoother control transition. While the strategy addressed these issues, the blending nature of the decision logic changes the safety guarantees provided by the simplex concept, thus making it unsuitable for safety-critical systems.

To address the same issues with the simplex decision logic for safety-critical systems, this chapter presents the "dynamic simplex strategy" (DSS). This strategy performs controller switching (as a traditional simplex strategy) instead of controller blending. It has a decision logic that is both online and allows for two-way switching while avoiding frequent back-and-forth arbitration. The problem is modeled as a sequential decision-making problem, and it is formulated as a Semi-Markov Decision Process. The forward switch from the performant controller to the safety controller is performed myopically from a safety perspective, and the reverse switch is performed by looking into the near future using the Monte Carlo tree search (MCTS) online heuristic. Compared to the previous RL-based logic (of the WSS approach), the online logic can incorporate the most updated information about the system's state and its operating environment. We demonstrate the performance of the proposed strategy using an autonomous vehicle example in an urban simulated environment.

The work comprising this chapter is complete at the time of this writing and is awaiting submission.

- **Shreyas Ramakrishna**, Baiting Luo, Christopher Kuhn, Ayan Mukhopadhyay, Gabor Karsai, and Abhishek Dubey. "Dynamic Simplex Strategy for Autonomous Cyber-Physical Systems" July 2022, Awaiting Submission.

## 8.2 Introduction

**Problem Domain**: Autonomous Cyber-Physical Systems (CPSs) are an important component of many applications in the fields of medicine, aviation, and the automotive industry. Such systems are often equipped with learning enabled components (LECs) that are often trained on data using machine learning (ML) methods like deep learning and reinforcement learning. A critical challenge for these systems is to make safe and efficient decisions under unanticipated system faults and dynamically changing operating conditions. While these components have illustrated autonomous CPSs have illustrated exceptional performance in a variety of challenging problems such as urban driving [4], recent studies show their predictions to be error-prone under three operating contexts: (1) when the testing condition is *out-of-distribution (OOD)* with the data used to train the LEC [91, 92] (see Chapter 5). (2) when the testing condition is in distribution with the training data, but it is severely *under-represented* (also called the class imbalance problem) in the dataset [378]. (3) few in-distribution conditions are *complex* to learn, and despite extensive training, the model may not learn correct predictions. These conditions pose a problem for the safety assurance of the CPS employing the LEC.

**State-of-the-art and Limitations**: One commonly used approach for providing (some) safety assurance to CPS is the use of *controller-redundant* architectures such as the simplex architecture [61] and controller sandboxing [98]. These architectures augment a safety controller and a decision logic to a CPS with a high-performing but unverifiable controller (also called the *performant* controller). The logic is usually trained with a safety-based utility function. Under unsafe operating conditions or system faults, the decision logic switches from the performant to the safety controller to maintain safety. Verification-based variants like linear matrix inequality [63], reachability analysis [98], and safety certificates [103] are also widely used as the decision logic. They are designed offline and deployed on the system for decision-making at runtime. Recently, adaptive approaches like behavior trees [379] and reinforcement learning [380, 381] are increasingly being used as the decision logic in these architectures. Indeed, such techniques have been successfully used in unmanned aerial vehicles [100], remote-controlled cars [101], and industrial infrastructures [102].

While these approaches have shown promising results, there are two major limitations. First, the decision logic is generally trained offline. While offline training provides the advantage of invoking the policy almost instantaneously at the time of making decisions, such policies can often become stale in non-stationary and dynamic conditions [104, 105, 106]. Second, the decision logic is usually designed to only perform a one-way switch, i.e., when the system under consideration detects an imminent threat to safety, the logic dictates a switch from the performant controller to the safety controller. Once the logic switches to the secondary controller, the control remains with it forever (barring some exceptions that perform the reverse switch based on system stability [107, 108, 109]). However, once the threat no longer exists, using the safety controller

could delay or ignore the system's mission-critical objectives [108].

Performing the reverse switch, i.e., transitioning back to the performant controller from the safety controller is highly non-trivial for several reasons. First, in real-world CPS, safety is paramount, and a myopic switch could be detrimental to the overall health of the system and related entities, including humans. As a result, it is imperative that a careful and non-myopic evaluation is done on the evolution of the system and possible exogenous factors before switching back. Second, these exogenous factors could evolve in a manner that is non-stationary and dynamic; such variation makes it necessary that the logic is equipped with an online algorithm that can incorporate the most updated information at hand. Online decision-making is shown to be beneficial in tasks such as autonomous driving [382, 104, 383], and real-time resource allocation [384, 106, 105]. However, online approaches are slow (compared to their offline counterparts), which inhibits their usage in practice. Finally, although the reverse switch is crucial for improving the system's performance, frequent back-and-forth switching among the controllers can be detrimental to stability and performance.

**Contributions**: In this paper, we present a principled hybrid approach to address the challenges of balancing safety and performance in CPS. Our approach aims at alleviating the safety problem caused when the performant controller (LEC) encounters a known operating condition that it has not learned well enough (complex or under-represented conditions in the training set). Our approach is based on a combination of data-driven planning under uncertainty and domain rules. Specifically, we present a dynamic simplex strategy (DSS) that is both online and allows for two-way switching while avoiding frequent back-and-forth arbitration. We formulate the decision-making problem as a Semi-Markov Decision Process. At a given state of the process, the decision for forward switching to the safety controller is performed by a selector based on the controllers' historical performances. The decision for reverse switching to the performant controller is performed using data-driven generative models to find a promising action by using Monte Carlo tree search, an online approximate search algorithm. The decision to switch is performed on the occurrence of a decision event, which is detected using runtime monitors that collectively indicate the system's risk of a consequence stemming from the system's faults and uncertainties in the operating environment. We evaluate the proposed approach in an autonomous vehicle (AV) study in a simulated urban environment and demonstrate that DSS leads to fewer infractions and higher performance than state-of-the-art alternatives.

**Outline**: The rest of this paper is organized as follows. In Section 8.3 we discuss the problem formulation. In Section 8.4, we present the proposed approach. In Section 8.5, we evaluate our approach using an AV example in the CARLA simulator [111]. In Section 8.6, we conclude this work. The source code of this work will be published alongside this paper under https://github.com/scope-lab-vu/AV-Adaptive-Mitigation.

## 8.3 Problem Formulation

### 8.3.1 Problem Setup

Consider an autonomous CPS, e.g., an autonomous vehicle, with both performance and safety objectives. A decision-maker, i.e., the system designer, must choose operational parameters (e.g., speed and throttle). Instead of directly optimizing the parameters, such systems are typically equipped with controller(s) that determine operational parameters. The decision-maker must therefore select a controller, which in turn, selects the parameters. Typically, a performant $C^p$ controller is used to ensure that the autonomous CPS focuses on its performance objectives (e.g., speed). Performant controllers are mostly designed using ML approaches and trained on data denoting operating conditions. Such data typically involves information from sensors like cameras, radar, and lidar to compute high-level trajectories or low-level control actions for the system. Formally, we denote a data point representing an operating condition as a *scene*, which are short-time trajectories of the system in the operating environment. For example, a scene could be a collection of scalar values denoting precipitation over a few seconds. Each scene is associated with a scene_label that includes different structural features $f^s$ (e.g., type of road, road curvature, and the presence of road signs) and temporal features $f^t$ (weather conditions and traffic density) characterizing the operating conditions. Typically, the performant controller is trained on sensor data from a large number of scenes.

In trying to achieve the system's performance objectives, the performant controller may neglect the safety objectives. As a result, the system is also equipped with several runtime monitors, a backup safety controller $C^s$, and a decision logic for safety assurance. These monitors raise alarms based on identifying different operational hazards (e.g., pedestrians in the system's travel path) and system hazards (e.g., sensor failure) that may increase the system's risk. When a hazard is detected, the decision logic switches the system's control from the performant controller to the safety controller. In this setting, our goal is to design an approach that balances the safety and performance objectives of the system.

### 8.3.2 Problem Formulation

We begin with an assumption on the problem structure and the available information for decision making. We assume we are given a route map of the system's travel (*srout*) broken up into a finite collection of scenes. Next, though the state space of the decision-making problem evolves in continuous time, for convenience we view it as a finite set of decision-making states that evolve in discrete times. To explain this, consider a system traveling on a given route. During its journey in the scene, the system changes the state of the world but presents no requirement for decision-making unless a *decision event* occurs. We have three events that

present scope for decision-making: (a) when the system enters a new scene, (b) when the weather changes, or (c) when a component of the system fails.

Next, a key component of our decision-making problem is that the system physically travels across scenes, which makes the temporal transitions between states non-memoryless. This further makes the underlying stochastic process governing the system's evolution to be semi-Markovian [106]. Therefore, we model the sequential decision-making problem of selecting an optimal controller as a Semi-Markov Decision Process (SMDP), which can be represented as a tuple $\langle S, A, T, R, \tau \rangle$, where $S$ is a finite state space, $A$ is a set of actions that can be performed from a state, $T$ is the state-action transition model, $R$ is a reward function that decides the instantaneous reward for taking an action in a state, and $\tau$ is the temporal distribution over state transitions.

**State** We denote the finite set of states by $\mathscr{S}$. We use $s_t \in \mathscr{S}$ to denote the state of the system at time $t$, which includes information about the scene, the controller driving the system, current component faults, and a counter to keep track of the number of controller switches so far. Formally, we represent the state $s_t$ by the tuple $(f_t^s, f_t^w, C_t^d, F_t, cnt_t)$, where $f_t^s$ and $f_t^w$ are the structural and temporal scene features, $C_t^d$ is the controller driving the system, $F_t = \{c1_t^f, \cdots, cn_t^f\}$ is the failure state of the $n$ components (e.g., sensor) on the system, and $cnt_t$ is a counter that keeps track of the number of switches that have been performed up to the time, all of them are observed at time $t$.

**Actions** We denote the set of all actions by $\mathscr{A}$. The action space of our SMDP is restricted to selecting one controller. In this paper, we restrict our attention to two controllers–a safety controller and a performant controller. In principle, our problem formulation (and the solution approach) can accommodate an arbitrary number of controllers as part of the action space. Our problem is thus simplified to decide whether to switch controllers, i.e., if the performant controller is operating at time $t$, the decision logic needs to decide if a switch to the safety controller is required at time $t$.

**Transitions**: The evolution of our system model is governed by several stochastic processes. First, as our system travels through several scenes, the travel is governed by a model of travel times. Second, the transitions of weather conditions are governed by a generative model, which provides a distribution to sample new weather conditions. Third, sensor failure may happen at any time and space governed by a certain distribution. We assume a sensor failure is a rare event following a non-memoryless distribution as an old sensor would be more likely to have a failure than a new sensor. Such an event is commonly approximated by a Weibull distribution with a shape parameter $\alpha$ and a scale parameter $\beta$. We refrain from defining the exact form of the temporal transitions and the state transition probabilities, as we capture the state transitions by a set of generative models. Finally, when a switch comes as the result of planning, the state variable that keeps track of the number of switches ($cnt_t$) is incremented by 1.

**Reward Function**: Reward in SMDP consists of a lump sum immediate reward and/or a continuous-time

reward as the system evolves [106]. Since we are interested in both performance and safety objectives, the reward for an action $a \in \mathscr{A}$ includes a performance score $perf$ and a safety score $safe$. We use the controller's average speed as the performance indicator and the controller's likelihood of collision as the safety indicator to model an immediate reward $R$. During forward switching, the instantaneous reward is calculated as the weighted sum of the two terms:

$$R = w_1 \cdot perf - w_2 \cdot safe \tag{8.1}$$

While these measures are sufficient for forward switching, we need an additional measure to prevent frequent back-and-forth switching among the controllers during reverse switching. Essentially, switching to the safety mode can be done myopically to avoid an imminent hazard. However, switching from the safety to the performant mode must be non-myopic. Therefore, we include a third term called cost of switching *cos* to track the number of previously performed switches leading up to the current state. We use the cost of switching counter *cnt* in computing the reward. We assign $cos = 0$, if the *cnt* variable is 0 or 1; otherwise $cos = cnt/ms$, in which *ms* is the maximum number of preferred switches. During reverse switching, the reward is calculated as the weighted sum of the three terms:

$$R = w_1 \cdot perf - w_2 \cdot safe - w_3 \cdot cos \tag{8.2}$$

## 8.4 Dynamic Simplex Strategy

We show an overview of our dynamic simplex strategy in Fig. 8.1. To perform the switching decision, the strategy has the following components. (1) the switcher we model as an SMDP decides "if" and "when" the switching must be performed. For deciding if a switch is required, at the occurrence of a *decision event* (refer to the problem formulation), the switcher queries one of the following decision-makers: (a) the selector for a forward switch in case the performant controller is driving the system, or (b) the planner for a reverse switch in case the safety controller is driving the system. In implementing these components, we assume the availability of a map that can provide the route the system will travel and its current location on the route. We also assume access to a lookup table (LUT) with historical performance (speed) and safety (collisions) information of the performant and the safety controllers.

Fig. 8.2 shows a transition diagram with the decision events occurring at different times (highlighted in red). On the occurrence of a decision event, the switcher works as follows. It first checks for the controller driving the system. If the performant controller is operating the system, the switcher activates the selector, which takes $t_s$ seconds to decide if the switch is required or not. For this, it searches a LUT containing the controller's historical performances and then computes the reward for each controller using Eq. (8.1). No switching is

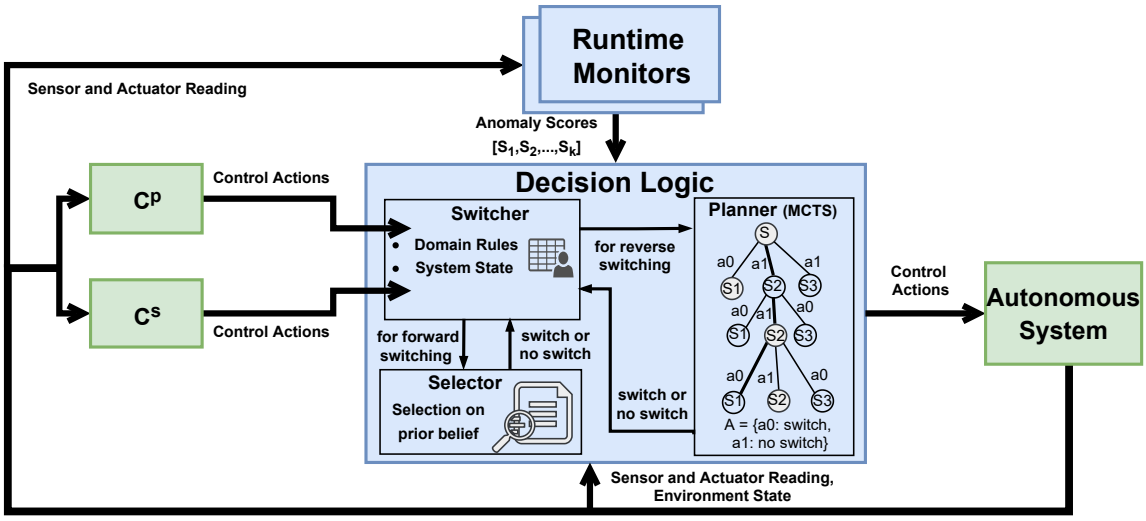



Figure 8.1: Overview of the proposed dynamic simplex strategy. The blocks in blue are designed through the solution approach. The green blocks represent generic controllers that can operate an autonomous CPS.

performed if the performant controller has a higher reward. Otherwise, a forward switch is performed. Next, if the safety controller is operating, the switcher activates the planner, which takes $t_p$ seconds to decide if a reverse switch is required. For this, it uses a generative model to simulate different weather conditions for the $n$ scenes in the future using MCTS. It performs a tree search that returns if a switch to the performant controller is worthy of being executed. If the decision maker's decision is to switch the controller, then a switching routine is activated, which will take $t_r$ seconds to perform the controller transition. However, if a new decision event occurs while either decision-maker is processing, the decision process will be terminated and initiated with the new state. We describe each decision-making component in greater detail in the rest of this section.

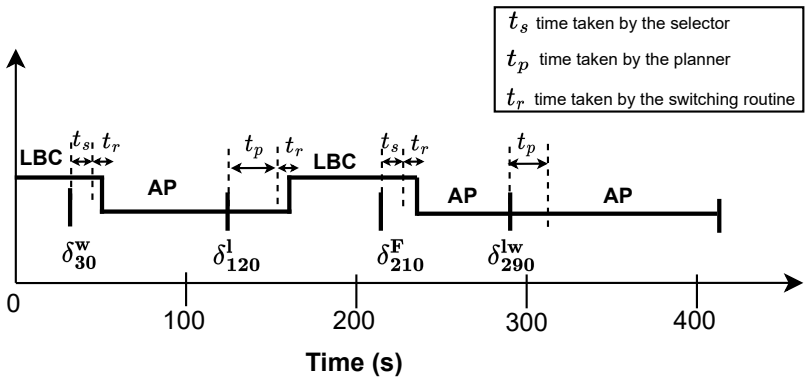



Figure 8.2: State transitions caused by different decision events at different times. Once an event occurs, the decision-maker takes a short time to decide whether a switching must be performed. In the case of a forward switch, the selector takes $t_s$ seconds to decide, and in the case of a reverse switch, the planner takes $t_p$ seconds to make the decision. Then, a switching routine will take $t_r$ seconds to make the controller transition.

**Algorithm 7** Controller Switching

**Parameter**: switching speed $\tau_s$
**Input**: current state $s_i$, current location $sloc$, current speed $v$
**Output**: $control\_action$

1: **if** $C^d == C^p$ **then**
2: $\quad a_i \longleftarrow Selector(s_i)$ ▷ run selector
3: $\quad control\_action \longleftarrow routine(\mathrm{v},\tau_s,a_i)$
4: **else if** $C^d == C^s$ **then**
5: $\quad a_i \longleftarrow Planner(s_i)$ ▷ run planner
6: $\quad control\_action \longleftarrow routine(\mathrm{v},\tau_s,a_i)$
7: **end if**
8: **return** $control\_action$

### 8.4.1 Switcher

As discussed, the switcher is activated when a *decision event* occurs. Depending on the kind of events affecting the system's safety, one or more monitors must be designed to detect these events at runtime. While we do not prescribe specific monitors, we leverage our previous work on runtime monitors [385, 119]; the implementation details are presented in the evaluation section.

Next, once these onboard monitors detect an event (change in weather, scenes, or component failures), the switcher selects an action as illustrated in Algorithm 7. If the performant controller is driving the system and the selector chooses to use the safety controller, the switcher will prepare to switch the system's control to the safety controller. However, if the safety controller is operating and the planner chooses to use the performant controller, the switcher will prepare to switch to the performant controller. If the decision-maker decides not to switch, the switcher will continue using the current controller.

A switching routine is activated if the decision-maker decides to switch the current driving controller. Instead of an instantaneous controller transition, the routine will consider the system's physical state and domain-specific rules to determine when and how the control transition must be performed. For this, it will first trigger a slow change in the control action (e.g., reduce speed). The change in the control actions is performed when: (a) the system's current speed ($v$) $<$ switching speed ($\tau_s$), and (b) the system is operating within the acceptable areas prescribed by our domain-specific rules. That is, we have a set of rules describing where the switching can be performed. For example, some acceptable areas for switching include the main road, overpass, and freeway. Rules for no switching include intersections, lane changes, or roundabouts. The routine will perform the controller transition when both these conditions are satisfied.

Also, to avoid the excessive burden of computation caused by running both controllers in parallel, the switcher warms up the controller selected by the decision-maker before bringing it online during this transition phase. Accordingly, the switcher routine will start operating the selected controller in a shadow mode by

sending it sensor data for predicting the control actions, which are discarded until the completion of the transition phase. After the transition phase is completed, the selected controller's predictions are used for operating the system.

### 8.4.2 Selector

When activated, the selector searches a LUT to select an optimal controller for the current state based on its historical performances. The table includes the scene_label of the given state $s$, the historical performance data of the controllers ($p(C^p|s)$ and $p(C^s|s)$) in the given state $s$. To get the performance scores, we run a search on the table using the k-nearest neighbors (K-NN) [386] algorithm. However, searching the entire table may get expensive if it contains many entries. To speed up the search, we first cluster the entries into distinct groups based on the structural scene labels $f^s$ of the system's travel route $srout$. Then, we select one cluster based on the system's current location $sloc$ (assumed to be available from a GPS sensor).

The search within the cluster is shown in Algorithm 8, and it works as follows: We separate the cluster entries further into two groups. The first group has entries where a sensor failure occurred, and the second group has entries where no failures occurred. Then, we apply the K-NN algorithm to one of these groups based on the occurrence of a failure or not (information available from the onboard monitors). The goal of the search is to find the $k$ nearest entries in the cluster that matches the weather conditions of the current scene_label. The controller's performance is averaged across the selected neighbors. These scores are used to compute the reward (see Eq. (8.1)), which is used to decide if a forward switch is required or not.

### 8.4.3 Planner

When activated, the planner takes the current state as the input and returns if a reverse switch is required by looking into the near future. For this, it simulates the possible future operating conditions and system component failures using the MCTS online algorithm. Since the operating conditions and the component failures vary dynamically, using an online algorithm is a natural choice.

