SAFETY ASSURANCE OF AUTONOMOUS LEARNING-ENABLED CYBER PHYSICAL SYSTEMS

By

Patrick Musau

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

May 31, 2022

Nashville, Tennessee

Approved:

Taylor Johnson Ph.D.

Gabor Karsai Ph.D.

Janos Sztipanovits Ph.D.

Richard Alan Peters Ph.D.

Stanley Bak Ph.D.

Dedicated to my family, friends, and everyone who has supported me through my life.

ACKNOWLEDGMENTS

In his poem “Satisfaction”, Otto Rene Castillo writes beautifully that ”the most beautiful thing, for those who have fought a whole life, is to come to the end and say; we believed in people and life, and life and the people never let us down.” In the spirit of that message, I would like to say that both this dissertation and I are better because of all the people who made this possible. I would like to take the time to acknowledge the people that have helped and encouraged me along the way.

First, I would like to express my appreciation and gratitude to my committee members Professor Taylor Johnson, Professor Gabor Karsai, Professor Janos Sztipanovits, Professor Alan Peters, and Professor Stanley Bak for the technical guidance given to me along the way. Their expertise in formal methods, control theory, model integrated computing, robotics, and machine learning verification have greatly impacted this work. I would like to especially acknowledge my advisor, Taylor Johnson, for his dedicated guidance and help in overcoming numerous obstacles and challenges during my doctoral study.

To my colleagues and now good friends, Diego Manzanas Lopez, and Nathaniel Hamilton, thank you for your insight, encouragement, and support over these last couple of years. Whether it was spending time together at game nights, happy hours, or scrambling to get a paper in minutes before the deadline, thank you for being present for me during the highs and lows of graduate school.

I would also like to thank Neelanjana Pal, Ayana Wild, Preston Robinette, and Xiaodong Yang, Paulius Stankaitis, Shreyas Ramakrishna, Preston Robinette, Timothy Darrah, Nagabhushan Mahadevan, Daniel Stojcsics, and Hoang-Dung Tran who I had the pleasure of working with over the last couple of years. Tran, you quite literally helped me decipher how to be a graduate student and helped my find my footing when I felt lost. Thank you so much. To Katherine Scott, Michael Jeronimo, Michael Caroll, and Addisu Taddese, thank you for your kindness and mentorship both during my internship at Open Robotics and beyond.

To my brothers, Joseph Mugisha, and Thomas Trankle. Thank you for the many mornings, and evenings you all spent with me while I wrote this document and did so much of my work. Thank you for the pep talks, the laughs, the regular and random FaceTime calls, and everything in between. You all have been with me since the start of this journey and I cannot thank you enough. I am deeply thankful for your friendship. To Erskine Nyoike, thank you for the random drop-ins, our morning gym sessions, the advice you offered, and all the random adventures we went on together. To Joe Inger, I finally made ”car go round track.” Thank you for your support, the many laughs and the energy you bring to my life. To Kyalo Muindi, I wish you were still here, man. Thank you for always seeing the best in me, for inspiring me, and for the joy you shared with me always. I miss you deeply.

The road to graduate school was not always a straight one for me and I would like to thank my dad, Dr. George Ruigu, and Dr. William Baker, Dr. William Brantley, and Dr. Andrea Tartaro for inspiring me to go to graduate school. I would not have started this journey if it were not for your help, kindness, and guidance. I would also like to thank Dr. Courtney Williams, Dr. Ciera Scott, Walker Duncan, and Elizabeth Hunt for the many conversations that got me through the last couple of years.[1] Thank you for walking with me through the good and rough patches.

I would also like to thank Haley Adams, Sarah Miele, for all the writing sessions and feedback that you were kind enough to offer. Thank you also for all our random hangouts and good times. To the members of the Black Codes, Trevor Guinn, Robert Collins III, Calvin Foster, Kanesha Patterson, Curtis Crutchfield, Dr. Teresa Vasquez, Lataevia Berry, thank you for your support, kindness, and for inspiring me every day. You all are incredible.