MCTS is a heuristic search algorithm for finding optimal decisions in a decision space by building a search tree [387]. The tree nodes represent the states, and the edges represent the actions that mark the state transitions. The search begins with a root node, and a child node is recursively selected until a leaf node is reached. Unless this leaf node denotes the end of the planning horizon, an action is taken in this state node, and the tree is expanded by adding corresponding child nodes. To estimate the value of an action from a node, the algorithm simulates a "rollout" from the child node to the end of the planning horizon with a computationally cheap default policy. Using the algorithm requires three components: (1) a generative model to simulate the

**Algorithm 8** Controller Selection

**Parameter**: neighborhood size $k$
**Input**: LUT data $X_{input}$, travel route $srout$, current state $s_i$, current location $sloc$
**Output**: $p(C^p|s_i)$, $p(C^s|s_i)$

$X_{sub} \longleftarrow$ generate_clusters($X_{input}$,$r$) ▷ generate data clusters
$X_{cluster} \longleftarrow X_{sub}[sloc]$ ▷ match the cluster
**for** $x$ in $X_{cluster}$ **do** ▷ search within cluster entries
  **if** $c_i^f$ in $x$ **then**
    $X_{sel} \longleftarrow x$
  **end if**
**end for**
**if** $s_i[f^t]$ in $X_{sel}$ **then**
  **return** $p(C^p|s_i)$, $p(C^s|s_i)$
**else**
  Neighbors ($N$) $\longleftarrow$ KNN($s_i[f^t]$,$X_{sel}$,$k$)
  T1 $\longleftarrow$ [], T2 $\longleftarrow$ []
  **for** $n \in N$ **do**
    $p(C^p)$,$p(C^s) \longleftarrow$ get_score($X_{sel}[n]$) ▷ get beliefs
    T1.append($p(C^p)$, T2.append($p(C^s)$
  **end for**
  $p(C^p|s_i) \longleftarrow sum(T1)/len(T1)$ ▷ average belief
  $p(C^s|s_i) \longleftarrow sum(T2)/len(T2)$
**end if**
**return** $p(C^p|s_i)$, $p(C^s|s_i)$

future states, (2) a tree policy to navigate the tree search, and (3) a default policy to estimate the value of an action. We describe these components below.

**Generative Models**: As discussed, we use a set of generative models to perform the state transition. The weather model provides a method for sampling new weather conditions as the tree is built. The sensor failure model provides a method for sampling different sensor failures. We learn the weather model based on the historical data gathered from many previous simulations. We primarily learn the rate at which the different weather parameters vary. Similarly, we use the sensor failure rate and severity to build the sensor failure model. For example, a digital camera's failure rate is roughly 0.1% over 2 years of operation. While some of these failures (e.g., bright images) get rectified over time, other failures (e.g., broken lens) persist until replacement.

**Tree Policy**: We employ the tree policy for searching the tree and deciding which nodes must be visited. The action selection in the policy is driven by the standard Upper Confidence bound for Trees (UCT) algorithm [388], which defines the score of a node as

$$UCB(s,a) = Q(s,a) + c_{uct}\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{8.3}$$

Where, $Q(s,a)$ is an estimated value for the the state-action pair, $N(s,a)$ denotes how often the action $a$ has been taken in the state $s$, $N(s)$ is the number of times $s$ has been visited, and $c_{uct}$ is used to tune the trade-off

**Algorithm 9** Monte Carlo Tree Search (MCTS)

**Parameter**: Number of tree simulations $M_n$, tree search depth $M_d$, parameter controlling tree exploration $c_{uct}$
**Input**: Current state $s_i$, Generative models $G$
**Output**: Policy $\pi$

**function** MCTS($s_i$)
    initialize $s_i$ as the root node
    **for** $m = 1, \cdots, M_n$ **do**
        Tree_Search($s_i$)
    **end for**
    $\pi(a|s_i) \longleftarrow \frac{N(s_i,a)}{N(s_i)}$ ▷ switching policy
    **return** $\pi$
**end function**

**function** TREE_SEARCH($s$) ▷ recursive tree search
    **if** $s$ is terminal **then**
        $r \longleftarrow R(s,a)$
        **return** $r$
    **else if** $s$ not visited **then**
        $r \longleftarrow rollout(s)$ ▷ rollout
        **return** $r$
    **end if**
    $a \longleftarrow \text{argmax}_{a \in A} UCB(s,a,c_{uct})$
    $s^* \sim G(s,a)$ ▷ simulate new state
    $q \longleftarrow R(s,a) + \gamma Tree\_Search(s^*)$ ▷ perform tree search
    $N(s,a) \longleftarrow N(s,a) + 1$
    $Q(s,a) \longleftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
    $N(s) \longleftarrow N(s) + 1$
    **return** $q$
**end function**

between exploration and exploitation. A higher value for $c_{uct}$ puts less emphasis on the current value estimation $Q(s,a)$ and forces the tree towards actions taken less frequently. The first term $Q(s,a)$ is the exploration term that biases the search toward the nodes which appear promising. The second term is the exploitation term which encourages the attempts at the under-represented action for the current state.

**Default Policy**: We employ a default policy to estimate the value of action during rollout. Our default policy is random. We randomly select an action that should be taken when the safety controller is operating the system. If the chosen action is "to switch" to the performant controller, we will sample the future state only from a set of states with the driving controller as the performant controller. The idea behind biasing the sampling process is that we want to see how the expected rewards will be if the system continues using the performant controller in the future.

**Tree Search**: The tree search algorithm is shown in Algorithm 9. Given that we are simulating the future operating conditions, running the tree search once does not adequately represent the future conditions. To handle this, we run the tree search for multiple iterations, which gives us more comprehensive coverage of

the possible conditions. The search begins each iteration by initializing the current state $s$ as the tree's root node. If the current state is the tree's terminal state (leaf node), then the reward for this state is calculated, and the search for this iteration is terminated. We use the K-NN algorithm, discussed in Algorithm 8 to get the performance and safety scores and combine it with the switch counts to compute the reward using Eq. (8.2).

If the current state was not visited during the earlier search, a rollout is performed from the current state node by randomly selecting actions until a leaf node is reached. The simulation outcome is then propagated back to the root node while averaging the rewards (see Eq. (8.2)). However, if the state was visited earlier, an action that maximizes the Upper Confidence bound is selected. Then, based on this action, the generative models simulate a new state, which then becomes the root node of a new tree search. Finally, a backward pass is performed to update the Q-value and the visit count of all the previous nodes visited during the iteration.

## 8.5 Evaluation

We evaluate the proposed strategy on an AV example in CARLA simulation [111]. We created 10 closed-loop tracks in towns 3 and 5 of the simulator. Ten waypoints define the geometry of the tracks. We divide each track into five scenes containing two waypoints each. In total, we have 50 scenes from these tracks. We assigned each scene with semantic labels that include the road types, road curvature, presence of traffic signs, and traffic density. We also changed the weather conditions by varying the cloudiness, precipitation, and precipitation deposit parameters in the range [0,100] and the time of day parameter in the range [0,90]. We leverage the ANTI-CARLA framework presented in Chapter 9 to automatically generate the required data.

### 8.5.1 Setup

**AV setup**: The system block diagram of the AV is shown in Fig. 8.3. It is primarily driven by a high-performant ML controller called Learning By Cheating (LBC) [5], which uses a DNN for navigation. The controller uses six sensors, including three forward-looking cameras, an inertial measurement unit (IMU), a global positioning system (GPS), and a speedometer. The controller uses these sensors to compute the control actions as follows. First, it uses a navigation planner that takes the waypoint information from the simulator and divides it into smaller position targets. Next, it uses the GPS and IMU sensors to get the vehicle's current position. It feeds this along with the position targets into a velocity planner that computes the desired speed. The desired speed and camera images are fed into a DNN, which predicts the trajectory angle and the target speed. These predictions and the current speed are sent to PID controllers to compute the throttle, brake, and steer control signals.

Next, we also have an autopilot controller (AP), which is slow but performs very few collisions. The

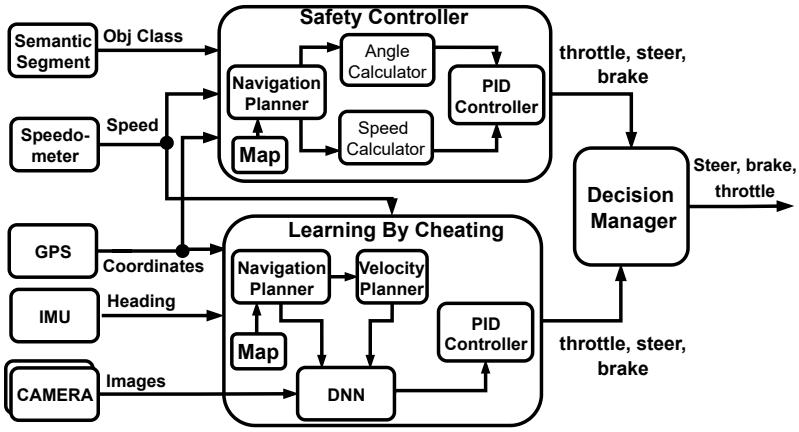



Figure 8.3: AV system model designed for the CARLA autonomous challenge setup. The AV is primarily driven by the LBC controller [5]. We have augmented a safety controller and a decision logic.

controller uses the GPS, the speedometer, and the semantic segmentation camera sensor to compute the control actions as follows. First, it uses the navigation planner to get the waypoints from the simulator and divides them into smaller position targets using the priority information (e.g., position of traffic signs) from the simulator. The position targets, current position, and speed (from GPS and IMU sensors) are sent to angle and speed calculator functions to calculate the angle of trajectory and the desired speed. These values are sent to different PID controllers to compute the throttle, brake, and steer control signals. Finally, the control actions from the two controllers are forwarded to a decision manager with the logic discussed in Section 8.4.

**Data collection and pre-processing**: To train the runtime monitors and curate the controller LUT, we ran 12500 simulations using each controller in the 50 regions across the CARLA towns, which amounts to 94 hours of driving. We randomly changed the time of day and weather parameters of cloudiness, precipitation, and precipitation deposits in each simulation run. In addition, we also simulated different camera failures of image blur and camera occlusion in some of these runs. In each simulation, we store the weather parameters, the speed, and infractions (collisions, route incompletion, and off-road driving) performed by each controller. Finally, we binned the weather parameters into 4 groups of low, medium, high, and very high. For example, we bin the precipitation levels into low rain, medium rain, high rain, and very high rain. Though we cannot cover all the possible weather conditions, we ensure each bin has sufficient samples.

**Baselines**: We evaluate the performance of DSS against the following baseline controller configurations: (1) LBC controller, (2) AP controller, and (3) traditional simplex architecture with an offline decision logic for forward switching ($SA$). The forward switching is performed on the controller's historical performances and (4) traditional simplex architecture with reverse switching ($SA^R$). Both forward and reverse switching is performed on the controller's historical performances.

### 8.5.2 Decision Logic and Runtime Monitors

The events triggering the decision-makers are: (a) change in the scenes, (b) change in weather, (c) high collision likelihood, and (d) a camera failure. To detect these events, we design and train the following monitors[1] using the 12500 simulations.

**Collision Likelihood Estimator**: To predict the collision likelihood, we use the concept of introspection [385]. The idea of introspection is to record the collision information and then learn patterns from the recorded observations to predict such collisions in the future. We trained the estimator with state vector sequences of 3 seconds sampled from 10 seconds of state data (velocity, acceleration, and steering angle) that led to a collision. We labeled each state sequence leading to a collision as *collision* and each sequence from the successful driving as *success*. The LBC controller had 250 collisions among the 12500 simulations. We used 90% of the recorded data for training and the rest for validation of the model. We used the adam optimizer at a learning rate of 0.01, with a decay factor of 0.1 every 20 epochs. We trained the estimator for 100 epochs and selected the model with the lowest validation error. In a receiver operating characteristic analysis on the validation set, the model achieved an area under the curve of 0.89, in line with results for real-life driving data [385]. The softmax score associated with the class *collision* is used as a collision likelihood.

**Novelty detector**: Novel scenarios are inherently less likely to be handled correctly by the performant controller. We use a semantic segmentation network to generate segmentation images from RGB camera images and use the generated images to detect semantic novelties [389]. For novel scenes, the segmentation network will generate spurious or uncertain predictions. We use the average softmax score of the predicted semantic image as the novelty score. To implement the detector, we train the DeepLabV3+ architecture [390] with a large-scale set of videos with pixel-wise semantic annotations recorded in CARLA. Our model achieved an average precision of 84.6% on the validation set. Fig. 8.4 illustrates the detector's performance on a regular image (image belonging to the training set), a noisy image, and an image with novel objects (helicopters). For both the noisy set and the set with semantic novelties, the average softmax score is significantly low at 0.8 compared to 0.95 for regular images.

**Camera fault detectors**: As occlusion and blur images are two common camera-related faults [391], we design detectors to detect their occurrence (modeled by a Weibull distribution). The blur detector uses the variance of the Laplacian operator across an image [324] to detect if the image is blurred. The operator highlights the image pixels with large changes in intensity. A variance in the operator's score below the threshold of $\tau_b = 50$ indicates the image is blurred. The detector had an F1 score of 97% on the validation images. Next, we also design an occlusion detector to detect continuous black image pixels blobs. We mask the

[1]The collision likelihood estimator and the novelty detector was entirely done by Christopher Kuhn.

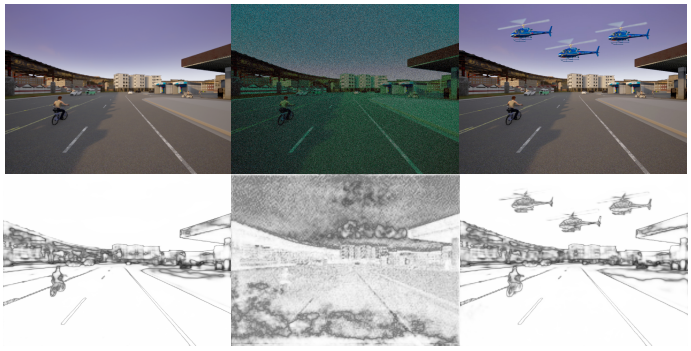


Figure 8.4: The top row shows the RGB camera images, and the bottom row shows the corresponding semantic segmentation images.

image to find connected black pixels in the image and then color these pixels into white. Then, we calculate the percentage of white pixels in the image. If the white pixels are larger than $\tau_o = 10\%$, the image is considered occluded; otherwise, it is not. The detector had an F1 score of 98% on the validation images. We empirically selected the thresholds based on the amount of blur and occlusion levels the LBC controller can tolerate.

**Decision logic parameters**: We report the experimental hyperparameters. These values were empirically tuned using data from 50 simulation runs. First, the decision epoch for the logic was set as 1 second. We ran the logic once every 20 inference cycle, given the simulations were set up to run at 20 frames per second. Second, for the switcher, we set the switching speed $\tau_s$ as 2.5 m/s. Third, for the K-NN algorithm used in the selector and the planner, the number of neighbors $k$ was varied between $\{3, 5\}$, and the neighborhood radius was set to 1.0. We found $k = 5$ to be the best parameter for our experiments. For the MCTS[2], the tree depth $M_d$ was to 5 (i.e., look at 5 scenes in the future), the number of MCTS iterations $M_n$ was varied between $\{40,60,100\}$, the exploration parameter $c_{uct}$ was set to 4, and the discount factor $\gamma$ was set to 0.9. We found $M_n = 50$ to be the most feasible parameter for our experiments. Finally, for the reward calculations, we chose the weights $w1 = 0.6$, $w2 = 0.2$, and $w3 = 0.2$ based on manual tuning.

### 8.5.3 Results

We evaluated our approach across four tracks, two each from towns 3 and 5. We spatially divided each track into 5 scenes. Fig. 8.5 illustrates a snapshot of these tracks with different weather conditions. Track 1 runs around the downtown with high traffic density and has several traffic signs in most scenes. Track 2 runs around the suburb with an overpass and has low traffic densities for the different scenes. Track 3 runs around a long freeway with low traffic density and no traffic lights for all the scenes. Track 4 runs through a tunnel and then enters a city road with traffic lights and medium traffic density. We run each baseline 10 times

[2]The MCTS was entirely implemented by Baiting Luo.


Figure 8.5: Screenshots of the tracks as captured by the forward-facing camera attached to the AV. Track 1 runs across the downtown, track 2 runs across the suburb, track 3 runs across a freeway, and track 4 runs across a tunnel. The occlusion in the track 4 image is because of a camera failure.

| Switching | Location & Actions |
|---|---|
| **Accepted** | main road, freeway, overpass, traffic signal |
| **Prohibited** | intersection, roundabout |
| | pedestrian crossing the road |
| | turning |

Table 8.1: Domain rules rules used by the switching routine for controller transition.

around each track for these experiments. In each run, we assume the weather condition and the traffic density to remain the same across one scene because of its short duration. Therefore, we start the initial scene with a random weather condition and then incrementally vary the weather parameters for each subsequent scene. Further, Table 8.1 summarizes the domain-specific rules applied by the switching routine (see Section 8.4) in performing the controller transition.

**AV operation with no sensor failure**: We ran each baseline configuration across tracks 1, 2, and 3 without any sensor failure. As our objective is to maximize the system's performance while maintaining safety, we chose travel time to measure performance and the number of infractions the vehicle performs to measure safety. We define travel time as the time the vehicle takes to complete driving around a given track. The travel times of the different controller configurations around the three tracks are provided in Fig. 8.6. We observe among the controllers, the LBC controller has lower travel times. It has a median travel time of 118 and 277.83 seconds for tracks 2 and 3, respectively. However, the controller fails to complete track 1 (8 of the 10 times) because of off-road driving (a safety-related problem as the controllers' steering predictions were erroneous). The AP controller has a longer travel time with a median of 251 seconds and 562 seconds for tracks 1 and 3, respectively. The controller failed to finish track 2 (7 of the 10 times) because of a route timeout caused due to its slow speed (not a safety-related problem). We observe no such route completion problems with the simplex configurations. This is because they switch to the alternate controller in conditions where the primary controller fails. We also observe the *DSS* configuration to have shorter travel times than the other simplex configurations. It has median travel times of 195, 117, and 326 seconds. This reduction in time indicates that the configuration is more consistent in performing the reverse switch to the performant controller.

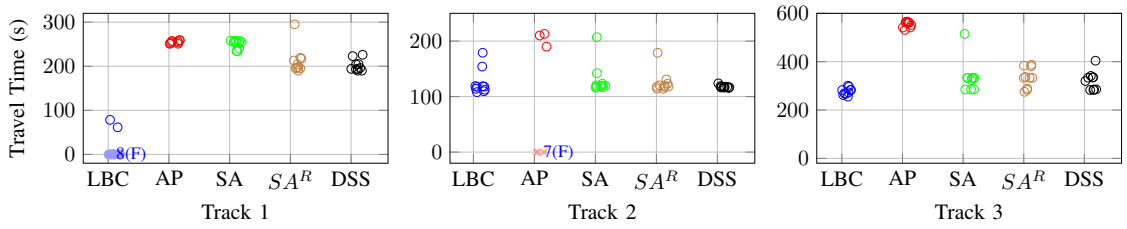



Figure 8.6: Travel times of AV without sensor failure. To compute the travel times, we ran each baseline 10 times around each track with different initial weather conditions. In most of the tracks, the LBC controller had the shortest travel time, and AP had the longest. Among the simplex configurations, DSS had shorter travel times. In track 1, the LBC controller did not complete the track 8 times because of off-road driving. In track 2, the AP controller could not complete the track 7 times because of a route timeout. We have marked the route in completion with travel times of 0 seconds and annotated the number of failure runs with a label.

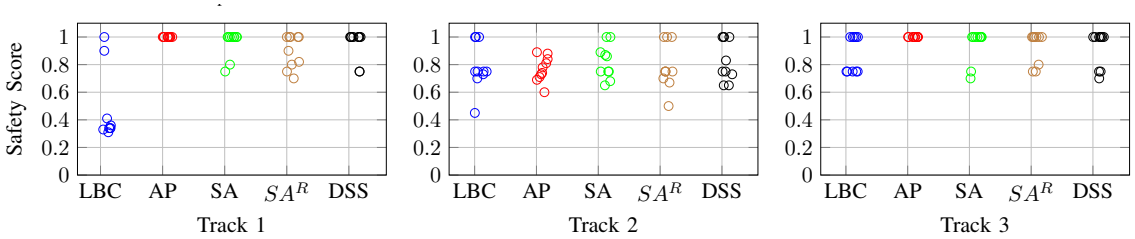



Figure 8.7: Safety score of AV without sensor failure. The safety score is measured as a weighted combination of infractions, including collisions, route completion, and off-road driving. Each configuration starts with an ideal score of 1.0 and reduces with the occurrence of infractions.