I want to express my greatest gratitude to my mom, Lucy Musau, and my sister Mary Musau. Where do I even begin? Over the last several years, I have had the privilege of pursuing my PhD, which was only possible because of your support and belief in me throughout my life. Thank you mom for modeling excellence to me in every way imaginable, and for instilling in me the desire to always do my best no matter the circumstances. Thank you for your tremendous support and for everything you are. Thank you, Mary, for being a role model in my life, for being a powerful force for good in my life, and for letting me be me no matter what.

Finally, I want to thank several other people who have played a significant role in my life through their friendship. Aidan Clarke, Mikey Martinez, Brian Ondeng, Ndirangu Warugongo, Alberto Esteban Linares,

[1] These individuals were my counselors at the university counseling center, and I would like to encourage graduate students to take advantage of such services as they navigate the academy.

Shannon Hurlston, Michael Roman, Brooks Musangu, Augustus Hayfron, Kayula Kaonga, Larkidus Robinson, Devika Harshan, Latif Gbadamoshie, Nathan Ring, Damon Davoudpour Ryan Conlogue, Adam Miller, Brad Potteiger, Tim Potteiger, Kenneth Konam, Tabitha Colter, Chelsea Lin, Regan Williams, Alexis Rodriguez, Frankie King, Webster Heath, Deanna Meador, Michael Bryant, Austin Dozier, and all my brothers on the Nashville Rugby Football Club. You all have had a profound impact on my life. There are so many others I want to thank, you know who you are. Thank you from the bottom of my heart.

As I sit here absorbed by a deep feeling of gratitude, remembering the highest of highs and the lowest and lows over the last several years, I cannot but think of David Whyte's reflection on friendship. "The ultimate touchstone of friendship is not improvement, neither of the other nor of the self: the ultimate touchstone is witness, the privilege of having been seen by someone and the equal privilege of being granted the sight of the essence of another, to have walked with them and to have believed in them, and sometimes just to have accompanied them for however brief a span, on a journey impossible to accomplish alone." This journey would have been impossible alone, so may this dissertation be a reflection of the power of community.

Acknowledgement of Support

This material is based upon work supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-22-1-0019, the National Science Foundation (NSF) under grant numbers 1918450, 1910017, and 2028001, the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, nor NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

I.1 Motivation

The vision of an ”automated” future has enraptured researchers, hobbyists, and corporations for decades. In recent years, the growth in the number and diversity of semi and fully-autonomous systems, has allowed us to re-imagine the ways we organize our cities, communicate, and move around. Underpinning these advancements has been the stunning progress in Artificial Intelligence (AI), where the success of models such as artificial neural networks has allowed for technologies including Amazon's Alexa, Apple's Siri, and DeepMind's AlphaGo to flourish and enter everyday conversation [5]. AI has been lauded as one of the most influential and disruptive set of methodologies of our era, and its applications within the autonomous domain has the potential to bring forth unprecedented benefits for society as a whole.

As with most disruptive technologies, societal acceptance of autonomous systems hinges on an ability to engender trust in the underlying technology. Unfortunately, in recent years, there has been a steady decline in trust within this space. As Mueller et al. put it, this decline can be attributed to ”the steady drumbeat of news stories of AI failures that range from the humorous to tragic” [6]. Rebuilding confidence in these systems will require holistic analyses of their design and their interactions with the world, particularly within the realm of safety. The following document focuses on the safety dimension of this challenge, recognizing that the development of trustworthy autonomous systems requires the use of multidimensional and interdisciplinary assurance approaches.

To address the safety challenge for autonomous systems, many assurance techniques combine reasoning about the functional correctness of a system at design time with monitoring relevant safety specifications at runtime [7]. These approaches often draw inspiration from control theory, model based engineering, formal methods, and sensor and actuator design [5]. Since autonomous systems generally operate in uncertain, unstructured, and variable environments, reasoning about safety is a notoriously difficult problem. Particularly since these systems often make use of *Machine Learning* (ML) or *Learning Enabled Components* (LECS) to handle the complexities of their environments and decipher the information observed from a diverse configuration of sensors [8]. Many of the existing assurance approaches are not well suited to handling these components due to their complexity and opaque nature. It is unrealistic to believe that one will ever verify all parts of an autonomous system [9]. Rather, the objective is to demonstrate that the system meets acceptable levels of safety, and adheres to engineering principles such as the As Safe As Reasonably Practicable

(ASARP) Principle [10]. This requires, the design of solutions that are both practical and rigorous.

I.2 Research Challenges

Reasoning about the safety of an autonomous system requires an understanding of the collective interaction of computers, networks, and the system's physical dynamics [8]. Thus, these systems fall within the realm of Cyber-Physical Systems (CPS). While there are slight variations of this notion, the underlying principle is that the computational and physical aspects of the system cannot be easily decoupled [11]. CPS are present within a wide range of engineering domains such as avionics [12], automotive systems (autonomous vehicles and smart cars), medical devices, industrial process controls [13], smart grids, traffic safety and control, robotics, and the internet of things (IOT) [14]. Many of these domains are safety critical in nature, and it is generally required that there be an orderly development process that begins with outlining a set of requirements, and then demonstrating that these requirements have been met consistently throughout the design [10]. The challenge here is that often it mandates reasoning about all the scenarios that a system will encounter, as well as the failure cases that could occur. Thus, one must potentially explore infinite spaces with finite resources [10]. *The task therefore becomes how to deliver maximally credible arguments by expending resources in proportion to the risks, that a system satisfies its requirements* [10].

Arguments about safety classically involve determinations of rates of failure, the results of formal verification, fault-tolerant design, and the evaluation of test cases. There are notable challenges that arise within this realm. The first challenge consists of obtaining suitable models of the underlying system (here "suitable" means that our model omits only inessential details) [15]. The second challenge is developing adequate specifications that ensure correct behavior. Finally, the third challenge consists of developing techniques that can analyze the aforementioned specifications. The following dissertation focuses on these latter two questions. The questions that this document sets out to answer are:

- How do we construct evidence that an autonomous system satisfies its requirements based on our assumptions about the nature of the environment it is tasked with operating within and the nature of the components governing its behavior?
- How does one provide guarantees of correct behavior, when methods such as machine learning are utilized within these systems?
- How does one ensure safety in dynamic contexts characterized by significant uncertainty?
- Finally, how can one reason about the correctness of a system in contexts where the system's dynamics are hard to characterize precisely?

I.3 Research Contributions

To address some of these challenges, we focus on the following contributions:

- We begin by presenting a case study of the use of two leading machine learning methods for training neural network controllers, Reinforcement Learning and Imitation Learning, for the control of a 1/0 scale autonomous vehicle. This case study is aimed at motivating the need to monitor learning-enabled components, and discusses several challenges related to assuring the safety of autonomous learning-enabled systems.

- We then introduce a safety monitoring regime leveraging real-time reachability that allows us to provide provable guarantees of safety for systems that make use of machine learning controllers.

- We propose a simplex architecture for the safety assurance of autonomous systems that abstracts away the need to analyze the correctness of controllers used within these systems and instead focuses on the effects of their decisions on the system's future states.

- We present a rigorous empirical analysis of accounting for various classes of uncertainty within the safety assurance task.

- Finally, we propose the development of a model predictive control regime Leveraging Real-Time Reachability, in an effort to design safe navigation strategies within the presence of dynamic and static obstacles.