We compute a safety score combining three metrics as $0.5 \cdot RC + 0.5 \cdot Inf$. Where *RC* is the percentage of the route completed by the vehicle. $Inf$ are infractions computed as $(0.5 \cdot Col + 0.5 \cdot OR)$. *Col* is the number of collisions performed by the vehicle. *OR* is the off-road driving performed by the vehicle. The vehicle starts with a perfect safety score of 1.0, which is reduced on the occurrence of infractions. The safety scores of the baseline configurations for the three tracks are shown in Fig. 8.7. Among the controllers, the AP controller has a higher safety score across all tracks. However, because of its route in-completion on track 2 (refer to the previous paragraph), it had a median score of 0.74. The simplex configurations overcome this problem. They all select the performant controller to overcome the route completion problem. We observe that the traditional simplex configuration *SA* is the safest among all the simplex configurations. The *DSS* configuration has a comparable median safety score of 1.0 except for track 2, where it has a score of 0.79 compared to 0.79 of the *SA* configuration. In conclusion, the DSS configuration does not compromise much on safety compared to the traditional simplex architecture.

**AV operation with sensor failure**: We simulate a center camera occlusion for the AV operating on track 4 (see Fig. 8.5). We introduced the occlusion at 10 seconds into the simulation, which persisted throughout the vehicle's operation. Our first observation is that the occlusion severely affects the LBC controller, causing it to collide all the 10 times, resulting in a low median safety score of 0.44. The AP controller is unaffected

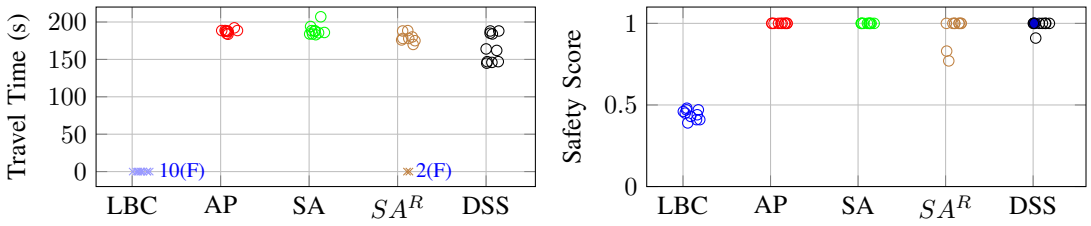



Figure 8.8: Travel times of AV with sensor failure. Travel times and safety scores for the AV with camera occlusion on track 4. (Left) Travel times: the LBC controller fails to complete the track, so we have marked the travel times at 0 seconds and annotated it with a label. The $SA^R$ configuration also fails twice because of a collision on switching to the LBC controller. (Right) Safety Score: The LBC controller has the least safety score because of the collisions, and the AP had the highest. Among the simplex configurations, *SA* has an ideal score of 1.0 across all runs.

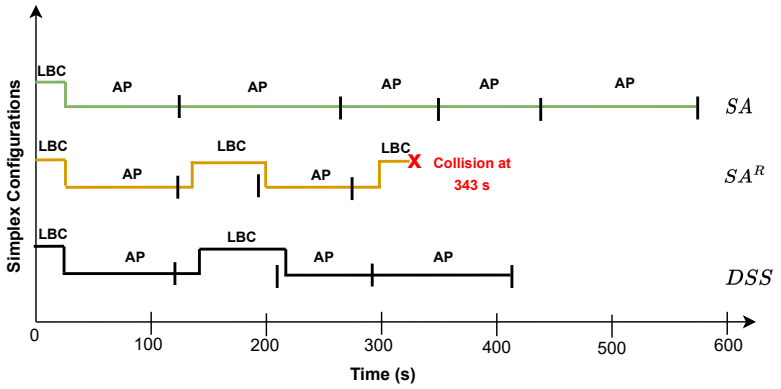



Figure 8.9: Controller transition of the three simplex configurations for one run across track 3. The track has 5 scenes indicated by the separators. The scene transitions happen at different times based on the controller being used. LBC controller is faster, so configurations using it will have faster scene transitions.

by the occlusion. So, it has a perfect safety score of 1.0 with a median travel time of 190 seconds. We also observed that the simplex configurations frequently switched to the AP controller to overcome the problem with the LBC controller. Among these, the *DSS* has shorter median travel times of 163 seconds, improving on the other simplex configurations. This reduction is because the occlusion does not affect the LBC controller's performance in all the scenes; the planner learns this. However, the other simplex configurations operate with the AP controller for most of the track.

**Controller transitions**: Fig. 8.9 shows the controller transition patterns of the simplex configurations that we observed for 3 of the 10 iterations across track 3. The black indicators mark the scene transition. For these iterations, the initial weather parameters for the first scene are set and incrementally varied for the other scenes. As the prior performance (computed from the LUT) of the LBC controller is low in scene 1, all three configurations switch to the AP controller. The control then remains with AP through the track for the *SA* configuration. However, $SA^R$ and *DSS* perform multiple switches among the controllers in scenes 3, 4, and 5.

We observe that $SA^R$ performs a reverse switch in scene 4, leading to a collision a short time after. However, the MCTS planner in *DSS* does not perform the reverse switch in scene 4, thus averting the collision.

**Times taken by the decision-makers and switching routine**: Fig. 8.10 shows the times taken by the decision logic of the three simplex configurations. To remind, the forward switching of the three configurations is performed based on prior belief. We use the K-NN algorithm on the 12500 simulation data for this. So, they all have an average decision time of $t_s = 0.13$ seconds. Also, the *SA* configuration does not perform reverse switching. The $SA^R$ configuration still uses the prior belief for reverse switching, so it has identical times to forward switching. The *DSS* uses the MCTS-based planner for reverse switching, which requires an average time $t_p = 2.5$ seconds. An influencing factor for the decision times is the size of the LUT. Finally, when a decision was made to switch, the switching routine took an average of $t_r = 1.39$ seconds during a forward switch and $t_r = 0.85$ seconds during a reverse switch. The difference in the time is because, during the forward switch, the speed of the performant controller had to be significantly reduced before the control could be transferred to the safety controller. However, in the reverse switch case, the safety controller's speed only required a little reduction before the transition could be performed.

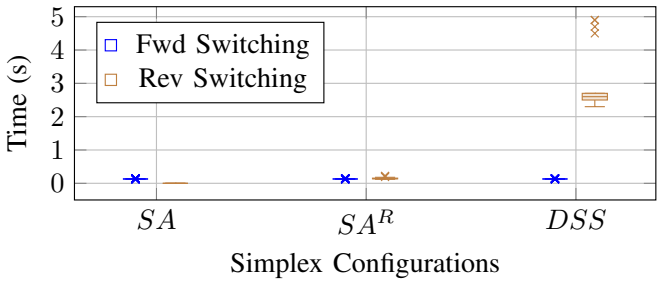



Figure 8.10: Execution times of the decision-makers. Time taken by the logic in *SA*, $SA^R$, and *DSS* configurations for performing forward and reverse switching. The forward switching for the three configurations is based on the LUT search, which takes $\sim 0.15$ seconds. The reverse switching times of *SA*, $SA^R$, and *DSS* configurations are 0, 0.15, and $\sim 2.5$ seconds, respectively.

**Resource utilization and Inference time**: Table 8.2 shows the computational resource utilized by different controller configurations. As seen, all the configurations mostly have a similar CPU usage. In terms of GPU, all the configurations other than the AP controller have a similar utilization. The utilization is mostly because of the ML-based controller and runtime monitors. In comparison, the AP controller has no ML components. Therefore, it does not utilize the GPU. We also observe that the AP controller takes the largest average execution time of 0.215 seconds and the LBC controller takes the least with an average of 0.031 seconds. The increased times in the case of the AP controller are because of the priority information (e.g., traffic signals, locations of other vehicles) that it must wait to receive from the simulator. The simplex configurations take an average execution time of less than 0.1 seconds. The increase in time is because these configurations often

| Controller Configuration | CPU | | GPU | | Inference Time (s) |
|---|---|---|---|---|---|
| | Util (%) | Mem (%) | Util (%) | Mem (%) | |
| *LBC* | 27.7 | 21.8 | 5.21 | 7.98 | 0.031 |
| *AP* | 28.4 | 23.4 | 0 | 0 | 0.215 |
| *SA* | 24.1 | 19.9 | 5.44 | 8.15 | 0.075 |
| $SA^R$ | 22.1 | 18.8 | 5.35 | 8.15 | 0.088 |
| *DSS* | 32.3 | 24.6 | 5.21 | 8.14 | 0.090 |

Table 8.2: Resource and time requirements of the different controller configurations. We report average values computed across 10 simulation runs.

switch to the AP controller. In conclusion, our approach only introduces no additional computational overhead as compared to the other simplex configurations.

## 8.6 Conclusions and Future Work

We present the dynamic simplex strategy for controller selection in controller redundant CPS. It has a logic that is both online and performs reverse switching. We model the logic as a sequential-decision problem and formulate it as an SMDP. The decision for a forward switch is performed by a selector based on the controller's past performance in the current state. The decision for a reverse switch is performed by a planner that simulates the future operating conditions using an online MCTS. Based on the switching decision, the switcher uses a set of domain-specific rules and the system's state for performing the switch. Experimental evaluations in CARLA simulations have shown our approach to improve the system's performance while not compromising safety. We plan to move this research in several directions. First, instantiate a chain of MCTS trees and average the scores across each tree to determine the optimal action. Second, validate the proposed dynamic simplex strategy for the DeepNNCar hardware testbed [80].

Chapters 4 to 8 have discussed the challenges with the dynamic assurance components (see Fig. 8.1 in Chapter 1) and presented a solution approach to address the challenges. However, one common challenge across these components is the need for abundant data. For example, the assurance case discussed in Chapter 4 requires ample data for curating the evidence necessary for the safety arguments. Similarly, computing the prior probabilities for the risk assessment component (Chapter 6) requires a lot of data from nominal (scene parameters within the training boundaries) and high-risk scenes (e.g., adverse weather conditions and sensor faults). In the current practice, data is generated manually by a designer who is aware of the nominal operating conditions and the failure conditions of the system [392]. However, for complex systems, this arbitrary data generation approach gets slow, and it requires a lot of manual effort and time to identify the hazards and the conditions in which the system fails. These problems can be alleviated with automation and by using

efficient and intelligent samplers to drive the data generation process towards high-risk scenes where the system is expected to fail. Therefore, an open question that we must address is *How can we automatically and intelligently sample data required for the dynamic assurance components?*

# Chapter 9

# Automated Adversarial Data Generation for CPS with LECs

## 9.1 Overview

As discussed in Chapter 1, designing each component of the proposed dynamic safety assurance framework requires a lot of data. Especially, designing the risk assessment component requires a large amount of labeled data from the system's nominal and high-risk operating modes (e.g., adverse operating conditions and sensor faults). While generating data from nominal operating conditions is straightforward, identifying and generating data from high-risk situations is difficult. A similar need for generating data from adversarial conditions is being faced in several CPS domains, especially in the automotive domain. For example, automotive companies like Tesla, Waymo, and Audi have spent the last decade driving their vehicles under different operating conditions to collect the data required for training their controllers and safety components. While each of these companies has driven more than a million miles [393, 394], they still do not have sufficient data from several operating conditions and events (e.g., snow, accidents) to design these components. Therefore, some companies like Waymo have recently started focusing on generating synthetic data from simulators [393]. Despite this growing requirement for synthetic data, their generation has mainly been manual, and only a few frameworks focus on automating the generation process [395, 396, 397, 398].

To address this need for an automated data generation mechanism, this chapter presents the "ANTI-CARLA", built on the popular open-source autonomous driving simulator CARLA [111]. The framework serves the dual purpose of (a) generating synthetic data and (b) testing autonomous driving pipelines. For this, the framework is equipped with several adversarial samplers in addition to the conventional grid and random samplers to generate high-risk scenes. It also provides a skeleton for plugging in any autonomous driving pipeline. The feasibility of the framework is demonstrated for generating high-risk data for several well-known autonomous driving controllers available through the CARLA autonomous driving challenge [399].

The work comprising this chapter has been accepted for publication in the 25th IEEE International Conference on Intelligent Transportation Systems [118]. This builds upon [119] which was published in IEEE International Conference on Assured Autonomy.

- **Shreyas Ramakrishna**[1], Baiting Luo[1], Christopher Kuhn, Gabor Karsai, and Abhishek Dubey. "ANTI-

[1]These authors have contributed equally. The Scenario Description Language, specification files, and samplers were done by Shreyas Ramakrishna. The adaptor and code integration into the simulator was done by Baiting Luo.

CARLA: An Adversarial Testing Framework for Autonomous Vehicles in CARLA" Accepted for publication in the 2022 IEEE Intelligent Transportation Systems Society Conference Management System (ITSC).

- **Shreyas Ramakrishna**, Baiting Luo, Yogesh Barve, Gabor Karsai, and Abhishek Dubey. "Risk-Aware Scene Sampling for Dynamic Assurance of Autonomous Systems." In IEEE International Conference on Assured Autonomy (ICAA), IEEE, 2021.

## 9.2 Introduction

**Problem Domain**: Design-time safety assurance approaches like the assurance case has been adopted by several safety standards such as the DO-178B [40], and ISO 26262 [41] and automotive companies [400]. However, recent accidents such as Tesla's autopilot crashes [23] and the Uber self-driving car accident [24] demonstrate that the design-time assurance is insufficient. As being pointed out in this work, these systems require a dynamic approach with proactive safety assessment components like OOD detectors [93, 116], failure predictors [385, 401] and dynamic assurance monitors [117] is required to enhance the safety assurance of these systems. In addition, these systems need to be subjected to thorough testing and validation before being deployed for real-world operations. These activities often require labeled data from different operating modes of the system that belong to scenes with adverse operating conditions and sensor or actuator faults. These scenes are referred to as risky scenes [154] or safety-critical scenarios [402]; in this paper, we refer to them as high-risk scenes [2].

**State-of-the-art and Limitations**: Often, the data related to high-risk scenes are under-represented in the training sets [403], leading to a data imbalance problem. If we can generate these under-represented events, they can be used to design the safety assessment components required for dynamic assurance and retrain the controller LECs to improve their accuracy [287]. However, collecting real-world data of such high-risk scenes can be expensive and slow in real-world conditions. Hence, recent research has been increasingly focusing on using synthetic data from simulators such as AirSim [404], LGSVL [405], Deepdrive [406], and CARLA [111]. CARLA is particularly focused on autonomous driving and is a widely used option in academia and industry for this domain [117, 92, 116, 5].

Traditionally, synthetic data generation is restricted to a set of handmade scenarios with the goal of satisfying safety standards such as ISO 26262 [392]. However, manually creating test cases is tedious and requires significant human labor. Recently, there has been substantial ongoing work in automating the generation process. Domain-Specific scenario description languages (DSMLs) such as Scenic [287] and

[2] We use "scenes" and "test cases" inter-changeably throughout the paper

MSDL [153] are being developed for describing what a test case should contain. They provide a mechanism for describing the test conditions, the variable parameters, and their distribution, which is required for generating the test cases. These parameters are then sampled from their distribution ranges using state-of-the-art sampling algorithms such as random search, grid search, and Bayesian optimization search for generating data from high-risk scenes [407, 154, 148, 155].

Only a few frameworks have been proposed that focus on automatically generating data for autonomous driving [395, 408, 397, 398]. These frameworks are built on custom simulators that are not open-source or available for use. Also, the conventional samplers used by these frameworks have several limitations. First, they perform *passive sampling*, which does not use the feedback of previous results in the sampling process. Second, the scene variables (e.g., environmental conditions) being sampled typically have *sampling constraints* and *co-relations* that need to be considered. For example, environmental conditions (e.g., precipitation) may have physical constraints on their values that govern their temporal evolution. Applying these constraints is necessary for generating meaningful scenes [287]. However, the conventional samplers do not include these sampling constraints. Third, conventional samplers do not balance the *exploration vs. exploitation trade-off*. For example, random and Halton searches prioritize uniform search space coverage, so they only explore. In contrast, grid search aims to cover a given grid exhaustively, so they only exploit it. However, as discussed by Jerebic, Jernej *et al.* [409], balancing these strategies can result in higher coverage diversity which is commonly measured using clustering properties like the number of clusters and cluster population.

**Contributions**: To address these issues with data generation, we present ANTI-CARLA, a framework for automated data generation, adversarial testing, evaluation, and exploration of the performance of AVs within the open-source CARLA simulator. It provides a skeleton that allows for plugging in and testing any autonomous driving pipeline. It includes a scenario description language (SDL) for describing the vehicle's operating conditions and a simple interface for specifying the conditions. The language is supported by two adversarial samplers called the random neighborhood search (RNS) and the guided Bayesian optimization search (GBO), in addition to the conventional random and grid search. These adversarial samplers perform active sampling by using previous simulation results in the sampling process. They also include sampling constraints that govern the evolution of scene variables. In addition, they provide explicit hyperparameters to control the tradeoff between exploration vs. exploitation of the search strategy. Further, to facilitate the samplers, we use a novel risk-based scoring function that evaluates how risky the scene was for the vehicle. We use ANTI-CARLA to evaluate the state-of-the-art Learning By Cheating (LBC) controller [5]. Despite LBC achieving an accuracy of 100% in parts of the CARLA challenge, ANTI-CARLA generates several operating conditions that fail the controller. We also show our samplers to outperform the conventional random sampler in generating high-risk test cases.

**Outline**: The rest of this chapter is organized as follows. In Section 9.3, we summarize related research. We discuss the problem formulation in Section 9.4 and introduce the proposed framework in Section 9.5. In Section 9.6, we evaluate the framework by generating fail cases for the LBC controller and analyzing them. Section 9.7 concludes. The source code of this work can be found under https://github.com/scope-lab-vu/ANTI-CARLA.

## 9.3 Related Work

In this section, we discuss related work. First, we introduce how different driving scenes can be described and sampled. Then, we summarize existing frameworks that allow for generating different scenes for a given driving pipeline. Finally, we summarize state-of-the-art driving pipelines for CARLA.

### 9.3.1 Test Case Description and Sampling

Tools like Dakota [410] with sampling approaches like incremental sampling, importance sampling, and adaptive sampling have been long available for uncertainty quantification in engineering design. Such samplers have recently gained interest in sampling simulation-based scenes in the AV domain. Several domain-specific SDL like Scenic [287] and MSDL [153] have been designed for specifying scenarios of complex traffic conditions and operating conditions to identify counterexamples that affect the system's safety. These languages use probabilistic samplers that sample scenes across the entire search space formed by combining scene variables. For example, Grid search is a popular passive sampler, which takes a single risky scene as the starting condition and generates similar risky scenes around it. However, manual tuning makes the grid sampler labor-intensive and time-consuming. For accelerating the search process, an alternate approach [407] uses importance sampling to identify important scene variables and sample scenes using them. Worst-Case Scenario Evaluation [154] is another approach that uses model-based optimization to identify the weakness of the system and uses this information to generate worst-case disturbances. VERIFAI [148] software toolkit has several passive samplers like random search, Halton search [411], and active samplers like cross-entropy optimization and Bayesian optimization. However, these samplers do not effectively balance the exploration vs. exploitation trade-off. Viswanadha, Kesav, *et al.* [155] recently proposed a multi-armed bandit sampler that balances the two strategies.

Also, recently, there has been growing interest in using generative models for generating risky scenes [412, 413, 414, 402]. In this approach, scenes are randomly sampled from a distribution learned by a generative model instead of sampling from the entire search space (as performed by the samplers discussed above). For example, Ding, Wenhao, *et al.* [402] use an auto regressive model to factorize the scene variables into

conditional probability distributions, which are then sampled to generate risky traffic scenes. In a prior work [412], they also used a variational autoencoder to project high-dimensional traffic information into a lower-dimensional latent space, which is sampled to generate critical traffic scenes. Vardhan *et al.* [413] uses a Gaussian Mixture Model to find the corner case scenes in the training set that is not well learned by the system. Another approach [414] uses generative adversarial imitation learning to perform adaptive importance-sampling to learn rare events from an underlying data distribution.

We integrate both passive and active samplers into our framework. Due to the modular design of ANTI-CARLA, other intelligent samplers [410] can be integrated easily.

### 9.3.2 Testing Frameworks

Simulation-based testing frameworks allow generating test cases for a given system. A framework consists of two main components: a mechanism for describing test cases, and a mechanism for generating test cases. Tuncali *et al.* [415] presented a testing framework built using their MATLAB tool called S-TaLiRo. The tool provides an optimization engine and a stochastic sampler to automatically sample test cases across the search space generated from the testing specification. The same authors later presented a simulation-based adversarial testing framework called Sim-ATAV [396], which performs adversarial testing in the scenario configuration space of perception-based AVs. Son *et al.* [397] proposed an advanced driver-assistance system (ADAS) testing framework based on co-simulation of Siemens Amesim and Prescan software. The testing is driven by several handmade scenarios derived from safety standards such as ISO26262. The Paracosm framework [398] allows users to describe complex driving situations as programs. It provides a systematic approach for exploring the search space and contains a coverage metric to quantify the coverage. For Grand Theft Auto, a simulation-based harness for AV is available [395]. It includes a testbench that performs sampling using simulated annealing to sample the next simulation parameters based on the AV's performance in the current simulation. While all of these works focused on AV testing, they are implemented either in proprietary simulators or in limited custom scenarios.

To the best of our knowledge, no framework is readily available for testing AVs in open-source simulation. Despite CARLA being a widely used simulator across academia and industry, a flexible framework for testing AV is currently unavailable for CARLA. This motivates the introduction of ANTI-CARLA for adversarial testing of arbitrary driving pipelines with minimal effort.