I.4 Organization

The remainder of this dissertation is organized as follows. Chapter II presents an introduction to safety assurance and surveys the relevant research literature. This chapter is divided into several sections that present safety assurance approaches for CPS at runtime and at design time. The chapter concludes with a discussion of assurance techniques for machine learning components and a brief discussion of real-time systems. Chapter III presents a case study that deals with the comparison of two leading machine learning methods, Reinforcement Learning and Imitation Learning, within the context of autonomous racing. This chapter helps motivate the need to monitor machine learning components. Chapter IV presents an online safety assurance architecture for machine learning components. The regime makes use of a real-time reachability algorithm that (a) provides provable guarantees of safety, and (b) is used to detect potentially unsafe scenarios for an autonomous driving task. The method is evaluated both in simulation and on a hardware platform. Chapter V presents an extension of the approaches discussed in Chapter IV to handle dynamic obstacles, as well as uncertainty with respect to the models of the system and its environment that are utilized in the reachability

regime. The approach is evaluated in an autonomous racing context commonly utilized by the international F1/10 Autonomous Racing Competition. Chapter VI presents an optimization based approach for the static and dynamical obstacle avoidance problem within an autonomous vehicle racing context. This work combines the model predictive control paradigm with the approach outlined in Chapter IV, in order to realize safe racing strategies. Chapter VII, presents a discussion of the challenges in designing learning-enabled autonomous systems that possess rigorous assurances of correctness with respect to formal mathematical specifications. It explores the limitations and assumptions of many assurance approaches present within the research literature, as well as the work presented in this document, from a practical and philosophical standpoint. Chapter VIII ends with concluding remarks, and Chapter IX lists the author's current publications. Finally, Appendix A presents a detailed description of the source code and experimental artifacts used within our work.

I.5 Copyright Acknowledgements

The required copyright statements for permission to reprint portions of [16, 17] are included in the following.

- For portions of [17] reproduced in this dissertation, we acknowledge the IEEE copyright: © 2022 IEEE. Reprinted, with permission, from Nathaniel Hamilton, Patrick Musau, Diego Manzanas Lopez, and Taylor Johnson, "Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning," IEEE International Conference on Assured Autonomy (ICAA), March 2022.

- For portions of [16] reproduced in this dissertation, we acknowledge the IEEE copyright: © 2022 IEEE. Reprinted, with permission, from Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston Robinette, and Taylor Johnson, "On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers," IEEE International Conference on Assured Autonomy (ICAA), March 2022.

CHAPTER II

Related Work

II.1 Safety Assurance

II.1.1 What is Safety Assurance?

Safety Assurance is defined as the process of creating clear and comprehensible arguments that a system will operate as intended in a particular context and not cause harm. Harm, in this case, can be defined as "any condition that results in death, injury, damage to or loss of equipment, and/or damage to the environment," [18]. To engender confidence in the correctness of a particular system, it is imperative that these arguments establish how a system's design considers all the possible ways in which things could go wrong, and that these scenarios are effectively addressed [19]. In practice however, as Rushby et al. state, "the difficulty lies in statements like 'everything that could go wrong'. This requires that one explore a potentially infinite space [of scenarios and system designs] with only finite resources. The challenge of assurance, therefore, is to "deliver maximally credible arguments and evidence for believing that the system will do no harm, while recognizing that it cannot provide an absolute guarantee," [11].

To formalise this notion, safety assurance involves a rigorous consideration of the assumptions of the system's operating environment within the context of sources of evidence that its construction is correct. This involves an orderly elaboration of requirements and specifications, followed by analyses of the overall system implementation. In this context, requirements codify what the nature of the system is, and specifications are translations of those requirements into properties that can be analyzed. Each industry has its own set of standards, guidelines, and best practices that are usually outlined by domain experts, but the central idea is that design process must adhere to rigorous certification procedures [11].

II.1.2 Assurance Cases

Arguments about the safety of a system are often presented in the form of *Assurance Cases* [10]. Assurance Cases are structured arguments that consist of three elements: a *claim* that clearly states the property of the system to be investigated, *evidence* that the design and construction of the system satisfies the property under consideration, and an *argument* demonstrating how the evidence is sufficient to establish the aforementioned claim [10]. Arguments of this nature typically involve a hierarchical decomposition of arguments and evidence, and they are often represented using graphical notations such as Goal Structuring Notation or Claims-Argument-Evidence frameworks [1]. An example of a Goal-Structuring Notation diagram is shown in Figure II.1, where a claim is justified by an argument and evidence, which is then substantiated by further

subclaims with corresponding arguments and evidence.

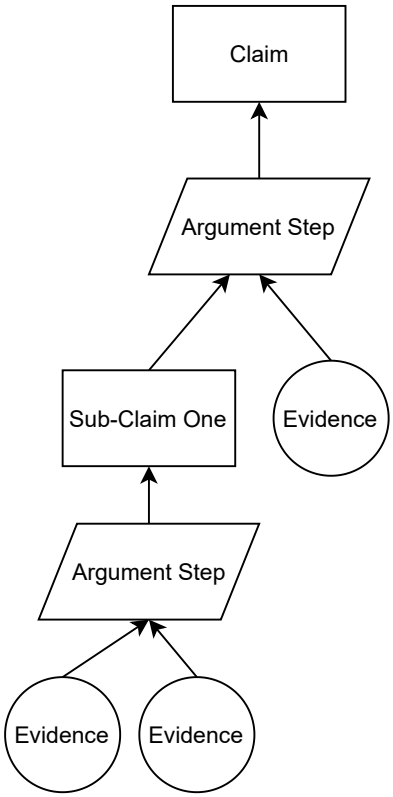

Figure II.1: Example of a structured argument in Goal Structured Notation [1].

Within the context of assurance cases for software, the underlying arguments can employ evidence that can be partially automated through formal methods, testing, and verification [11]. Thus, for software, an assurance argument typically involves demonstrating that a given software implementation satisfies its requirements. The more serious the failure conditions of the underlying software artifact, the more rigorous the evidence demonstrating its correctness must be. Additionally, the evidence must be accompanied by arguments about why we can trust the results of any analyses, and the assumptions made in their consideration. Thus, the entire process seeks to answer two fundamental questions. Are we building the right system (*Validation*)? Are we building the system right (*Verification*)? [11].

In surveying the relevant literature, we focus on these latter two questions. In particular, our survey focuses on these questions within the context of autonomous learning-enabled CPS, which has seen a large increase in the number of approaches proposed towards demonstrating their correctness both at design time and at runtime.

II.2 Design-Time Assurance Techniques

The safety assurance task for autonomous CPS requires drawing on knowledge from different areas such as control engineering, software engineering, and networking. As is characteristic of most engineering systems, the design of CPS predominantly makes use of model-based design (MBD) [20]. In this context, a model can be defined as a formal description of a system which allows us to mathematically consider properties such as reliability, fault tolerance, timing behavior, performance characteristics, and other relevant system parameters [21]. The main benefit of models is that while it is often not possible to reason about such properties with physical realizations of complex systems, these properties can often be efficiently considered using models of the underlying system [15]. If a model is a good abstraction of the system, then this gives system designers corresponding confidence in the real-world system [15]. The process of describing a system and its desired properties in a precise manner is known as *Specification*, and design time assurance techniques deal with demonstrating that a given model satisfies its specifications.