### 9.3.3 Autonomous Driving Pipelines

The proposed framework is designed to generate fail cases for a given autonomous driving pipeline. A range of controllers that perform well in CARLA is available. The Transfuser approach [416] uses transformers to fuse lidar and camera input. The World-on-Rails pipeline [417] only relies on camera images. It simplifies the driving task by assuming that the vehicle does not influence the environment, then uses reinforcement learning to obtain a driving policy. Finally, the LBC approach [5] is also based on visual input. First, a teacher agent with complete access to internal simulator information is trained to imitate expert trajectories. Next, the actual controller is trained to imitate the teacher agent's trajectories. By learning from a teacher who was "cheating" by having complete information, the student network is capable of learning highly accurate driving strategies. LBC achieved an accuracy of 100% on the original CARLA benchmark [5]. Current benchmarks are thus not always capable of identifying weaknesses of a controller, demonstrating the need for adversarial testing as offered by ANTI-CARLA.

## 9.4 Problem Formulation

Formally, the problem can be defined as follows. We have a simulator *Sim* that is given the current environment $E_t$, an AV with a driving controller $C$, and the current state of the AV $state_t$. The state of the vehicle $state_t \in \mathbb{R}^n$ describes the vehicle's position, speed, and steering angle. The current environment $E_t$ can be defined in terms of a set of temporal variables such as weather, traffic density, the time of day that change over time, and a set of spatial variables that describe roadway features. These features are characterized by waypoints $w$ denoted by a two-dimensional matrix of latitude and longitude. These waypoints can be mapped to different road segments that constitute a track on which the AV is tested. Further, the controller $C$ is a function that perceives the environment using measurements from multi-modal sensors such as a camera, lidar, radar, etc., to compute actuation controls of speed, throttle, and brake. The control signals are sent to the simulator, which generates the next state of the vehicle $state_{t+1}$ by running the controller $C$ under the given environment variables $E_t$: $Sim(state_i, E_i, C) = state_{i+1}$. An ordered sequence of $n$ consecutive states $state_i$, $i \in \{t-n, ..., t\}$ is considered a *scene*. The simulator has some infraction function $I$ that records for each state its infractions $I(state_t) \in \mathbb{R}_n$. A scoring function $TS$ assigns a score to the entire scene: $TS(scene) = \sum_{k=t-n}^{t} I(state_t)$. Then, we want to solve the following problem.

**Problem 3** *Given a simulator Sim, a driving controller C, and some conditions on the environment E, generate a set of high-risk test cases that results in high infraction scores TS.*

Addressing this problem requires a framework that allows for (1) specifying the operating conditions in

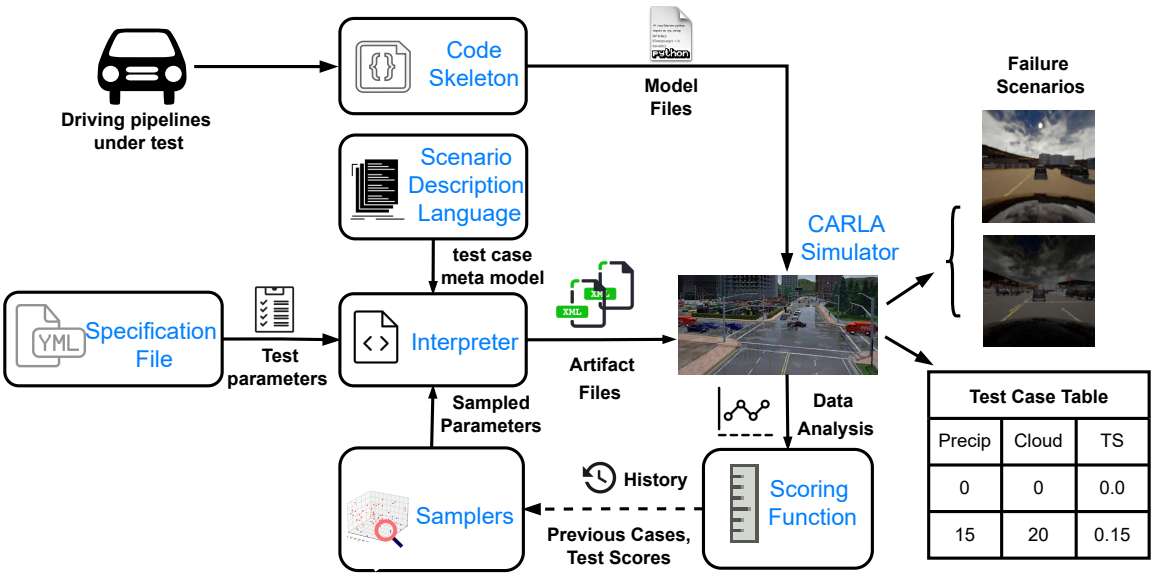



Figure 9.1: Overview of the ANTI-CARLA framework for generating test cases that fail an AV system in the CARLA simulator. The driving pipeline and the test specifications are taken as inputs by the framework to generate failure test cases that can be post-processed to analyze the problems with the system.

which the system vehicle is expected to operate. (2) finding the conditions that could result in a high infraction score, and (3) integrating a given controller $C$ into the given simulator $Sim$. In the next section, we introduce ANTI-CARLA to implement these requirements.

## 9.5 ANTI-CARLA Framework

In this section, we discuss the proposed ANTI-CARLA framework and its components shown in Fig. 4.4.

### 9.5.1 Scenario Generator

The scenario generator includes a scenario description language for modeling scenarios and specification files for specifying testing parameters. We define a test case in terms of a scene, which is a time-series trajectory of the system's path in the environment that lasts 30 seconds to 60 seconds. A scene is represented using the environmental conditions ($E$) and the AV system's parameters ($A$). $E$ can be defined using structural features (e.g., type of road and road curvature) and temporal features (e.g., weather and traffic density). $A$ includes information like the starting position, onboard sensors, and actuators. Together, $E$ and $A$ form the testing parameter set. The value for some of these parameters can be sampled from specified distributions. Further, the sampling process is governed by a set of physical constraints that limit the rate at which these parameters can evolve. For example, the time of day has a fixed rate at which it can change. Including these constraints during sampling results in more meaningful scenes, as shown in our previous work [119].

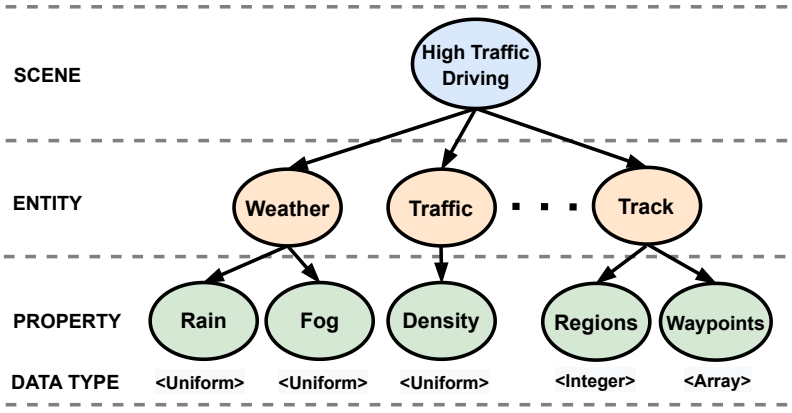



Figure 9.2: Example for the structure of the proposed SDL meta-model.

**Scenario description language**: We have designed an SDL in the textX [288] meta-language for modeling a scene. The grammar contains rules for describing a scene in the meta-language. A meta-model contains the actual description of a scene. A visualization of the structure of the meta-model is shown in Fig. 9.2. A scene $s$ is a collection of entities $\{e_1,e_2,...,e_k\}$ that represent different environmental conditions and agent parameters. For example, the scene *High Traffic Driving* is specified by its *Track* as well as its *Weather* and *Traffic* conditions. Entities are further specified by a set of properties $\{p_1,p_2,...,p_m\}$. For example, the *Weather* can have the properties *of Rain* and *Fog*. Each property has a name $n$ and a data type $t$. For example, *Rain* is a uniform distribution, while *Waypoints* are an array of integers. Special data types such as distributions can again have properties, e.g., the range of a uniform distribution. The meta-model allows for a structured description of any desired scene in the CARLA simulator.

**Specification Files**: We also provide a set of specification files with the scene parameters that the user can select. These files serve as an abstract representation of the SDL modeling concepts. Based on the user's selection of the sampler, the chosen parameters are sampled from their respective distributions. The remaining parameters are assigned default values. We divide the information into three specification files. First, the user selects a scene specifier. We show an excerpt of the scene specifier file in Fig. 9.3. It specifies which town to use as a map, which track to drive, the distributions for the weather traffic density, pedestrian density, and the sampling constraints. The user also specifies which infraction metric to use and at which frequency data should be recorded. Second, an agent specifier is needed, which includes agent-related information such as the available controllers, a list of sensors and their positions, and the sensor data that can be recorded. Third, a sampler specifier determines which sampler to use from a list of available samplers.

Finally, the language has an interpreter, as shown in Fig. 9.1. It connects the specification files, the SDL, and the probabilistic samplers discussed in the following section. First, the interpreter extracts the parameters

```
Scenario Description{
    town: 5 //Available towns 3 and 5
    track: 1 // 1 track available for each town
       regions: 5 //Each town has 5 regions
    weather:  //Weather parameters and distribution range
      cloudiness: [0,100]
      precipitation: [0,100]
      time-of-day: [-90,90]
    pedestrian_density: [0,3]
    traffic_density: [0,10]
    Constraints: //A constraint on the rate of change in parameter
       values
        weather_delta: 2
        traffic_delta: 2
        pedestrian_delta: 1
    Infraction_Metrics: //Infraction metrics to be recorded
        Infraction Penalty: true
        Off-road Driving: true
        Route Deviation: false
    Record Frequency: 5Hz } //Frequency of data recording
```

Figure 9.3: Excerpt of a scene specification file. The specification files describe the available inputs, the user then only has to select a value for each concept.

that require sampling and the fixed parameters from the specification files. It then sends the parameters that need sampling to the sampler. The sampler generates a value for these parameters from their distribution ranges, which is parsed to the SDL to generate artifact files that drive the simulator.

### 9.5.2 Adapter Glue Code

The framework also requires a driving pipeline. Integrating different pipelines is not straightforward since they might not have the right interface to be used "as in" the framework. For example, driving pipelines developed outside of CARLA may not be directly used in the simulator because of strict interface requirements. They may need to be "adapted" to meet the interface expectations of the simulator [418]. To address this, we generate an adapter that interfaces the driving pipeline code with the sensors and actuators in the format required by the simulator's API as shown in Figure 9.4. The adapter is synthesized from the agent specification file, which has a list of sensors required by the driving pipeline, its positions, and the sampling rates.

The adapter reads the available sensors from the autonomous agent class in the simulator's API and extends it with the sensors requested in the specification file. Thus, a code structure for the requested sensors is generated and provided to the driving pipeline code. Also, the actuators required by the simulator are read from the API and made available in the code. With the required sensors and actuators available through the adapter, the user needs to provide the driving pipeline code. There are specific directories for any utility files and model

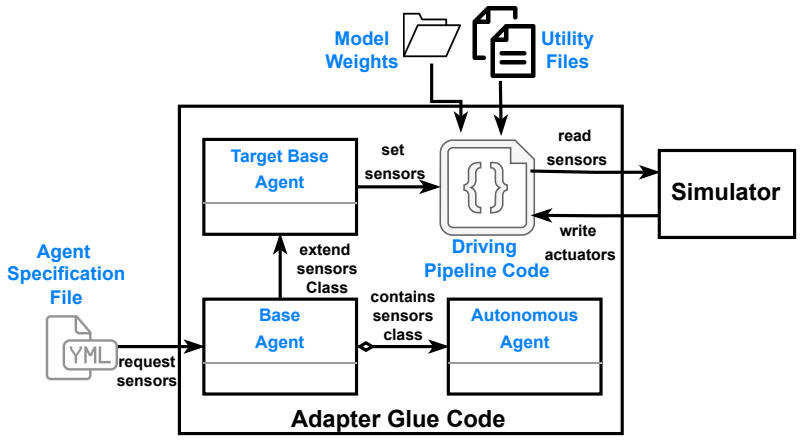



Figure 9.4: An illustration of the adapter glue code that interfaces the sensors and actuators of CARLA to the code of the AV system.

weights of the controller, which are linked with the code skeleton. The user only needs to handle this interface for setting up their controller correctly. If the actuators and sensors are not properly set up, the simulator will throw an error. In future work, we will include an automatic check for ensuring a correct-by-construction setup between the code, sensors, and actuators.

### 9.5.3 Scoring Function

For evaluating the driving proficiency of the AV system in the generated scene, the framework also provides a scoring function called test score $TS$. The scoring is based on all of the infractions performed by the system in a given scene. Infractions measured in CARLA include route deviation, lane violation, traffic rule violation, running a stop sign, running a red light, and off-road driving. Each infraction $I_k$ is assigned a weight $w_k$. In this work, these weights are set to the values used in the CARLA challenge [399]. However, they can be varied depending on the use case. The infractions are then combined into the weighted score as shown in Eq. (9.1):

$$TS = \sum_{k=1}^{n} w_k \cdot I_k \tag{9.1}$$

The test scores $TS$ generated from the infractions are stored along with the test case parameters in a test case table as shown in Fig. 9.1. These test cases and the scores are used online to drive the active samplers towards regions of the search space that have previously resulted in high test scores. The table can also be used to perform an offline post-analysis to identify the failure conditions of a given controller, allowing for retraining, and improving the controller.

### 9.5.4 Samplers

We have integrated several samplers to perform search-based test case generation. Here, a sampler is interfaced to the SDL through an interpreter, which provides it with the names, distribution, and the current value of the parameters. We included two kinds of samplers available in the framework.

First, we implemented several passive samplers since they are fast and widely used. They do not use the feedback of previous results in the sampling process. *Random Search*, uniformly samples the parameter value from their respective distributions at random. *Grid Search* exhaustively searches all of the combinations of the parameters in a given grid. *Halton Sequence Search* [411] is a pseudo-random technique that samples the parameters using co-primes as their bases. While these samplers perform well, their non-feedback sampling approach results in a directionless search that could miss several important failure test cases. Further, they do not balance the exploration vs. exploitation of the search space, which is required for generating diverse failure cases [119].

To overcome these limitations, we have also included two active samplers, *RNS* and *GBO* [119] designed in our previous work. These samplers use the feedback of the AV system's previous performances when sampling the parameters for the current test case. The feedback uses the test score and the previously sampled parameters. In addition, they also capture constraints and correlations between the different test parameters.

#### 9.5.4.1 Random Neighborhood Search (RNS) Sampler

The *RNS* sampler extends the conventional random search with the kd-tree nearest neighborhood search algorithm [386]. This extension provides the random sampler with the capability to exploit. The region around these parameter values is exploited if the test case generated from randomly sampled parameters results in a high test score. Otherwise, the parameters are again randomly sampled from the entire distribution. Algorithm 10 illustrates the steps involved, and it works as follows.

*First*, the algorithm explores the search space $\mathscr{D}$ by randomly sampling the scene variables $s_v$ from their respective distribution range $v_d$. These randomly selected variables are used to generate the scene artifact files, run the simulator, and compute the test score $TS$.

*Second*, if the $TS < \delta$, the statistics of the scene are stored, and the scene variables are re-sampled again from their respective distribution range $v_d$. However, if the $TS > \delta$, the neighborhood around the randomly sampled scene $R$ is exploited to generate more scenes. For this, we create a bounded region $\mathscr{B}$ around $R$ by applying the sampling constraints $\mathscr{SC}$ to the distribution ranges of the scene variables.

*Third*, the algorithm uses the kd-tree nearest neighborhood search to find if there are at least $k$ scenes generated in the neighborhood of the explored scene $R$. It checks if $R$ has at least $k$ similar scenes in $\mathscr{E}$. To

**Algorithm 10** Random Neighborhood Search

**Parameter**: number of iterations $t$, explored list $\mathscr{E}$, neighborhood size $k$
**Input**: search space $\mathscr{D}$, sampling constraints $\mathscr{SC}$, scene variables $s_v$, threshold $\delta$
**Output**: List of test cases

```
for x = 1,2,...,t do
    if S_Risk < δ or (TS > δ and N ≥ k) then
        Randomly sample s_v within 𝒟 to generate random scene R
    else
        Apply 𝒮𝒞 to create bounded search area ℬ
        Randomly sample s_v within ℬ to generate neighboring scene N
        Apply kd-tree to find if there are atleast k neighbors (N) in ℰ
    end if
    Use the sampled variables to generate the scene artifact files
    Simulate the scene and compute the TS
    Append sampled variables and TS to ℰ
end for
```

measure the similarity, we use the $l_2$ distance metric ($d(x,R) = ||x-R||_2$) and identify the similar scenes as $N = \{\forall s_l \in \mathscr{E} | d(R,s_l) < \tau\}$. Where $R$ is the currently sampled scene, and $s_l$ are previously explored scenes in the explored list $\mathscr{E}$. This step returns a list of neighbors whose distances to $R$ are smaller than a threshold $\tau$. If the number of the neighbors $N \geq k$, the search around $R$ is stopped. Then, the first step is repeated to randomly explore a new scene in the entire search space $\mathscr{D}$. However, if $N \leq k$, then the search around $R$ is continued. In this sampler, $k$ is the hyperparameter used to control the exploitation.

While the sampler can both explore and exploit, the uninformed exploration without strategic knowledge about the search space could result in several regions of the search space left unexplored.

#### 9.5.4.2 Guided Bayesian Optimization (GBO) Sampler

The uninformed exploration of the RNS sampler is addressed by our second sampler, which has two components to perform informed exploration and active sampling. First, a probability function model is fitted across all the previously explored variables. This feedback allows the model to learn the search space region with large uncertainty that will return a higher objective value. A Gaussian Process (GP) model is used to fit the function. Second, an acquisition function that strategically finds the succeeding variables to optimize the objective function. We use the Upper Confidence Bound (UCB) [419] acquisition function that provides the $\beta$ hyperparameter to control the exploration vs. exploitation trade-off (See Eq. (9.2)). We also include the sampling constraints to restrict the region where the acquisition function looks for the next sampling variables. Algorithm 11 shows the operation of the GBO sampler, which is discussed below.

*First*, the GP model with properties $\mu[0]$ and $\sigma[0]$ is initialized for the first k iterations. During these iterations, the scene variables $s_v$ are randomly sampled from their respective distribution range $v_d$. These

**Algorithm 11** Guided Bayesian Optimization

**Parameter**: number of iterations $t$, initial iterations $k$, explored list $\mathscr{E}$
**Input**: search space $\mathscr{D}$, sampling constraints $\mathscr{SC}$, scene variables $s_v$, threshold $\delta$
**Output**: List of test cases

```
for x = 1,2,...,t do
    if x ≤ k then
        initialize GP model with random samples s_v from 𝒟
    else
        Apply 𝒮𝒞 to create bounded search area ℬ
        Use μ_t and σ_t in the UCB function to sample s_v within ℬ
    end if
    Use the sampled variables to generate the scene artifact files
    Simulate the scene and compute the TS
    Append sampled variables and TS to ℰ
    Update the GP model using ℰ. Update μ_t and σ_t
end for
```

randomly selected variables are used to generate the scene artifact files, run the simulator, and compute the risk score $TS$ that is added to the exploration list $\mathscr{E}$ along with the sampled variables. After the k initial iterations, the initialized GP model is fitted across all the entries in $\mathscr{E}$ to calculate new posterior distribution $f(x_n)$ with updated $\mu[x_n]$ and $\sigma[x_n]$.

*Second*, the algorithm uses the sampling constraints $\mathscr{SC}$ to create a smaller search space $\mathscr{B}$ in which the acquisition function will sample the succeeding variables. The updated posterior distribution and the bounded search space created by the sampling constraints are used by an acquisition function to strategically select the following scene variables that will optimize $TS$. In this work, we use the UCB as the acquisition function:

$$x_{n+1} = \underset{x \in \mathscr{D}}{\operatorname{argmax}}\, \mu[x_n] + \beta^{1/2} \cdot \sigma[x_n] \tag{9.2}$$

$x_{n+1}$ are the newly selected variables with the largest UCB. $\mu[x_n]$ and $\sigma[x_n]$ are the properties of the updated GP model. $\beta$ is a parameter that allows for exploitation and exploration tradeoff. A larger value of $\beta$ allows for higher exploration, and a lower value allows for exploitation. So, $\beta$ needs to be carefully selected for optimizing the two strategies. The newly selected variables are used to run the simulation and compute $S_{Risk}$, which is added to the list $\mathscr{E}$ along with the selected variables. The steps of the algorithm are repeated for a specified number of iterations $t$.

The sampler has two problems that arise because of using the GP model. (1) Cold start requires the model to be trained from scratch each time the sampler is used. This increases the sampling time. To address this, we use the knowledge of randomly sampled scenes from the previous runs to "warm start" the search process. (2) Scalability, the GP model suffers from a cubic time complexity [420], which limits its applicability to a large number of executions.