II.2.1 Model Checking

Model checking is a technique that deals with building a finite model of a system and checking that a desired property holds [21]. Generally, this involves an exhaustive exploration of the system's behavior through algorithmic analyses of its executions [22]. In this realm, the model must be defined using an appropriate formalism that defines its execution in the context of its operating environment. The space of the possible states that the system may encounter is usually described using mathematical structures such as sets, relations, and other functions, and sufficient care must be paid to capture all the relevant information while abstracting unnecessary details [23]. The challenge here is designing algorithms that allow for efficient explorations of potentially large state spaces so that the relevant specifications can be adequately assessed. In many cases, as the size of a system increases, the space over which we are required to reason increases exponentially. This is known as the *state explosion problem*, and it has been well studied within the formal methods literature [24]. Correspondingly, there is a large body of work towards developing abstractions and algorithms aimed at addressing this challenge [25, 26, 27, 28, 29, 30]. The classic product of model checking consists of a mathematical proof that a system satisfies a particular property, and if the property is not satisfied then many approaches yield *counter-examples*, which are demonstrations of undesirable behavior [21]. The quality of the results obtained through model checking is largely governed by the quality of the underlying abstraction of the system, as well as the specifications used to evaluate its correctness [31]. To demonstrate that a model is a good representation of the system, it is imperative to carry out rigorous validation efforts that explore how well a model adheres to experimental data [32]. Additionally, to be useful, specifications must be high-quality in nature in order to realistically reveal inconsistencies, ambiguities and incomplete system designs

[33, 21]. In many cases, the set of specifications that can be considered is limited, since some questions may be computationally intractable or undecidable [25]. Especially within the realm of CPS. Thus, there has been a significant amount of work aimed at addressing these challenges. In recent years, model checking has displayed great efficacy in being used in numerous domains, and its broader impact has transitioned from being limited to academic circles to being utilized within industrial applications. In particular, it has been used extensively in industry for hardware verification [34].

Within the context of CPS, model checking has been used extensively [35, 36]. However, there are unique challenges that arise in considering models of CPS. Particularly with respect to time. For traditional software, time is often an issue of performance, whereas for CPS, time is an issue of correctness [15]. In interacting with the physical world, the time it takes to perform a task may be critical to ensuring safety [37, 38, 39]. As an example, consider the case of airbag deployment. In this scenario, slight variations in the time it takes to deploy an airbag could be the difference between life and death for a passenger [40]. Thus, model checking for CPS has required the development of unique domain specific languages and modeling formalisms capable of capturing these timing requirements [15, 23]. One such formalism that has enjoyed widespread use is the *Hybrid Systems* Framework [41, 42]. This framework has proved to be effective in capturing the heterogeneity and complexity exhibited by CPS applications.

Hybrid systems involve viewing CPS as a set of interacting pairs, one that is defined by discrete events, and the other by laws of physics. One standard modeling formalism for capturing this interaction is the Hybrid Automata (HA) framework, and it has been used to describe systems in many domains such as avionics, automotive control, robotics, process control, and embedded devices [43, 44, 45]. In this realm, the idea is that a CPS can be modeled as a finite-state transition system with discrete modes whose underlying states are governed by real-valued continuous functions describing the evolution of states. The system's physical dynamics are primarily modeled through the use of ordinary differential equations (ODEs), differential algebraic equations (DAES), and differential inclusions [45, 46]. Recently, there have been extensions of hybrid systems to include more complex dynamics described by partial differential equations (PDEs). HA have proven to be an effective and powerful formalism for enabling analyses of both temporal and static requirements for CPS [43].

II.2.2 Reachability Analysis

One common way of demonstrating that a system satisfies relevant safety properties is through the use of *Reachability Analysis*, and many model checking algorithms rely on this technique [47]. Reachability analysis involves computing the set of all states that a system can attain over a finite time horizon [39]. As Asarin et al. note, such an analysis "provides knowledge about the system with a completeness or coverage that a

finite number of simulations cannot deliver," [39]. Primarily because the reachable set describes the system's trajectories from all possible initial conditions, and under all admissible disturbances and variations in parameters values of the underlying model [39]. In deriving such a set, the safety assurance problem consists of determining whether there is an intersection between the reachable set of a system and a set of *undesirable states*. As an example, for an autonomous vehicle this analysis can be leveraged to investigate whether the vehicle remains within lane boundaries, and if static and dynamic obstacles are avoided as the vehicle navigates its environment [48].

There is a rich set of literature and software tools available for the reachability analysis of systems with continuous, discrete, and hybrid dynamics [39]. While we confine our focus to those with continuous and hybrid dynamics in this document, the reachability analysis of discrete systems has been extensively considered since the early 1960s and is a well studied problem [49, 50]. Generating the set of reachable states involves a combination of numerical analysis techniques, graph algorithms, and computational geometry [51, 39]. For systems described with piecewise constant dynamics, it is possible to compute the exact set of reachable states [52]. In contrast, for systems with non-trivial continuous dynamics, obtaining the exact reachable set is often extremely difficult or undecidable. In fact, even for linear systems, obtaining the exact reachable set is only possible if the matrices that describe the differential equations possess a specific eigen-structure [39]. Such a structure is outlined in [53]. Thus, deriving the reachable set for these classes of systems involves obtaining an approximation of this set using a variety of set representations.

There have been numerous formalisms and representations proposed for approximating reachable sets, and many of these approaches are supported by software tools for experimental validation [54]. These include polyhedra [55], level sets [56], star-sets [57], intervals [58, 59], zonotopes [52], taylor models [60], and barrier certificates [47]. A common approach utilized by these approaches is *flow-pipe* or *reachtube* construction. Flow-pipe construction deals with obtaining snapshots of the set of reachable set of states enumerated at successive points in time. It is frequently used for systems governed by linear or non-linear continuous dynamics. In this context, the flow-pipe represents a characterization of the system's evolution over a fixed time horizon. Deriving flow-pipes has been done using optimal control theory, numerical approximation techniques, and even aggregations of a finite number of individual trajectories [47]. As with other model checking techniques, the computational cost associated with computing flow-pipes varies exponentially with an increase in the number of dimensions of a system. Thus, there is a tradeoff between the accuracy of the underlying approximation of the reachable set and the time it takes to obtain such an approximation. The main challenge, in this context, is to obtain over-approximations that are granular enough to be useful for the analysis being carried out by a system designer [47].

Beyond safety assurance, reachability regimes have been used to study invariance properties, robust con-

trol, set-based fault detection and monitoring, controller synthesis, conformance checking, and constraint satisfiability [47]. An in depth summary of these topics can be found in the following surveys [47, 21, 51, 37].

II.2.3 Testing and Simulation

Simulation and testing are two essential activities within model-based design, and these approaches are widely used within industrial applications [47]. While these approaches suffer from the limitation that they do not exhaustively cover the space of all possible scenarios that a system could encounter, they can provide for incredibly useful analyses of the correctness of a system [61, 48]. Particularly, when formal verification is not possible, ill-defined, or too costly to perform [62]. Moreover, testing allows engineers to understand the dynamics of a system and assess the fidelity of the abstractions used in their modeling. There is a long and established history of efficient software-engineering testing methods such as unit testing, integration testing, path testing, branch testing, and network testing to name a few [63]. In recent years, there have been extensions of these approaches to CPS [64, 65, 66]. For CPS, this requires reasoning about hardware as well as the integration of multiple components that form a cohesive whole [67].

Many approaches utilize model based testing strategies, in which an executable formalism enables the execution of numerous simulations in the hopes of uncovering errant behavior [68]. Other approaches do not assume an underlying model and rather inspect the correctness of the inputs and outputs of the system. The level of detail with which models are described, as well as the amount of simulations conducted, depends on the system testing goals [67]. One challenge regarding CPS testing is that in order to provide meaningful results, tests must consider different models of the system's environment as well as the other components utilized within the system in tandem. To address this challenge, co-simulation methods where integrated simulations can be run simultaneously in a cooperative manner have been proposed [69].