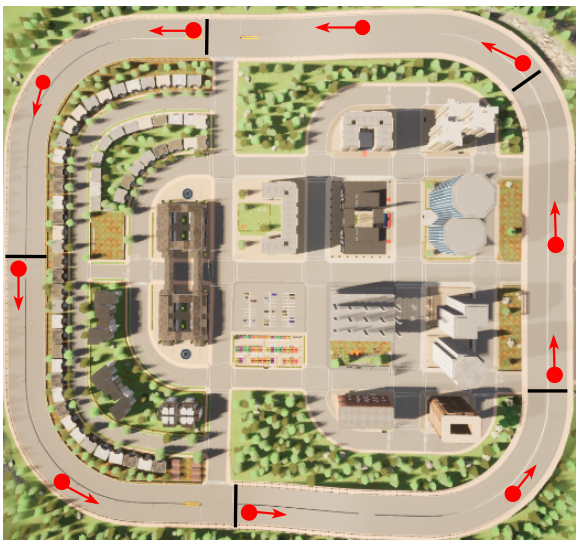


Figure 9.5: An illustration of the track in CARLA's town5. The waypoints are highlighted by red arrows and the five regions are separated by black lines.

## 9.6 Evaluation

In this section, we present several experiments to evaluate the usability of ANTI-CARLA for adversarial testing. We use the framework to compare the adversarial test cases found by different samplers. Then, we compare the performance of the LBC controller and the transfuser controller [416] on those test cases, analyze the failures and make suggestions on how to improve the controllers. The experiments were run on a desktop computer with AMD Ryzen Threadripper 16-Core Processor, 4 NVIDIA Titan XP GPUs, and 128 GiB RAM.

### 9.6.1 Simulation Setup

The proposed framework is integrated into the CARLA simulator. We used the CARLA challenge API [399] to create one closed-loop track in both town5 and town3. We use LBC as the controller $C$ under test. The geometry of the track is defined by ten waypoints as shown in Fig. 9.5. We divide the track into several regions containing two waypoints each. For each track, we thus obtain five regions. We divided each track into regions to create shorter scenes in which we can vary the weather conditions as well as traffic and pedestrian densities. Each track can then have ten different environmental conditions. The length of the track and the number of regions can be specified in the scene description file and can thus be changed with minimal effort in the code. We use the API to create and control the traffic and pedestrians in each scene. For evaluating each simulation, we record the driving score, the route completion score, and the test scores $TS$ based on the list of infractions available from the API.

(a) Town5, t=1, c=10%, p=5%, d= $8^\circ$ (b) Town5, t=10, c=70%, p=60%, d= $24^\circ$ (c) Town3, t=4, c=20%, p=0%, d= $60^\circ$ (d) Town3, t=2, c=40%, p=0%, d= $0^\circ$

Figure 9.6: Screenshots of the test cases as captured by the forward-looking camera of the AV. Descriptions of these scenes are provided below the images.

#### 9.6.1.1 Comparison Metrics

Besides evaluating the performance of the driving pipeline, the framework also allows us to evaluate the performance of the samplers used to generate adversarial test cases. We use the following two metrics.

**Failed Test Cases (FT)**: This score measures the efficacy of the samplers in generating failure test cases. It is calculated as the number of failed test cases $N_{Fail}$ compared to the total number of sampled test cases $N_{Total}$:

$$FT(\%) = \frac{N_{Fail}}{N_{Total}} \cdot 100 \tag{9.3}$$

**Total Execution Time**: The overall time taken by the sampler to sample $N_{Total}$ test cases and execute them in the simulator is a metric relevant for practice.

### 9.6.2 Results

We generated 100 test cases each for the track in town3 and town5. Each test case represents a scene that lasts between 30 seconds to 60 seconds. We varied the environment parameters of cloudiness $c$, precipitation $p$, time of day $d$, and traffic density $t$ to generate different test cases. We varied the cloudiness and precipitation in the range $[0,100]$, the time of day in the range $[0,90]$, and the traffic in the range $[0,50]$. We used the initial conditions of $d = 0^\circ$ (dusk), $c = 0^\circ$, $p = 0^\circ$, and $t = 5$. To score the test cases, we computed a test score as $TS = R_i \cdot I_S$. Here, $R_i$ is the route completion percentage of the $i^{th}$ route, and $I_S$ is the weighted sum of major infractions such as collisions with pedestrians, collisions with vehicles, collisions with static objects, and minor infractions such as running a stop sign, running a red-light signal, and off-road driving. The weights for these infractions are taken directly from the CARLA challenge setup. To drive the test case generation process, we selected the commonly used random sampler as a baseline and compared it against the RNS sampler. The simulator was run in synchronous mode at a fixed rate of 20 frames per second. If a test case includes a collision, infraction, or route in-completion, it is considered a failed test case.

#### 9.6.2.1 Visualization

First, we visualize several test cases by showing the frontal camera view from each scene. In Fig. 9.6, we show four exemplary test cases for LBC. Fig. 9.6-a is a nominal test case from town5 with low precipitation and dusk time of the day and low traffic. Fig. 9.6-b is a failed test case from town5 with high precipitation and traffic. A collision occurred when the controller tried to steer to the right lane. Fig. 9.6-c is a failed test case from town3 with no precipitation and low traffic. Here, the AV can be seen navigating a tunnel. Towards the end of the tunnel, the AV collides with a pillar. Finally, Fig. 9.6-d is another failed test case from town3, where the AV runs a red traffic light. These examples demonstrate that a diverse set of fail cases could be obtained.

| **Samplers** | **Town** | **Failed Test Cases (%)** | **Test cases with collisions (%)** | **Test cases with infractions (%)** |
|---|---|---|---|---|
| **Random** | 3 | 17 | 9 | 10 |
| | 5 | 13 | 7 | 8 |
| **RNS** | 3 | 27 | 13 | 15 |
| | 5 | 21 | 10 | 13 |

Table 9.1: Test case statistics for the random and RNS sampler.

#### 9.6.2.2 Sampler Comparison

Table 9.1 shows the statistics of the collisions and infractions generated by the two samplers across all test cases. The RNS sampler generates a higher number of failed test cases than the random sampler. In general, the AV performed better in town5 than in town3. The track in town5 was shorter, had fewer traffic lights and stop signs, and did not have complex landmarks such as a tunnel or a roundabout. The controller failed 13% and 21% of the test cases generated by the random and RNS sampler, respectively. Town3 has several traffic lights and a tunnel. Here, the controller failed 17% and 27% of the test cases generated from the random and RNS sampler, respectively. These failure cases occurred in the region that included the tunnel and areas with traffic and stop signs nearby. The framework required 140 minutes to generate the test cases for town5 and 325 minutes for town3 when using the random sampler. With the RNS sampler, the execution times are 176 minutes for town5 and 384 minutes for town3. This shows that the more efficient RNS sampler does not add significant overhead, with both samplers requiring around five minutes per test case.

Fig. 9.7 shows the test cases sampled by the random and RNS sampler plotted in the operating conditions space. The failed test cases are marked in red, and the test cases that passed without failures or collisions are marked in blue. The random sampler randomly samples the test cases across the search space. This makes it hard to analyze common causes of the controller's failures. In contrast, the failed test cases generated by the

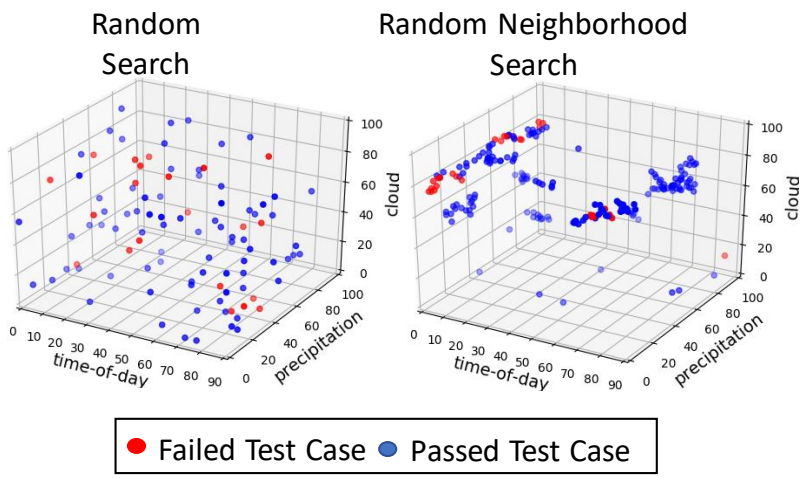



Figure 9.7: Comparison of the 100 scenes sampled by the random and RNS samplers. The RNS sampler generates distinct clusters of failures. Plot axis: x-axis represents the time of day, the y-axis represents the precipitation level, and the z-axis represents the cloud level.

RNS sampler occur in clusters, making it easier to hypothesize the causes of the failures. Fig. 9.7 shows that the controller had failures mostly in two operating conditions: (a) high precipitation and (b) dusk as the time of the day.

#### 9.6.2.3 Controller Comparison

ANTI-CARLA can also be used to compare different controllers. We compare the performance of the transfuser [416] controller to LBC using the track in town5. We ran the transfuser and LBC for the same 100 test cases generated with the random sampler. The LBC and transfuser controllers had 13 and 23 failed test cases, respectively. Fig. 9.8 shows a breakdown of the infractions caused by these controllers. Controller timeout is the main reason for the transfuser's higher failure rate. This could be due to the expert policy used during training not being sufficient for handling all scenarios. Besides the frequent timeouts, the transfuser had significantly fewer collisions and infractions than LBC. Both controllers struggled with detecting red traffic lights. The authors of transfuser suggested that this is due to traffic lights being placed on the opposite side of intersections, which is difficult to detect in the images captured by forward-looking cameras [416]. These results show how ANTI-CARLA allows the identification of weaknesses of different controllers.

#### 9.6.2.4 Recommendations

The previous sections show that LBC tends to crash into the leading vehicle in heavy rain or in unusual lighting conditions, caused either by the time of day or by entering a tunnel. The highest infraction scores are obtained for adverse weather conditions and challenging landscapes such as traffic lights, tunnels, and

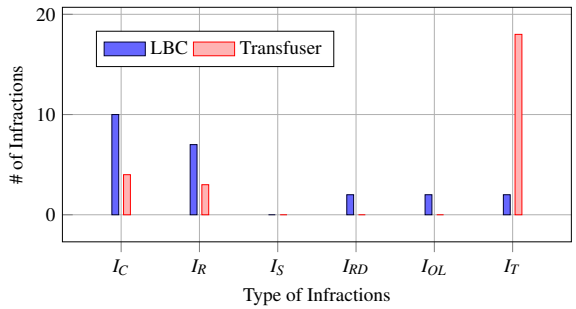



Figure 9.8: Infractions caused by the LBC and transfuser controllers.

roundabouts. By identifying those main reasons for failures, several suggestions can be made for improving LBC. First, adding another sensor modality could make the perception more robust to rain or darkness. For example, lidars do not require sunlight and are less susceptible to rain. Second, the robustness of the camera-based perception could be improved by training the controller with more images taken in the adverse conditions of rain and at dusk.

## 9.7 Conclusions and Discussion

In this chapter, we introduced ANTI-CARLA, an adversarial testing framework for the CARLA simulator that allows to automatically generate test cases that fail an arbitrary AV. Testing frameworks available in the literature are either tailor-made for a specific use case or built on proprietary simulators. In contrast, ANTI-CARLA is an open-source extension to an open-source simulator. The framework integrates a scenario description language for modeling test scenarios in terms of the system's operating conditions, sensor, and actuator faults. A specification file is used to specify and select the test conditions, infraction metrics, and samplers required for generating the test cases. The SDL is driven by an adversarial sampler that searches across the specified operating conditions. We used this framework to test the popular Learning By Cheating controller. Then, we derived recommendations for improving LBC. We also used ANTI-CARLA to compare the performance of LBC to the transfuser approach, allowing us to identify the weaknesses of each controller. While LBC achieved an accuracy of 100% across several scenarios in the CARLA challenge, the proposed framework automatically found various test cases in which the controller failed.

There are some limitations of ANTI-CARLA. First, only individual, static scenes can be sampled. A temporal sequence of scenes leading up to each failure case is unavailable. Temporal and dynamic sampling should be explored and added to the framework. Second, the framework currently does not include a coverage metric, which is essential in evaluating the test cases and making conclusions about a controller's capabilities.

Third, the samplers could also be further improved. One potential direction is to use reinforcement learning to learn what changes in the operating conditions led to failures and then generate the sequence of scenes that resulted in the system's failure. Fourth, sensor faults as a testing parameter and the remaining towns available in CARLA will be added to the framework to allow for adversarial testing of any AV system in any environment.

# Chapter 10

# Conclusions and Future Work

## 10.1 Conclusions

Assuring the safety of CPS has always been a complex problem, which has been exacerbated by the introduction of LECs. To address several problems of safety assurance, this dissertation presents an overarching "dynamic safety assurance" framework with the goal of performing through life safety assurance of autonomous systems. The proposed framework is a synergy of design-time safety assurance and runtime safety assurance approaches. The overall idea of the proposed framework is, to begin with, the design-time assurance models and include a range of monitors that considers runtime information about the system's operating condition and operational state (including information on system faults) to detect violations in the design-time assurance assumptions. Then, once the violation is detected, dynamically select a suitable action to mitigate the risk introduced to the system because of the assumption violation. In implementing this framework, this dissertation makes several important contributions to the safety assurance field, which is reiterated in this section.

First, this dissertation addresses the problems in developing and evaluating an assurance case. To remind, an assurance case is increasingly being integrated into safety standards [40, 41] establishing the minimum requirements for safety certification. However, the ever-increasing complexity of these systems has made the certification process complex, labor-intensive, and time-consuming. To address this, a workflow that automates the development and evaluation of an assurance case is presented in Chapter 4. The workflow applies a divide-and-conquer strategy using assurance case fragments called "patterns". The overall idea of the workflow is to select a set of patterns (with each pattern assuring one part of the system) and combine them to develop a comprehensive assurance argument for the system. To realize this idea, the workflow provides an array of functions, which consume the available artifacts of the system to automatically select the required patterns and synthesize an assurance case. Finally, an evaluation function automatically evaluates the assurance case for coverage and generates a report, which can be utilized for future iterations.

Second, this dissertation presented a workflow for detecting the OOD data problem resulting from the violations in the closed-world implicit assumptions involved in training the LEC (Chapter 5). The detector is specially targeted to identify OOD data in images, which is usually complex because of the computational resource and time needed to process the image pixels. To handle this complexity, the dimensionality of the images is first reduced to a lower-dimensional latent using a $\beta$-variational autoencoder detector on which

the detection is performed. The detector is trained using the concept of disentanglement to generate a latent space where each latent representation encodes information about a single image feature. The disentangled representations allow detection using lesser computational resources. It also allows one to map the image features and the representation required for identifying the exact factor (image feature) causing the problem.

Third, a framework called ReSonAte (Runtime Safety Evaluation in Autonomous Systems) is presented in Chapter 6 to dynamically assess the system's risk caused by the OOD data problem. The framework performs proactive assessment by first utilizing the design-time hazard analysis information to build a causal risk causation model called a bow-tie diagram. The model describes potential threats, and hazards to the system, and allows computing the probabilities of how these hazards may result in a consequence. Next, the threat probabilities are combined with information about the system's current state derived from system monitors (e.g., anomaly detectors, assurance monitors) and the operating environment (e.g., weather, traffic) to estimate the dynamic hazard rates of the system during operation.

Fourth, two extensions to the conventional simplex strategy are presented in Chapter 7 and Chapter 8 to adaptively select an optimal action to mitigate the system's risk caused by the OOD data problem. The first extension is, called the "weighted simplex strategy" (WSS), performs a weighted blending of the two controllers, and the second approach is, called the "dynamic simplex strategy" (DSS), performs a control arbitration among the two controllers (like the conventional simplex architecture). The goal with these extensions is to (a) reduce the conservatism of the simplex decision logic and (b) perform two-way switching between the two controllers (which most of the existing decision logic cannot do). The approaches are modeled as a sequential decision-making problem and formulated as a Markov (or semi-Markov) Decision Process. While the WSS approach learned an offline policy using reinforcement learning, the DSS approach performs the policy online using the Monte Carlo tree search heuristic. Finally, these approaches use a two-objective reward function focusing on the system's performance and safety objectives to reduce conservatism.

Finally, the ANTI-CARLA framework is presented in Chapter 9 to automatically generate the data required for developing the dynamic assurance components. The framework is built on the CARLA simulator (Chapter 9), which is a widely used simulator for autonomous driving in academia and industry [5, 417, 92]. The framework provides a scenario description language for describing the vehicle's operating conditions and a simple interface for specifying the conditions. The language is supported by a set of conventional and adversarial samplers to sample different values for the operating parameters (e.g., weather parameters like precipitation and time of day) from their respective distributions. The adversarial samplers use previous simulation results to guide the sampling process towards scenes where the system is expected to fail. Together, these samplers can automatically generate the data needed to develop the components of the proposed dynamic safety assurance framework.

## 10.2 Future Work

Today, CPSs have become an integral part of our everyday lives, and their prominence is only rising with the emergence of smart cities. Especially autonomous CPSs are seen as the prime factor that will accelerate this emergence. Artificial Intelligence and Machine Learning have been the most influential technology driving the transformation from CPS to autonomous CPS. While these technologies have allowed CPS to perform several tasks autonomously, wide-scale societal acceptance of these autonomous systems hinges on their ability to demonstrate safe operations. Unfortunately, there have been several accidents involving these systems recently, which have shown they can fail. While a stringent certification process exists and a variety of safety standards to assure the safety of these systems, they have struggled to keep pace with rapid technological advancement. To keep up with the certification process, the safety teams are following "a culture of paper safety at the expense of real safety" [48], which is not helping the cause. Therefore, safely integrating these systems into our society will require a new class of safety assurance processes that is vigorous and can incorporate these modern technologies.

This dissertation has presented the dynamic safety assurance framework with several design-time and runtime assurance components for continuous safety assurance of autonomous systems. However, significant work is yet to be accomplished to get this research into real-world systems. While each chapter in this dissertation presents potential future directions, an emphasis is made on the most prominent directions for the safety assurance of autonomous systems.

First, to get wide adoption of the dynamic assurance concept into real-world systems, a meaningful change must happen in how the assurance process is performed currently. The process is considered a one-time activity performed at the end of the system's development. In contrast, the overarching goal, as discussed by several authors [55, 48] and motivated in this work, should be to make the process a "through life safety assurance" activity, such that the design-time safety reasoning evolves based on the information from the runtime monitors. While this work has provided several design-time and runtime components for achieving this goal, these components are not tightly coupled. This means allowing (a) data flow from runtime monitors to complete/update the design-time assurance cases and (b) continually update the design-time safety reasoning and continually train the runtime components based on the system's latest experiences. Allowing this data flow would support the purpose of continuous life safety assurance. While there have been some recent efforts [48, 71] towards this goal, they are only applied to offline use cases.

Second, while this work presented several implementations for the components of the proposed framework, their development did not follow a standard process listed in current standards (e.g., ISO 26262 [41]), required for wide industry acceptance. Third, the current standards, such as ISO 26262, are non-prescriptive, which

has undoubtedly helped the assurance case development process, as pointed out by several authors [169, 48]. However, a repercussion of the non-prescriptive nature is that there is no standard procedure and metrics for evaluating the assurance case. This has resulted in several independent metrics being used to evaluate the assurance case's different aspects (coverage and confidence). Therefore, vigorous evaluation may require an array of evaluation metrics in addition to the coverage metrics presented in Chapter 4.

Fourth, in building the assurance case, this work assumed the LEC as one other conventional component and used the available patterns to argue about its safety. However, these general patterns cannot capture (a) how the assurance of the LEC impacts the overall assurance of the system and (b) the implicit assumptions involved in training the LEC. There is ongoing work in designing exclusive patterns for LECs [421], which is essential for future work. Finally, while this work has developed an automated data generation framework in simulation, it has not presented a metric to measure the coverage of the generated data. This metric is essential in developing and testing the dynamic assurance components.

While this dissertation has not yet achieved these goals, it provides several key components which can be further explored and tightly integrated as the field continues this pursuit.

List of Publications

Journal Articles

- **Shreyas Ramakrishna**, Zahra Rahiminasab, Gabor Karsai, Arvind Easwaran, and Abhishek Dubey. "Efficient Out-of-Distribution Detection Using Latent Space of $\beta$-VAE for Cyber-Physical Systems." In Transactions on Cyber-Physical Systems (TCPS), 2021.

- **Shreyas Ramakrishna**, Charles Hartsell, Matthew P. Burruss, Gabor Karsai, and Abhishek Dubey. "Dynamic-weighted simplex strategy for learning enabled cyber physical systems." Journal of systems architecture 111 (2020): 101760.

Conference and Workshop Papers

- **Shreyas Ramakrishna**, Hyunjee Jin, Abhishek Dubey, and Arun Ramamurthy. "Automating Pattern Selection for Assurance Case Development of Cyber-Physical Systems", Accepted for publication in the 41st international conference on Computer Safety, Reliability and Security (SAFECOMP), 2022.

- **Shreyas Ramakrishna**[1], Baiting Luo[1], Christopher Kuhn, Gabor Karsai, and Abhishek Dubey. "ANTI-CARLA: An Adversarial Testing Framework for Autonomous Vehicles in CARLA" Accepted for publication in the 2022 IEEE Intelligent Transportation Systems Society Conference Management System (ITSC).

- **Shreyas Ramakrishna**, Baiting Luo, Christopher Kuhn, Ayan Mukhopadhyay, Gabor Karsai, and Abhishek Dubey. "Dynamic Simplex Strategy for Autonomous Cyber-Physical Systems" July 2022, Awaiting Submission.

- **Shreyas Ramakrishna**, Baiting Luo, Yogesh Barve, Gabor Karsai, and Abhishek Dubey. "Risk-Aware Scene Sampling for Dynamic Assurance of Autonomous Systems." In IEEE International Conference on Assured Autonomy (ICAA), IEEE, 2021.

- Charles Hartsell[1], **Shreyas Ramakrishna**[1], Abhishek Dubey, Daniel Stojcsics, Nagabhushan Mahadevan, and Gabor Karsai. "ReSonAte: A Runtime Risk Assessment Framework for Autonomous Systems." In Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 118-129. IEEE, 2021.

[1]These authors have contributed equally

- Matthew Burruss, **Shreyas Ramakrishna**, and Abhishek Dubey. "Deep-RBF Networks for Anomaly Detection in Automotive Cyber-Physical Systems." In IEEE International Conference on Smart Computing (SMARTCOMP), pp. 55-60. IEEE, 2021.

- Vijaya Kumar Sundar[1], **Shreyas Ramakrishna**[1], Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. "Out-of-distribution detection in multi-label datasets using latent space of $\beta$-VAE." In IEEE Security and Privacy Workshops (SPW), pp. 250-255. IEEE, 2020.

- **Shreyas Ramakrishna**, Charles Hartsell, Abhishek Dubey, Partha Pal, and Gabor Karsai. "A Methodology for Automating Assurance Case Generation." In Tools and Methods of Competitive Engineering (TMCE), pp. 265-278. 2020.

- **Shreyas Ramakrishna**, Abhishek Dubey, Matthew P. Burruss, Charles Hartsell, Nagabhushan Mahadevan, Saideep Nannapaneni, Aron Laszka, and Gabor Karsai. "Augmenting learning components for safety in resource constrained autonomous robots." In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 108-117. IEEE, 2019.

- Charles Hartsell, Nagabhushan Mahadevan, **Shreyas Ramakrishna**, Abhishek Dubey, Theodore Bapty, Taylor Johnson, Xenofon Koutsoukos, Janos Sztipanovits, and Gabor Karsai. "Model-based design for cps with learning-enabled components." In Proceedings of the Workshop on Design Automation for CPS and IoT, pp. 1-9. 2019.

- Charles Hartsell, Nagabhushan Mahadevan, **Shreyas Ramakrishna**, Abhishek Dubey, Theodore Bapty, Taylor Johnson, Xenofon Koutsoukos, Janos Sztipanovits, and Gabor Karsai. "CPS Design with Learning-Enabled Components: A Case Study." In Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP'19), pp. 57-63. 2019.

Poster and Work in Progress Presentations

- **Shreyas Ramakrishna**, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. "Efficient Multi-Class Out-of-Distribution Reasoning for Perception Based Networks: Work-in-Progress." In 2020 International Conference on Embedded Software (EMSOFT), pp. 40-42. IEEE, 2020.

- Matthew P. Burruss, **Shreyas Ramakrishna**, Gabor Karsai, and Abhishek Dubey. "DeepNNcar: A testbed for deploying and testing middleware frameworks for autonomous robots." In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), pp. 87-88. IEEE, 2019.

- Charles Hartsell, Nagabhushan Mahadevan, **Shreyas Ramakrishna**, Abhishek Dubey, Theodore Bapty, and Gabor Karsai. “A CPS toolchain for learning-based systems: demo abstract.” In Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, pp. 342-343. 2019.

Bibliography


[1] [Online] Federal Aviation Administration. Advisory circular (AC120-92A), 2021. URL https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC%20120-92A.pdf.

[2] Ashok N Srivastava and Johann Schumann. Software health management: a necessity for safety critical systems. *Innovations in Systems and Software Engineering*, 9(4):219–233, 2013.

[3] Sergey Levine. Deep reinforcement learning. http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_1_introduction.pdf, 2020.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[6] Edward A Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 363–369. IEEE, 2008.

[7] Vasso Reppa, Marios M Polycarpou, and Christos G Panayiotou. Distributed sensor fault diagnosis for a network of interconnected cyberphysical systems. *IEEE Transactions on Control of Network Systems*, 2(1):11–23, 2014.

[8] Pieter J Mosterman and Justyna Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(1):5–16, 2016.

[9] John Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef. From embedded to cyber-physical systems: Challenges and future directions. In *Collaborative design for embedded systems*, pages 293–303. Springer, 2014.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Ignacio Carlucho, Mariano De Paula, Sen Wang, Yvan Petillot, and Gerardo G Acosta. Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning. *Robotics and Autonomous Systems*, 107:71–86, 2018.

[13] Qilei Zhang, Jinying Lin, Qixin Sha, Bo He, and Guangliang Li. Deep interactive reinforcement learning for path following of autonomous underwater vehicle. *IEEE Access*, 8:24258–24268, 2020.

[14] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[15] Pin-Chu Yang, Kazuma Sasaki, Kanata Suzuki, Kei Kase, Shigeki Sugano, and Tetsuya Ogata. Repeatable folding task by humanoid robot worker using deep learning. *IEEE Robotics and Automation Letters*, 2(2):397–403, 2016.

[16] Mesay Belete Bejiga, Abdallah Zeggada, Abdelhamid Nouffidj, and Farid Melgani. A convolutional neural network approach for assisting avalanche search and rescue operations with uav imagery. *Remote Sensing*, 9(2):100, 2017.

[17] Carlos Sampedro, Alejandro Rodriguez-Ramos, Hriday Bavle, Adrian Carrio, Paloma de la Puente, and Pascual Campoy. A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques. *Journal of Intelligent & Robotic Systems*, 95(2):601–627, 2019.

[18] Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[19] Raz Lin, Eliyahu Khalastchi, and Gal A Kaminka. Detecting anomalies in unmanned vehicles using the mahalanobis distance. In *2010 IEEE international conference on robotics and automation*, pages 3038–3044. IEEE, 2010.

[20] Zhipeng Deng, Hao Sun, Shilin Zhou, Juanping Zhao, Lin Lei, and Huanxin Zou. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS journal of photogrammetry and remote sensing*, 145:3–22, 2018.

[21] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[22] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[23] Bill Vlasic and Neal E Boudette. 'self-driving tesla was involved in fatal crash,'us says. *New York Times*, 302016, 2016.

[24] Puneet Kohli and Anjali Chadha. Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash. In *Future of Information and Communication Conference*, pages 261–279. Springer, 2019.

[25] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28:2503–2511, 2015.

[26] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[27] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[28] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[29] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Reachable set computation and safety verification for neural networks with ReLU activations. *arXiv preprint arXiv:1712.08163*, 2017.

[30] H Dezfuli, A Benjamin, C Everett, C Smith, M Stamatelatos, and R Youngblood. Nasa system safety handbook, volume 1, system safety framework and concepts for implementation. *National Aeronautics and Space Administration NASA Headquarters, Washington, DC*, 20546, 2011.

[31] [Online] Federal Aviation Administration. FAA System Safety Handbook, 2021.

[32] Robert G Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 winter simulation conference*, pages 166–183. IEEE, 2010.

[33] Edmund M Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[34] Siddhartha R Dalal, Ashish Jain, Nachimuthu Karunanithi, JM Leaton, Christopher M Lott, Gardner C Patton, and Bruce M Horowitz. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*, pages 285–294, 1999.

[35] Mark Utting and Bruno Legeard. *Practical model-based testing: a tools approach*. Elsevier, 2010.

[36] Nicholas J Bahr. *System safety engineering and risk assessment: a practical approach*. CRC press, 2014.

[37] [Online] Federal Aviation Administration. Risk management handbook (faa-h-8083-2), 2019. URL https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/FAA-H-8083-2.pdf.

[38] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *Safety and Reliability*, volume 20, pages 34–42. Taylor & Francis, 2000.

[39] Patrick J Graydon, John C Knight, and Elisabeth A Strunk. Assurance based development of critical systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 347–357. IEEE, 2007.

[40] Thomas K Ferrell and Uma D Ferrell. Rtca do-178b/eurocae ed-12b. In *Digital Avionics Handbook*, pages 467–478. CRC Press, 2000.

[41] [Online] International Organization for Standardization. Iso 26262, 2021. URL https://www.iso.org/standard/43464.html.

[42] ISO Iso. Pas 21448-road vehicles-safety of the intended functionality. *International Organization for Standardization*, 2019.

[43] Junko Yoshida. Ul takes autonomy standards plunge. *EE Times*, 16, 2019.

[44] Peter B Ladkin. The ariane 5 accident: A programming problem?, 1998.

[45] Mishap Investigation Board. Mars climate orbiter mishap investigation board phase i report november 10, 1999, 1999.

[46] Thomas Young, James Arnold, Thomas Brackey, Michael Carr, Douglas Dwoyer, Ronald Fogleman, Ralph Jacobson, Herbert Kottler, Peter Lyman, and Joanne Maguire. Mars program independent assessment team report. *NASA STI/Recon Technical Report N*, page 32462, 2000.

[47] Nancy G Leveson. The role of software in recent aerospace accidents. In *Proceedings of the 19th International System Safety Conference, System Safety Society: Unionville, VA*, 2001.

[48] Ewen Denney, Ganesh Pai, and Ibrahim Habli. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 587–590. IEEE, 2015.

[49] Charles Haddon-Cave. *The Nimrod Review: an independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006, report*, volume 1025. DERECHO INTERNACIONAL, 2009.

[50] John Alexander McDermid, Yan Jia, and Ibrahim Habli. Towards a framework for safety assurance of autonomous systems. In *Artificial Intelligence Safety 2019*, pages 1–7. CEUR Workshop Proceedings, 2019.

[51] Tim Pearce, Felix Leibfried, Alexandra Brintrup, Mohamed Zaki, and Andy Neely. Uncertainty in neural networks: Approximately bayesian ensembling. *arXiv preprint arXiv:1810.05546*, 2018.

[52] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[53] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069, 2017.

[54] Sohag Kabir, Ioannis Sorokos, Koorosh Aslansefat, Yiannis Papadopoulos, Youcef Gheraibia, Jan Reich, Merve Saimler, and Ran Wei. A runtime safety analysis concept for open adaptive systems. In *International Symposium on Model-Based Safety and Assessment*, pages 332–346. Springer, 2019.

[55] Nancy G Leveson. The use of safety cases in certification and regulation. *Massachusetts Institute of Technology*, 2011.

[56] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, 2012.

[57] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[58] Ahmed Badreddine and Nahla Ben Amor. A new approach to construct optimal bow tie diagrams for risk analysis. In Nicolás García-Pedrajas, Francisco Herrera, Colin Fyfe, José Manuel Benítez, and

Moonis Ali, editors, *Trends in Applied Intelligent Systems*, pages 595–604, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13025-0.

[59] Ahmed Badreddine and Nahla Ben Amor. A bayesian approach to construct bow tie diagrams for risk evaluation. *Process Safety and Environmental Protection*, 91(3):159–171, 2013.

[60] Mahsa Teimourikia, Mariagrazia Fugini, and Claudia Raibulet. Run-time security and safety management in adaptive smart work environments. In *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 256–261. IEEE, 2017.

[61] D Seto, BH Krogh, L Sha, and A Chutinan. The simplex architecture for safe on-line control system upgrades. In *Proceedings of the American Control Conference*, volume 6, pages 3504–3508. AMERICAN AUTOMATIC CONTROL COUNCIL, 1998.

[62] D Seto, Bruce H Krogh, Lui Sha, and A Chutinan. Dynamic control system upgrade using the simplex architecture. *IEEE Control Systems*, 18(4):72–80, 1998.

[63] Danbing Seto and Lui Sha. A case study on analytical analysis of the inverted pendulum real-time control system. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1999.

[64] John Rushby. Just-in-time certification. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pages 15–24. IEEE, 2007.

[65] John Rushby. Runtime certification. In *International Workshop on Runtime Verification*, pages 21–35. Springer, 2008.

[66] Daniel Schneider and Mario Trapp. A safety engineering framework for open adaptive systems. In *2011 IEEE Fifth International Conference on Self-Adaptive and Self-Organizing Systems*, pages 89–98. IEEE, 2011.

[67] Daniel Schneider, Martin Becker, and Mario Trapp. Approaching runtime trust assurance in open adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 196–201, 2011.

[68] Abhishek Dubey, Gabor Karsai, and Nagabhushan Mahadevan. Model-based software health management for real-time systems. In *2011 Aerospace Conference*, pages 1–18. IEEE, 2011.

[69] Nagabhushan Mahadevan, Abhishek Dubey, and Gabor Karsai. Architecting health management into software component assemblies: Lessons learned from the arinc-653 component mode. In *2012*

*IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 79–86. IEEE, 2012.

[70] Ashok N Srivastava and Johann Schumann. The case for software health management. In *2011 IEEE Fourth International Conference on Space Mission Challenges for Information Technology*, pages 3–9. IEEE, 2011.

[71] Erfan Asaadi, Ewen Denney, Jonathan Menzies, Ganesh J Pai, and Dimo Petroff. Dynamic assurance cases: A pathway to trusted autonomy. *Computer*, 53(12):35–46, 2020.

[72] Shunsuke Shida, Atsushi Uchida, Masaki Ishii, Masahiro Ide, and Kimio Kuramitsu. Assure-it: a runtime synchronization tool of assurance cases. In *Safecomp 2013 fastabstract*, page NC, 2013.

[73] Mo Chen, Qie Hu, Casey Mackin, Jaime F Fisac, and Claire J Tomlin. Safe platooning of unmanned aerial vehicles via reachability. In *2015 54th IEEE conference on decision and control (CDC)*, pages 4695–4701. IEEE, 2015.

[74] Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, and Claire J Tomlin. An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1758–1765. IEEE, 2019.

[75] Claudia Priesterjahn, Christian Heinzemann, Wilhelm Schäfer, and Matthias Tichy. Runtime safety analysis for safe reconfiguration. In *IEEE 10th International Conference on Industrial Informatics*, pages 1092–1097. IEEE, 2012.

[76] Richard Hawkins, Colin Paterson, Chiara Picardi, Yan Jia, Radu Calinescu, and Ibrahim Habli. Guidance on the assurance of machine learning in autonomous systems (amlas). *arXiv preprint arXiv:2102.01564*, 2021.

[77] Chiara Picardi, Colin Paterson, Richard David Hawkins, Radu Calinescu, and Ibrahim Habli. Assurance argument patterns and processes for machine learning in safety-related systems. In *Proceedings of the Workshop on Artificial Intelligence Safety (SafeAI 2020)*, pages 23–30. CEUR Workshop Proceedings, 2020.

[78] Erfan Asaadi, Ewen Denney, and Ganesh Pai. Quantifying assurance in learning-enabled systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 270–286. Springer, 2020.

[79] Sunil Nair, Jose Luis de la Vara, Mehrdad Sabetzadeh, and Davide Falessi. Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. *Information and Software Technology*, 60:1–15, 2015.

[80] Shreyas Ramakrishna, Abhishek Dubey, Matthew P Burruss, Charles Hartsell, Nagabhushan Mahadevan, Saideep Nannapaneni, Aron Laszka, and Gabor Karsai. Augmenting learning components for safety in resource constrained autonomous robots. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 108–117. IEEE, 2019.

[81] [Online] F1TENTH Foundation. F1Tenth autonomous racing competition, 2022. URL http://f1tenth.org/.

[82] Blue Robotics. Bluerov2. *Datasheet, June*, 2016.

[83] Rob Palin, David Ward, Ibrahim Habli, and Roger Rivett. Iso 26262 safety cases: Compliance and assurance. In *6th IET International Conference on System Safety 2011*, pages 1–6, 2011. doi: 10.1049/cp.2011.0251.

[84] European Organisation for the Safety of Air Navigation. Safety case development manual, ver 2.2, 2006.

[85] FDA, [Online]. Introduction of assurance case method and its application in regulatory science, 2019. URL https://www.fda.gov/media/125182/download.

[86] [Online] Federal Aviation Administration. Airworthiness certification, 2022. URL https://www.faa.gov/aircraft/air_cert/airworthiness_certification/.

[87] DARPA. Automated rapid certification of software (arcos), 2022. URL https://www.darpa.mil/program/automated-rapid-certification-of-software.

[88] [Online] AMASS consortium. Amass assurance and certification of cps, 2019. URL https://www.amass-ecsel.eu/.

[89] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[90] Nick Drummond and Rob Shearer. The open world assumption. In *eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web*, volume 15, 2006.

[91] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. In *Robotics: Science and Systems*, 2017.

[92] Feiyang Cai and Xenofon Koutsoukos. Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems*

*(ICCPS)*, pages 174–183, Los Alamitos, CA, USA, apr 2020. IEEE Computer Society. doi: 10.1109/ICCPS48487.2020.00024. URL https://doi.ieeecomputersociety.org/10.1109/ICCPS48487.2020.00024.

[93] Vijaya Sundar, Shreyas Ramakrishna, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. Out-of-distribution detection in multi-label datasets using latent space of $\beta$-vae. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 250–255. IEEE, 2020.

[94] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1), 2015.

[95] Taylor Denouden, Rick Salay, Krzysztof Czarnecki, Vahdat Abdelzad, Buu Phan, and Sachin Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *arXiv:1812.02765*, 2018.

[96] Aleksei Vasilev, Vladimir Golkov, Marc Meissner, Ilona Lipp, Eleonora Sgarlata, Valentina Tomassini, Derek K. Jones, and Daniel Cremers. q-space novelty detection with variational autoencoders. In Elisenda Bonet-Carne, Jana Hutter, Marco Palombo, Marco Pizzolato, Farshid Sepehrband, and Fan Zhang, editors, *Computational Diffusion MRI*, pages 113–124, Cham, 2020. Springer International Publishing. ISBN 978-3-030-52893-5.

[97] Christian Delvosalle, Cécile Fievez, Aurore Pipart, and Bruno Debray. Aramis project: A comprehensive methodology for the identification of reference accident scenarios in process industries. *Journal of Hazardous Materials*, 130(3):200–219, 2006.

[98] Stanley Bak, Karthik Manamcheri, Sayan Mitra, and Marco Caccamo. Sandboxing controllers for cyber-physical systems. In *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pages 3–12. IEEE, 2011.

[99] Danbing Seto, Enrique Ferreira, and Theodore F Marz. Case study: Development of a baseline controller for automatic landing of an f-16 aircraft using linear matrix inequalities (lmis). Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2000.

[100] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, and Shawn Keshmiri. A simplex architecture for intelligent and safe unmanned aerial vehicles. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 69–75. IEEE, 2016.

[101] Tanya L Crenshaw, Elsa Gunter, Craig L Robinson, Lui Sha, and PR Kumar. The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 400–412. IEEE, 2007.

[102] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. S3a: secure system simplex architecture for enhanced security of cyber-physical systems. *arXiv preprint arXiv:1202.5722*, 2012.

[103] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[104] Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J Kochenderfer. Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *IEEE transactions on intelligent vehicles*, 5(2):294–305, 2019.

[105] Geoffrey Pettet, Ayan Mukhopadhyay, and Abhishek Dubey. Decision making in non-stationary environments with policy-augmented monte carlo tree search. *arXiv preprint arXiv:2202.13003*, 2022.

[106] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J Kochenderfer, and Abhishek Dubey. Hierarchical planning for resource allocation in emergency response systems. In *International Conference on Cyber-Physical Systems*, pages 155–166, 2021.

[107] Ankush Desai, Shromona Ghosh, Sanjit A Seshia, Natarajan Shankar, and Ashish Tiwari. Soter: a runtime assurance framework for programming safe robotics systems. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 138–150. IEEE, 2019.

[108] Taylor T Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):1–27, 2016.

[109] Dung T Phan, Radu Grosu, Nils Jansen, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. Neural simplex architecture. In *NASA Formal Methods Symposium*, pages 97–114. Springer, 2020.

[110] Stanley Bak, Taylor T Johnson, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 138–148. IEEE, 2014.

[111] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv:1711.03938*, 2017.

[112] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[113] Shreyas Ramakrishna, Hyunjee Jin, Abhishek Dubey, and Arun Ramamurthy. Automating pattern selection for assurance case development of cyber-physical systems. In *Accepted at SAFECOMP 2022, Pending Publication*. Springer, 2022.

[114] Shreyas Ramakrishna, Charles Hartsell, Abhishek Dubey, Partha Pratim Pal, and Gabor Karsai. A methodology for automating assurance case generation. *Tools and Methods of Competitive Engineering*, pages 265–278, 2020.

[115] Shreyas Ramakrishna, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. Efficient multi-class out-of-distribution reasoning for perception based networks: Work-in-progress. In *2020 International Conference on Embedded Software (EMSOFT)*, pages 40–42. IEEE, 2020.

[116] Shreyas Ramakrishna, Zahra Rahiminasab, Gabor Karsai, Arvind Easwaran, and Abhishek Dubey. Efficient out-of-distribution detection using latent space of $\beta$-vae for cyber-physical systems. *Transactions on Cyber Physical Systems (TCPS)*, 2021.

[117] C. Hartsell, S. Ramakrishna, A. Dubey, D. Stojcsics, N. Mahadevan, and G. Karsai. Resonate: A runtime risk assessment framework for autonomous systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2021.

[118] Shreyas Ramakrishna, Baiting Luo, Christopher Kuhn, Gabor Karsai, and Abhishek Dubey. Anti-carla: An adversarial testing framework for autonomous vehicles in carla. In *Accepted at ITSC 2022, Pending Publication*. IEEE, 2022.

[119] Shreyas Ramakrishna, Baiting Luo, Yogesh Barve, Gabor Karsai, and Abhishek Dubey. Risk-aware scene sampling for dynamic assurance of autonomous systems. In *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pages 107–116, 2022. doi: 10.1109/ICAA52185.2022.00022.

[120] Shreyas Ramakrishna, Charles Harstell, Matthew P Burruss, Gabor Karsai, and Abhishek Dubey. Dynamic-weighted simplex strategy for learning enabled cyber physical systems. *Journal of Systems Architecture*, page 101760, 2020.

[121] Dimitra Giannakopoulou, Corina S Pasareanu, and Jamieson M Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *Proceedings. 26th International Conference on Software Engineering*, pages 211–220. IEEE, 2004.

[122] Doron Peled, Amir Pnueli, and Lenore Zuck. From falsification to verification. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 292–304. Springer, 2001.

[123] Antti Valmari. The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer, 1996.

[124] Claire J Tomlin, Ian Mitchell, Alexandre M Bayen, and Meeko Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

[125] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 379–395, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22110-1.

[126] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[127] Zhi Han and Bruce H Krogh. Reachability analysis of large-scale affine systems using low-dimensional polytopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 287–301. Springer, 2006.

[128] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*, pages 670–686. Springer, 2019.

[129] Matthias Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[130] Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.

[131] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, pages 171–178, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-71070-7.

[132] Stewart Schlesinger. Terminology for model credibility. *Simulation*, 32(3):103–104, 1979.

[133] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. Perceptions on the state of the art in verification and validation in cyber-physical systems. *IEEE Systems Journal*, 11(4):2614–2627, 2015.

[134] Xi Zheng and Christine Julien. Verification and validation in cyber physical systems: Research challenges and a way forward. In *2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 15–18. IEEE, 2015.

[135] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[136] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[137] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.

[138] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.

[139] [Online] SRI Lab. Eran: Eth robustness analyzer for neural networks, 2022. URL https://github.com/eth-sri/eran.

[140] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *International Static Analysis Symposium*, pages 296–319. Springer, 2019.

[141] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. *Advances in neural information processing systems*, 29:2613–2621, 2016.

[142] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, page 3, 2018.

[143] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[144] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1599–1614, 2018.

[145] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[146] Xin Zhou, Xiaodong Gou, Tingting Huang, and Shunkun Yang. Review on testing of cyber physical systems: Methods and testbeds. *IEEE Access*, 6:52179–52194, 2018.

[147] Larry Apfelbaum and John Doyle. Model based testing. In *Software quality week conference*, pages 296–300, 1997.

[148] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*, pages 432–442. Springer, 2019.

[149] Ayman Faza, Sahra Sedigh, and Bruce McMillin. Integrated cyber-physical fault injection for reliability analysis of the smart grid. In *International Conference on Computer Safety, Reliability, and Security*, pages 277–290. Springer, 2010.

[150] Matthias Woehrle, Kai Lampka, and Lothar Thiele. Segmented state space traversal for conformance testing of cyber-physical systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 193–208. Springer, 2011.

[151] Nathan P Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Citeseer, 2004.

[152] Daniel Fremont, Xiangyu Yue, Tommaso Dreossi, Shromona Ghosh, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: Language-based scene generation. *arXiv preprint arXiv:1809.09310*, 2018.

[153] [Online] foretellix. Open m-sdl, 2021. URL https://www.foretellix.com/open-language/.

[154] Wen-Hou Ma and Huei Peng. A worst-case evaluation method for dynamic systems. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, 121:191–199, 1999.

[155] Kesav Viswanadha, Edward Kim, Francis Indaheng, Daniel J Fremont, and Sanjit A Seshia. Parallel and multi-objective falsification with scenic and verifai. In *International Conference on Runtime Verification*. Springer, 2021.

[156] Diomidis H Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press, 2003.

[157] Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. Fault tree analysis, methods, and applications: a review. *IEEE transactions on reliability*, 34(3):194–203, 1985.

[158] Reece A Clothier, Brendan P Williams, and Neale L Fulton. Structuring the safety case for unmanned aircraft system operations in non-segregated airspace. *Safety science*, 79:213–228, 2015.

[159] Brendan P Williams, Reece Clothier, Neale Fulton, Sandra Johnson, Xunguo Lin, and Kelly Cox. Building the safety case for uas operations in support of natural disaster response. In *14th AIAA Aviation Technology, Integration, and Operations Conference*, page 2286, 2014.

[160] Ewen Denney, Ganesh Pai, and Iain Whiteside. Modeling the safety architecture of uas flight operations. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 162–178, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66266-4.

[161] Ewen Denney, Ganesh Pai, and Iain Whiteside. The role of safety architectures in aviation safety cases. *Reliability Engineering & System Safety*, 191:106502, 2019.

[162] Nancy Leveson, Mirna Daouk, Nicolas Dulac, and Karen Marais. A systems theoretic approach to safety engineering. *Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Technology, Cambridge*, pages 16–17, 2003.

[163] Jens Rasmussen. Risk management in a dynamic society: a modelling problem. *Safety science*, 27(2-3): 183–213, 1997.

[164] Nancy G Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Transactions on Dependable and Secure computing*, 1(1):66–86, 2004.

[165] Nancy G Leveson, Polly Allen, and Margaret-Anne Storey. The analysis of a friendly fire accident using a systems model of accidents. In *Proceedings of the 20th International System Safety Conference*, pages 5–9. Citeseer, 2002.

[166] Nancy G Leveson. System safety engineering: Back to the future. *MIT Press*, 2002.

[167] Nancy G Leveson. Analyzing accidents and incidents (cast). *MIT Press*, 2012.

[168] Björn Johansson and Mattias Lindgren. A quick and dirty evaluation of resilience enhancing properties in safety critical systems. In *Third Symposium on Resilience Engineering*, pages 28–30. École des mines de Paris, 2008.

[169] Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. Assurance cases and prescriptive software safety certification: A comparative study. *Safety science*, 59:55–71, 2013.

[170] Charles W Kneupper. Teaching argument: An introduction to the toulmin model. *College Composition and Communication*, 29(3):237–241, 1978.

[171] Douglas N Walton and David N Walton. *Informal logic: A handbook for critical argument*. Cambridge University Press, 1989.

[172] Chris Edwards. Railway safety cases. In *Safety and Reliability of Software Based Systems*, pages 317–322. Springer, 1997.

[173] Lord W Douglas Cullen. The public inquiry into the piper alpha disaster. *Drilling Contractor;(United States)*, 49(4), 1993.

[174] RE Bloomfield, PG Bishop, C Jones, and PKD Froome. Ascad—adelard safety case development manual. *Adelard, 1998. ISBN 0-9533771-0*, 5, 1998.

[175] Tim Kelly. A systematic approach to safety case management. Technical report, SAE Technical Paper, 2004.

[176] Timothy Patrick Kelly. *Arguing safety: a systematic approach to managing safety cases*. PhD thesis, University of York York, UK, 1999.

[177] Richard Hawkins and Tim Kelly. A software safety argument pattern catalogue. *The University of York, York*, 30, 2013.

[178] Monika Szczygielska and Aleksander Jarzkebowicz. Assurance case patterns on-line catalogue. In *Advances in Dependability Engineering of Complex Systems*, pages 407–417. Springer, 2017.

[179] Janusz Górski, Aleksander Jarzḱebowicz, Jakub Miler, Michaĺ Witkowicz, Jakub Czyżnikiewicz, and Patryk Jar. Supporting assurance by evidence-based argument services. In *International Conference on Computer Safety, Reliability, and Security*, pages 417–426. Springer, 2012.

[180] David J Rinehart, John C Knight, and Jonathan Rowanhill. *Current practices in constructing and evaluating assurance cases with applications to aviation*. National Aeronautics and Space Administration, Langley Research Center, 2015.

[181] P Steele, K Collins, and J Knight. Access: A toolset for safety case creation and management. In *Proc. 29th Intl. Systems Safety Conf.(August 2011)*, 2011.

[182] LLP Adelard. Assurance and safety case environment (asce), 2011.

[183] Yutaka Matsuno. D-case editor: A typed assurance case editor. *University of Tokyo*, 2011.

[184] Matthew R Barry. Certware: A workbench for safety case production and analysis. In *2011 Aerospace conference*, pages 1–10. IEEE, 2011.

[185] Andrew Gacek, John Backes, Darren Cofer, Konrad Slind, and Mike Whalen. Resolute: an assurance case language for architecture models. *ACM SIGAda Ada Letters*, 34(3):19–28, 2014.

[186] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[187] Ewen Denney, Ganesh Pai, and Josef Pohl. Advocate: An assurance case automation toolset. In *International Conference on Computer Safety, Reliability, and Security*, pages 8–21. Springer, 2012.

[188] Ewen Denney and Ganesh Pai. A lightweight methodology for safety case assembly. In *International Conference on Computer Safety, Reliability, and Security*, pages 1–12. Springer, 2012.

[189] Richard Hawkins, Ibrahim Habli, Dimitris Kolovos, Richard Paige, and Tim Kelly. Weaving an assurance case from design: a model-based approach. In *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, pages 110–117. IEEE, 2015.

[190] John Rushby. *Modular certification*. Citeseer, 2002.

[191] Alex Wilson and Thierry Preyssler. Incremental certification and integrated modular avionics. *IEEE Aerospace and Electronic Systems Magazine*, 24(11):10–15, 2009.

[192] Tim P Kelly. Managing complex safety cases. In *Current Issues in Safety-Critical Systems*, pages 99–115. Springer, 2003.

[193] L Fenn, Richard D Hawkins, PJ Williams, Tim P Kelly, Michael G Banner, and Y Oakshott. The who, where, how, why and when of modular and incremental certification. In *2007 2nd Institution of Engineering and Technology International Conference on System Safety*, pages 135–140. IET, 2007.

[194] Patrick Graydon and Iain Bate. The nature and content of safety contracts: Challenges and suggestions for a way forward. In *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pages 135–144. IEEE, 2014.

[195] Tim Kelly. Using software architecture techniques to support the modular certification of safety-critical systems. In *ACM International Conference Proceeding Series*, volume 248, pages 53–65, 2007.

[196] Iain Bate and Tim Kelly. Architectural considerations in the certification of modular systems. *Reliability Engineering & System Safety*, 81(3):303–324, 2003.

[197] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming dr. frankenstein: Contract-based design for cyber-physical systems. *European journal of control*, 18(3):217–238, 2012.

[198] Sidharta Andalam, Daniel Jun Xian Ng, Arvind Easwaran, and Karthikeyan Thangamariappan. Clair: A contract-based framework for developing resilient cps architectures. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 33–41. IEEE, 2018.

[199] Dung Phan, Junxing Yang, Matthew Clark, Radu Grosu, John D Schierman, Scott A Smolka, and Scott D Stoller. A component-based simplex architecture for high-assurance cyber-physical systems. *arXiv preprint arXiv:1704.04759*, 2017.

[200] Alina Eqtami and Antoine Girard. A quantitative approach on assume-guarantee contracts for safety of interconnected systems. In *2019 18th European Control Conference (ECC)*, pages 536–541. IEEE, 2019.

[201] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[202] Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Davide Bresolin, Luca Geretti, and Tiziano Villa. A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. *Proceedings of the IEEE*, 103(11):2104–2132, 2015.

[203] Patricia Derler, Edward A Lee, Stavros Tripakis, and Martin Törngren. Cyber-physical system design contracts. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 109–118, 2013.

[204] Charles Haddon-Cave. *The Nimrod Review: an independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006, report*, volume 1025. DERECHO INTERNACIONAL, 2009.

[205] Yutaka Matsuno and Shuichiro Yamamoto. Toward dynamic assurance cases. In *JCKBSE*, pages 154–160, 2012.

[206] Daniel Schneider and Mario Trapp. Conditional safety certification of open adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(2):1–20, 2013.

[207] Tiago Amorim, Alejandra Ruiz, Christoph Dropmann, and Daniel Schneider. Multidirectional modular conditional safety certificates. In *International Conference on Computer Safety, Reliability, and Security*, pages 357–368. Springer, 2014.

[208] Xi Zheng, Christine Julien, Rodion Podorozhny, Franck Cassez, and Thierry Rakotoarivelo. Efficient and scalable runtime monitoring for cyber–physical system. *IEEE Systems Journal*, 12(2):1667–1678, 2016.

[209] Konstantine Arkoudas and Martin Rinard. Deductive runtime certification. *Electronic Notes in Theoretical Computer Science*, 113:45–63, 2005.

[210] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. *Engineering dependable software systems*, pages 141–175, 2013.

[211] Pierfrancesco Bellini, Riccardo Mattolini, and Paolo Nesi. Temporal logics for real-time system specification. *ACM Computing Surveys (CSUR)*, 32(1):12–42, 2000.

[212] Anayo K Akametalu, Jaime F Fisac, Jeremy H Gillula, Shahab Kaynama, Melanie N Zeilinger, and Claire J Tomlin. Reachability-based safe learning with gaussian processes. In *53rd IEEE Conference on Decision and Control*, pages 1424–1431. IEEE, 2014.

[213] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 313–329. Springer, 1998.

[214] Stefan Mitsch and André Platzer. Modelplex: Verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, 49(1):33–74, 2016.

[215] Kostas Margellos and John Lygeros. Hamilton–jacobi formulation for reach–avoid differential games. *IEEE Transactions on automatic control*, 56(8):1849–1861, 2011.

[216] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[217] Cesar Sanchez, Gerardo Schneider, Wolfgang Ahrendt, Ezio Bartocci, Domenico Bianculli, Christian Colombo, Ylies Falcone, Adrian Francalanza, Srdjan Krstic, Joao M Lourenco, et al. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods in System Design*, 54(3):279–335, 2019.

[218] Oleg Sokolsky, Klaus Havelund, and Insup Lee. Introduction to the special section on runtime verification, 2012.

[219] Allen Goldberg, Klaus Havelund, and Conor McGann. Runtime verification for autonomous spacecraft software. In *2005 IEEE Aerospace Conference*, pages 507–516. IEEE, 2005.

[220] Eleni Zapridou, Ezio Bartocci, and Panagiotis Katsaros. Runtime verification of autonomous driving systems in carla. In *International Conference on Runtime Verification*, pages 172–183. Springer, 2020.

[221] Angelo Ferrando, Louise A. Dennis, Davide Ancona, Michael Fisher, and Viviana Mascardi. Recognising assumption violations in autonomous systems verification. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 1933–1935, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[222] Andrzej Wardziński. Safety assurance strategies for autonomous vehicles. In *International Conference on Computer Safety, Reliability, and Security*, pages 277–290. Springer, 2008.

[223] Patrik Feth, Daniel Schneider, and Rasmus Adler. A conceptual safety supervisor definition and evaluation framework for autonomous systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 135–148. Springer, 2017.

[224] Zeshan Kurd, Tim Kelly, John McDermid, Radu Calinescu, and Marta Kwiatkowska. Establishing a framework for dynamic risk management in 'intelligent'aero-engine control. In *International Conference on Computer Safety, Reliability, and Security*, pages 326–341. Springer, 2009.

[225] Fabio L Leite, Rasmus Adler, and Patrik Feth. Safety assurance for autonomous and collaborative medical cyber-physical systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 237–248. Springer, 2017.

[226] Kesav Viswanadha, Francis Indaheng, Justin Wong, Edward Kim, Ellen Kalvan, Yash Pant, Daniel J Fremont, and Sanjit A Seshia. Addressing the ieee av test challenge with scenic and verifai. *arXiv preprint arXiv:2108.13796*, 2021.

[227] Nikita Bhardwaj and Peter Liggesmeyer. A runtime risk assessment concept for safe reconfiguration in open adaptive systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 309–316. Springer, 2017.

[228] Fábio L Leite, Daniel Schneider, and Rasmus Adler. Dynamic risk management for cooperative autonomous medical cyber-physical systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 126–138. Springer, 2018.

[229] Johann Schumann, Timmy Mbaya, Ole Mengshoel, Knot Pipatsrisawat, Ashok Srivastava, Arthur Choi, and Adnan Darwiche. Software health management with bayesian networks. *Innovations in Systems and Software Engineering*, 9(4):271–292, 2013.

[230] Sherif Abdelwahed, Abhishek Dubey, Gabor Karsai, and Nagabhushan Mahadevan. Model-based tools and techniques for real-time system and software health management. *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, 285, 2011.

[231] Matthew P Burruss, Shreyas Ramakrishna, Gabor Karsai, and Abhishek Dubey. Deepnncar: A testbed for deploying and testing middleware frameworks for autonomous robots. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 87–88. IEEE, 2019.

[232] Tim Kelly and Rob Weaver. The goal structuring notation–a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, page 6. Citeseer, 2004.

[233] Yakoub Nemouchi, Simon Foster, Mario Gleirscher, and Tim Kelly. Isabelle/sacm: Computer-assisted assurance cases with integrated formal methods. In *International Conference on Integrated Formal Methods*, pages 379–398. Springer, 2019.

[234] Jose Luis de la Vara, Eugenio Parra, Alejandra Ruiz, and Barbara Gallina. The amass tool platform: An innovative solution for assurance and certication of cyber-physical systems. In *REFSQ Workshops*, 2020.

[235] Mike Maksimov, Nick LS Fung, Sahar Kokaly, and Marsha Chechik. Two decades of assurance case tools: a survey. In *International Conference on Computer Safety, Reliability, and Security*, pages 49–59. Springer, 2018.

[236] Shuichiro Yamamoto and Yutaka Matsuno. An evaluation of argument patterns to reduce pitfalls of applying assurance case. In *2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*, pages 12–17. IEEE, 2013.

[237] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Patrick Valduriez, et al. Applying generic model management to data mapping. In *BDA*, 2005.

[238] Ewen Denney and Ganesh Pai. A formal basis for safety case patterns. In *International Conference on Computer Safety, Reliability, and Security*, pages 21–32. Springer, 2013.

[239] [Online] Federal Aviation Administration. Requirements engineering management handbook, 2009. URL https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/ar-08-32.pdf.

[240] Camilo Almendra, Carla Silva, Luiz Eduardo G Martins, and Johnny Marques. How assurance case development and requirements engineering interplay: a study with practitioners. *Requirements Engineering*, pages 1–20, 2022.

[241] Luiz Eduardo G Martins and Tony Gorschek. Requirements engineering for safety-critical systems: A systematic literature review. *Information and software technology*, 75:71–89, 2016.

[242] Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering*, 17(1):57–78, 2012.

[243] David E Arney, Raoul Jetley, Paul Jones, Insup Lee, Arnab Ray, Oleg Sokolsky, and Yi Zhang. Generic infusion pump hazard analysis and safety requirements version 1.0. *Technical Reports (CIS)*, page 893, 2009.

[244] Yi Zhang, Paul L Jones, and Raoul Jetley. A hazard analysis for a generic insulin infusion pump. *Journal of diabetes science and technology*, 4(2):263–283, 2010.

[245] Charles Hartsell, Nagabhushan Mahadevan, Abhishek Dubey, and Gabor Karsai. Automated method for assurance case construction from system design models. In *2021 5th International Conference on System Reliability and Safety (ICSRS)*, pages 230–239, 2021.

[246] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In *Advances in systems safety*, pages 3–23. Springer, 2011.

[247] Ewen Denney and Ganesh Pai. Automating the assembly of aviation safety cases. *IEEE Transactions on Reliability*, 63(4):830–849, 2014.

[248] Khana Chindamaikul, Takai Toshinori, Daniel Port, and Iida Hajimu. Automatic approach to prepare information for constructing an assurance case. In *The 15th International Conference of Product Focused Software Development and Process Improvement*, 2014.

[249] Nagabhushan Mahadevan, Abhishek Dubey, and Gabor Karsai. Application of software health management techniques. In *Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems*, pages 1–10, 2011.

[250] Alexander Felfernig, David Felipe Benavides Cuevas, José Ángel Galindo Duarte, and Florian Reinfrank. Towards anomaly explanation in feature models. In *ConfWS-2013: 15th International Configuration Workshop (2013), p 117-124*. CEUR-WS, 2013.

[251] [Online] nuscenes. nuimage dataset, 2021. URL https://www.nuscenes.org/nuimages.

[252] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[253] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2009.

[254] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent space for multi-label classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[255] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333, 2011.

[256] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 109–120, 2007.

[257] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[258] Shijin Wang, Jianbo Yu, Edzel Lapira, and Jay Lee. A modified support vector data description based novelty detection approach for machinery components. *Applied Soft Computing*, 13(2):1193–1205, 2013.

[259] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402, 2018.

[260] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[261] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[262] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian conference on computer vision*, pages 622–637. Springer, 2018.

[263] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.

[264] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1): 53–65, 2018.

[265] Ha Son Vu, Daisuke Ueta, Kiyoshi Hashimoto, Kazuki Maeno, Sugiri Pranata, and Sheng Mei Shen. Anomaly detection with adversarial dual autoencoders. *arXiv preprint arXiv:1902.06924*, 2019.

[266] Jack Klys, Jake Snell, and Richard Zemel. Learning latent subspaces in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 6444–6454, 2018.

[267] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, X-avier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexan-der Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. *ICLR17*, 2016.

[268] Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning gan for pose-invariant face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1415–1424, 2017.

[269] Xi Peng, Xiang Yu, Kihyuk Sohn, Dimitris N Metaxas, and Manmohan Chandraker. Reconstruction-based disentanglement for pose-invariant face recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1623–1632, 2017.

[270] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *Advances in Neural Information Processing Systems*, pages 517–526, 2018.

[271] Shuo Wang, Tianle Chen, Shangyu Chen, Carsten Rudolph, Surya Nepal, and Marthie Grobler. Oiad: One-for-all image anomaly detection with disentanglement learning. *arXiv preprint arXiv:2001.06640*, 2020.

[272] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

[273] Marek Jakab, Lukas Hudec, and Wanda Benesova. Partial disentanglement of hierarchical variational auto-encoder for texture synthesis. *IET Computer Vision*, 14(8):564–574, 2020.

[274] Emile Mathieu, Tom Rainforth, Nana Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pages 4402–4412. PMLR, 2019.

[275] Francisco J Martinez-Estudillo, Pedro Antonio Gutiérrez, César Hervás, and Juan Carlos Fernández. Evolutionary learning by a sensitivity-accuracy approach for multi-class problems. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1581–1588. IEEE, 2008.

[276] Sheri Edwards. Elements of information theory, thomas m. cover, joy a. thomas, john wiley & sons, inc.(2006), 2008.

[277] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[278] Tian Qi Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2610–2620, 2018.

[279] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

[280] Valentina Fedorova, Alex Gammerman, Ilia Nouretdinov, and Vladimir Vovk. Plug-in martingales for testing exchangeability on-line. *arXiv preprint arXiv:1204.3251*, 2012.

[281] Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. prentice Hall Englewood Cliffs, 1993.

[282] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[283] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[284] Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. Bayesian optimization with inequality constraints. In *ICML*, volume 2014, pages 937–945, 2014.

[285] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[286] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[287] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2019.

[288] Igor Dejanović, Renata Vaderna, Gordana Milosavljević, and Željko Vuković. Textx: a python tool for domain-specific languages implementation. *Knowledge-Based Systems*, 115:1–4, 2017.

[289] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[290] Giampaolo Rodola. Psutil package: a cross-platform library for retrieving information on running processes and system utilization, 2016.

[291] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[292] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[293] Christina Heinze-Deml and Nicolai Meinshausen. Conditional variance penalties and domain shift robustness. *Machine Learning*, 110(2):303–348, 2021.

[294] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. *arXiv preprint arXiv:1805.12018*, 2018.

[295] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11293–11302, 2019.

[296] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[297] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[298] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[299] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[300] Yonatan Geifman, Guy Uziel, and Ran El-Yaniv. Bias-reduced uncertainty estimation for deep neural classifiers. *arXiv preprint arXiv:1805.08206*, 2018.

[301] Susmit Jha, Sunny Raj, Steven Fernandes, Sumit K Jha, Somesh Jha, Brian Jalaian, Gunjan Verma, and Ananthram Swami. Attribution-based confidence metric for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[302] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.

[303] Heiko Hoffmann. Kernel pca for novelty detection. *Pattern recognition*, 40(3):863–874, 2007.

[304] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[305] Luigi Pietro Cordella, Claudio De Stefano, Francesco Tortorella, and Mario Vento. A method for improving classification reliability of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 6 (5):1140–1147, 1995.

[306] Penny Chong, Lukas Ruff, Marius Kloft, and Alexander Binder. Simple and effective prevention of mode collapse in deep one-class classification. *arXiv preprint arXiv:2001.08873*, 2020.

[307] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[308] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. Adversarially learned anomaly detection. In *2018 IEEE International conference on data mining (ICDM)*, pages 727–736. IEEE, 2018.

[309] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[310] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[311] Tucker Graydon and Ferat Sahin. Novelty detection and analysis with a Beta-VAE network. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2687–2691. IEEE, 2018.

[312] Xiaoyan Li. *Anomaly Detection Based on Disentangled Representation Learning*. PhD thesis, Université d'Ottawa/University of Ottawa, 2020.

[313] Leixin Zhou, Wenxiang Deng, and Xiaodong Wu. Unsupervised anomaly localization using vae and beta-vae. *arXiv preprint arXiv:2005.10686*, 2020.

[314] Amanda Clare and Ross D King. Knowledge discovery in multi-label phenotype data. In *European conference on principles of data mining and knowledge discovery*, pages 42–53. Springer, 2001.

[315] Gesina Schwalbe and Martin Schels. A survey on methods for the safety assurance of machine learning based systems. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.

[316] Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety*, 4(1):15–24, 2016.

[317] Christos Katrakazas, Mohammed Quddus, and Wen-Hua Chen. A new integrated collision risk assessment methodology for autonomous vehicles. *Accident Analysis & Prevention*, 127:61–79, 2019.

[318] Gerald Steinbauer and Franz Wotawa. Model-based reasoning for self-adaptive systems–theory and practice. In *Assurances for Self-Adaptive Systems*, pages 187–213. Springer, 2013.

[319] Nima Khakzad, Faisal Khan, and Paul Amyotte. Dynamic risk analysis using bow-tie approach. *Reliability Engineering & System Safety*, 104:36–44, 2012.

[320] Refaul Ferdous, Faisal Khan, Rehan Sadiq, Paul Amyotte, and Brian Veitch. Handling data uncertainties in event tree analysis. *Process safety and environmental protection*, 87(5):283–292, 2009.

[321] Yao Zhang and Xin Guan. Selecting project risk preventive and protective strategies based on bow-tie analysis. *Journal of Management in Engineering*, 34(3):04018009, 2018.

[322] Sandy L Zabell. The rule of succession. *Erkenntnis*, 31(2):283–321, 1989.

[323] Dhanoop Karunakaran, Stewart Worrall, and Eduardo Nebot. Efficient statistical validation with edge cases to evaluate highly automated vehicles. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

[324] José Luis Pech-Pacheco, Gabriel Cristóbal, Jesús Chamorro-Martinez, and Joaquín Fernández-Valdivia. Diatom autofocusing in brightfield microscopy: a comparative study. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 3, pages 314–317. IEEE, 2000.

[325] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

[326] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[327] Musa Morena Marcusso Manhães, Sebastian A. Scherer, Martin Voss, Luiz Ricardo Douat, and Thomas Rauschenbach. UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In *OCEANS 2016 MTS/IEEE Monterey*. IEEE, sep 2016. doi: 10.1109/oceans.2016.7761080. URL https://doi.org/10.1109%2Foceans.2016.7761080.

[328] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2147–2154, 2014.

[329] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[330] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Attacking vision-based perception in end-to-end autonomous driving models. *arXiv preprint arXiv:1910.01907*, 2019.

[331] Sandeep Neema. Assured autonomy. *DARPA Research Program.*, 2019.

[332] Stanley Bak, Deepti K Chivukula, Olugbemiga Adekunle, Mu Sun, Marco Caccamo, and Lui Sha. The system-level simplex architecture for improved real-time embedded system safety. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 99–107. IEEE, 2009.

[333] Joel Finch. Toyota sudden acceleration: a case study of the national highway traffic safety administration-recalls for change. *Loy. Consumer L. Rev.*, 22:472, 2009.

[334] Daniel Jiménez. Dynamically weighted ensemble neural networks for classification. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 1, pages 753–756. IEEE, 1998.

[335] Weicun Zhang. Stable weighted multiple model adaptive control with improved convergence rate. *IFAC Proceedings Volumes*, 45(13):570–575, 2012.

[336] Shimon Ullman. Against direct perception. *Behavioral and Brain Sciences*, 3(3):373–381, 1980.

[337] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[338] Dario Mantegazza, Jérôme Guzzi, Luca M Gambardella, and Alessandro Giusti. Vision-based control of a quadrotor in user proximity: Mediated vs end-to-end learning approaches. *arXiv preprint arXiv:1809.08881*, 2018.

[339] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[340] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[341] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[342] Max Bajracharya, Andrew Howard, Larry H Matthies, Benyang Tang, and Michael Turmon. Autonomous off-road navigation with end-to-end learning for the LAGR program. *Journal of Field Robotics*, 26(1):3–25, 2009.

[343] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986. ISSN 1939-3539. doi: 10.1109/TPAMI.1986.4767851.

[344] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[345] Kevin McFall. Using visual lane detection to control steering in a self-driving vehicle. In *Smart City 360°*, pages 861–873. Springer, 2016.

[346] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[347] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[348] Ayan Mukhopadhyay, Zilin Wang, and Yevgeniy Vorobeychik. A decision theoretic framework for emergency responder dispatch. In *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[349] Alberto Maria Metelli. Configurable environments in reinforcement learning: An overview. *Special Topics in Information Technology*, pages 101–113, 2022.

[350] Claude Touzet and MA Arbib. Q-learning for robots. In *The handbook of brain theory and neural networks*, pages 934–937. MIT Press Cambridge, 2003.

[351] Frederick Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9):921–932, 1985.

[352] Carl G Looney. Fuzzy petri nets for rule-based decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):178–183, 1988.

[353] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of field Robotics*, 25(8):425–466, 2008.

[354] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[355] Lex Fridman, Benedikt Jenik, and Bryan Reimer. Arguing machines: Perceptioncontrol system redundancy and edge case discovery in real-world autonomous driving. *arXiv preprint arXiv:1710.04459*, 2017.

[356] Ali Gurcan Ozkil, Zhun Fan, Steen Dawids, Henrik Aanes, Jens Klestrup Kristensen, and Kim Hardam Christensen. Service robots for hospitals: A case study of transportation tasks in a hospital. In *2009 IEEE International Conference on Automation and Logistics*, pages 289–294. IEEE, 2009.

[357] Kim Bhasin and Patrick Clark. How amazon triggered a robot arms race. *Bloomberg Techology*, 2016.

[358] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor, Kenneth Gregson, Winter Guerra, Owen Guldner, Mubarik Mohamoud, Brian Plancher, Robert Shin, et al. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit. In *2017 IEEE Integrated STEM Education Conference (ISEC)*, pages 195–203. IEEE, 2017.

[359] Raspberry Pi frequency management, 2015. URL https://www.raspberrypi.org/documentation/hardware/raspberrypi/frequency-management.md.

[360] G Rodola. psutil documentation; 2017, 2017.

[361] Martin Sústrik et al. Zeromq. *Introduction Amy Brown and Greg Wilson*, 2015.

[362] Gul A Agha. Actors: A model of concurrent computation in distributed systems. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1985.

[363] Myeong-Wuk Jang, Smitha Reddy, Predrag Tosic, Liping Chen, and Gul Agha. An actor-based simulation for studying uav coordination. In *15th European Simulation Symposium*, page 323, 2005.

[364] A. Dubey, G. Karsai, P. Volgyesi, M. Metelko, I. Madari, H. Tu, Y. Du, and S. Lukic. Device access abstractions for resilient information architecture platform for smart grid. *IEEE Embedded Systems Letters*, pages 1–1, 2018. ISSN 1943-0663. doi: 10.1109/LES.2018.2845854.

[365] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[366] Junxing Yang, Md Ariful Islam, Abhishek Murthy, Scott A Smolka, and Scott D Stoller. A simplex architecture for hybrid systems using barrier certificates. In *International Conference on Computer Safety, Reliability, and Security*, pages 117–131. Springer, 2017.

[367] Dung Phan, Junxing Yang, Matthew Clark, Radu Grosu, John D. Schierman, Scott A. Smolka, and Scott D. Stoller. A component-based simplex architecture for high-assurance cyber-physical systems. In *17th International Conference on Application of Concurrency to System Design, ACSD 2017, Zaragoza, Spain, June 25-30, 2017*, pages 49–58, 2017. doi: 10.1109/ACSD.2017.23. URL https://doi.org/10.1109/ACSD.2017.23.

[368] Xiaofeng Wang, Naira Hovakimyan, and Lui Sha. L1simplex: Fault-tolerant control of cyber-physical systems. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 41–50. IEEE, 2013.

[369] Naira Hovakimyan and Chengyu Cao. $\mathscr{L}_1$ *adaptive control theory: Guaranteed robustness with fast adaptation*. SIAM, 2010.

[370] Tudor B Ionescu. Adaptive simplex architecture for safe, real-time robot path planning. *Sensors*, 21(8): 2589, 2021.

[371] Ronny Seiger, Christoph Seidl, Uwe Aßmann, and Thomas Schlegel. A capability-based framework for programming small domestic service robots. In *Proceedings of the 2015 Joint MORSE/VAO Workshop*

*on Model-Driven Robot Software Engineering and View-based Software-Engineering*, pages 49–54. ACM, 2015.

[372] Elias De Coninck, Steven Bohez, Sam Leroux, Tim Verbelen, Bert Vankeirsbilck, Bart Dhoedt, and Pieter Simoens. Middleware platform for distributed applications incorporating robots, sensors and the cloud. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pages 218–223. IEEE, 2016.

[373] Steven Bohez, Elias De Coninck, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. Enabling component-based mobile cloud computing with the aiolos middleware. In *Proceedings of the 13th Workshop on Adaptive and Reflective Middleware*, page 2. ACM, 2014.

[374] Chao Liu and Pingyu Jiang. A cyber-physical system architecture in shop floor for intelligent manufacturing. *Procedia Cirp*, 56:372–377, 2016.

[375] Tooba Samad, Sohail Iqbal, Asad Waqar Malik, Omar Arif, and Peter Bloodsworth. A multi-agent framework for cloud-based management of collaborative robots. *International Journal of Advanced Robotic Systems*, 15(4):1729881418785073, 2018.

[376] Michael Garrett Bechtel, Elise McEllhiney, and Heechul Yun. DeepPicar: A low-cost deep neural network-based autonomous car. *arXiv preprint arXiv:1712.08644*, 2017.

[377] W Roscoe. Donkey car: An opensource diy self driving platform for small scale cars, 2020.

[378] Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J Kennedy. Training deep neural networks on imbalanced data sets. In *2016 international joint conference on neural networks (IJCNN)*, pages 4368–4374. IEEE, 2016.

[379] Lina Castano and Huan Xu. Safe decision making for risk mitigation of uas. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1326–1335. IEEE, 2019.

[380] Haobin Shi, Zhiqiang Lin, Shuge Zhang, Xuesi Li, and Kao-Shing Hwang. An adaptive decision-making method with fuzzy bayesian reinforcement learning for robot soccer. *Information Sciences*, 436: 268–281, 2018.

[381] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[382] David Lenz, Tobias Kessler, and Alois Knoll. Tactical cooperative planning for autonomous highway driving using monte-carlo tree search. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 447–453. IEEE, 2016.

[383] Xin Huang, Sungkweon Hong, Andreas Hofmann, and Brian C Williams. Online risk-bounded motion planning for autonomous vehicles in dynamic environments. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 214–222, 2019.

[384] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. *arXiv preprint arXiv:2001.07362*, 2020.

[385] Christopher B Kuhn, Markus Hofbauer, Goran Petrovic, and Eckehard Steinbach. Introspective failure prediction for autonomous driving using late fusion of state and camera information. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[386] Jerome H Friedman, Jon Louis Bentley, and Raphael A Finkel. *An algorithm for finding best matches in logarithmic time*. Department of Computer Science, Stanford University, 1975.

[387] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4 (1):1–43, 2012.

[388] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006. doi: 10.1007/11871842_29. URL https://doi.org/10.1007/11871842_29.

[389] Krzysztof Lis, Krishna Nakka, Pascal Fua, and Mathieu Salzmann. Detecting the unexpected via image resynthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2152–2161, 2019.

[390] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[391] Andrea Ceccarelli and Francesco Secci. Rgb cameras failures and their effects in autonomous driving applications. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[392] Peter Kafka. The automotive standard iso 26262, the innovative driver for enhanced safety assessment & technology for motor cars. *Procedia Engineering*, 45:2–10, 2012.

[393] [Online] Waymo. Waymo safety report, 2021. URL https://waymo.com/safety/.

[394] [Online] Tesla. Tesla vehicle safety report, 2022. URL https://www.tesla.com/VehicleSafetyReport.

[395] Houssam Abbas, Matthew O'Kelly, Alena Rodionova, and Rahul Mangharam. Safe at any speed: A simulation-based test harness for autonomous vehicles. In *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*, pages 94–106. Springer, 2017.

[396] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1555–1562. IEEE, 2018.

[397] Tong Duy Son, Ajinkya Bhave, and Herman Van der Auweraer. Simulation-based testing framework for autonomous driving development. In *2019 IEEE International Conference on Mechatronics (ICM)*, volume 1. IEEE, 2019.

[398] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. Paracosm: A test framework for autonomous driving simulations. In *International Conference on Fundamental Approaches to Software Engineering*, pages 172–195. Springer, Cham, 2021.

[399] [Online] CARLA. Carla leaderboard challenge, 2020. URL https://leaderboard.carla.org/.

[400] [Online] Aurora. Aurora safety case framework, 2022. URL https://safetycaseframework.aurora.tech/gsn.

[401] Christopher B Kuhn, Markus Hofbauer, Goran Petrovic, and Eckehard Steinbach. Trajectory-based failure prediction for autonomous driving. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 980–986. IEEE, 2021.

[402] Wenhao Ding, Baiming Chen, Minjun Xu, and Ding Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2243–2250. IEEE, 2020.

[403] Salman Khan, Munawar Hayat, Syed Waqas Zamir, Jianbing Shen, and Ling Shao. Striking the right balance with uncertainty. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 103–112, 2019.

[404] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[405] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *23rd International conference on intelligent transportation systems*, pages 1–6. IEEE, 2020.

[406] Voyage. Deepdrive simulator, 2020. URL https://deepdrive.io/.

[407] Ding Zhao, Xianan Huang, Huei Peng, Henry Lam, and David J LeBlanc. Accelerated evaluation of automated vehicles in car-following maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[408] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Sim-atav: Simulation-based adversarial testing framework for autonomous vehicles. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*, pages 283–284, 2018.

[409] Jernej Jerebic, Marjan Mernik, Shih-Hsi Liu, Miha Ravber, Mihael Baketarić, Luka Mernik, and Matej Črepinšek. A novel direct measure of exploration and exploitation based on attraction basins. *Expert Systems with Applications*, 167:114353, 2021.

[410] Keith Dalbey, Michael Eldred, Gianluca Geraci, John Jakeman, Kathryn Maupin, Jason A Monschke, Daniel Seidl, Anh Tran, Friedrich Menhorn, and Xiaoshu Zeng. Dakota a multilevel parallel object-oriented framework for design optimization parameter estimation uncertainty quantification and sensitivity analysis: Version 6.14 theory manual. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2021.

[411] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[412] Wenhao Ding, Wenshuo Wang, and Ding Zhao. A new multi-vehicle trajectory generator to simulate vehicle-to-vehicle encounters. *arXiv preprint arXiv:1809.05680*, 2018.

[413] Harsh Vardhan and Janos Sztipanovits. Rare event failure test case generation in learning-enabled-controllers. In *2021 6th International Conference on Machine Learning Technologies*, pages 34–40, 2021.

[414] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, John Duchi, and Russ Tedrake. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *arXiv preprint arXiv:1811.00145*, 2018.

[415] Cumhur Erkan Tuncali, Theodore P Pavlic, and Georgios Fainekos. Utilizing s-taliro as an automatic test generation framework for autonomous vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1470–1475. IEEE, 2016.

[416] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[417] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15590–15599, 2021.

[418] Oliver Hummel and Colin Atkinson. The managed adapter pattern: Facilitating glue code generation for component reuse. In Stephen H. Edwards and Gregory Kulczycki, editors, *Formal Foundations of Reuse and Domain Engineering*, pages 211–224. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04211-9.

[419] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov), 2002.

[420] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020.

[421] Ramneet Kaur, Radoslav Ivanov, Matthew Cleaveland, Oleg Sokolsky, and Insup Lee. Assurance case patterns for cyber-physical systems with deep neural networks. In *International Conference on Computer Safety, Reliability, and Security*, pages 82–97. Springer, 2020.

LIST OF ABBREVIATIONS

**$\beta$-VAE** $\beta$-Variational Autoencoder

**A-G** Assume-Guarantee

**AC** Assurance Case

**AE** Autoencoder

**AI** Artificial Intelligence

**ALARP** As Low As Reasonably Practicable

**ASARP** As Safe As Reasonably Practicable

**AV** Autonomous Vehicle

**BR** Binary Relevance

**BTD** Bow-Tie Diagram

**CPS** Cyber-Physical System

**DNN** Deep Neural Network

**DSS** Dynamic Simplex Strategy

**GAN** Generative Adversarial Network

**GSN** Goal Structuring Notation

**KL** Kullback–Leibler

**LEC** Learning Enabled Component

**LMI** Linear Matrix Inequality

**MCTS** Monte Carlo tree search

**ML** Machine Learning

**RL** Reinforcement Learning

**SDL** Scenario Description Language

**SHM** System Health Management

**TFPG** Timed Failure Propagation Graph

**UAV** Unmanned Aerial Vehicle

**UUV** Unmanned Underwater Vehicle

**VAE** Variational Autoencoder

**WSS** Weighted Simplex Strategy