Simulation analyses have also been extensively leveraged to determine if an underlying model of a system accurately matches experimental data. This process is known as *model validation*, and there is a wealth of literature aimed at this idea [32]. One such approach is the Monte Carlo paradigm, where a large exploration of the possible parameter space for models can be used to engender statistical confidence in the fidelity of a model [32]. Often the distance between a model and the system it represents is measured by a term known as *conformance degree*. Intuitively, this is the distance between a model's executions and experimental data [32]. Model validation is an extremely important aspect of safety assurance, and an in depth discussion of these approaches can be found in [70].

II.2.4 Fault-Tolerance

Formal methods offer a powerful design paradigm for reasoning about the correctness of systems. However, they do not offer a complete solution to the development of safe autonomous systems [71]. Particularly within the context of software, "we often require that safety-critical software be highly correct. However, this misses a key point. First, software can fail frequently but still not lead to unsafe behavior if the failures do not cause hazardous consequences. Second, reliable software can be unsafe - if in the rare event of failures there are catastrophic consequences" [71].

Fault-tolerance is the discipline of designing systems that continue to provide required functionality in the presence of faults [72]. Throughout the fault-tolerance literature, the terms "fault", "failure" and "error" have distinct meanings. "An *error* is that part of the system which is liable to lead to subsequent *failure*. While a *fault* is the adjudged or hypothesized cause of an error" [72, 73]. To provide required functionality at all times, fault-tolerance techniques combine both system-level architectural considerations and software design considerations. Wilfredo et al. outline that there are four ways of dealing with faults: removal, prevention, tolerance and reconfiguration [72]. Fault removal and prevention pre-suppose verification and validation, while tolerance and reconfiguration deal with understanding the probabilities of failures and subsequently designing mitigation strategies [72]. There are a variety of techniques proposed for these challenges such as fault containment, redundancy, error detection, n-version programming, and error handling [74].

Designing fault-tolerant systems requires that an understanding of the failure modes of the system is available and that estimates of those failures be determined. One common approach for obtaining these estimates is through failure mode and effects analysis (FEMA), fault-tree analysis, and hazard analysis [75]. These efforts allow system designers to think about the probability of failure, or said another way, the system's overall reliability. With those estimates, the task is to design a system such that failures in one component of a system do not cascade, and that mitigating strategies are enacted at appropriate times [74]. A good summary of these approaches can be found in [72, 73]. While many of these analyses can be done at design time, many approaches also rely on the online monitoring of system specifications during operation. Online monitoring and verification approaches are the topic of the subsequent section.

II.3 Runtime-Time Assurance Techniques

II.3.1 Runtime Verification and Assurance

Autonomous CPS are often tasked with operating in uncertain and dynamic environments, where they must appropriately handle complex interactions with other environmental participants. In such contexts, verification and validation results obtained with respect to the system's model at design-time may only be partially transferable to the system's behavior at runtime [76]. As an example, for an autonomous vehicle, it is imper-

ative that collisions are avoided at all times. This requires monitoring the vehicle's state during operation, as design time considerations cannot feasibly consider all the possible scenarios that a vehicle may encounter [61]. The response to this challenge have been the rise of *runtime assurance* (RTA), and *runtime-verification* (RV) approaches. While there is no widespread agreement about the various meanings of these terminologies in the literature, an intuitive explanation of these regimes can be summarized as follows [77]. While RTA techniques may often utilize verification results, they often also employ statistical techniques such as anomaly detection or simulation based strategies that may not possess rigid guarantees [29, 78]. RV techniques, on the other hand, deal with lightweight yet rigorous considerations of presupposed formal properties at runtime [79, 80, 81, 82]. However, the essence of both approaches is that they deal with the creation of monitors that discern whether assumptions made at design time are also satisfied during operation.

RTA and RV techniques are widely used within industry and academic settings. The systems considered by these approaches include software systems, hardware applications, and CPS [83, 84, 85, 86]. The nature of properties that are considered by these approaches can be in terms of desired or undesired behavior (known as falsification). While falsification was not discussed in our treatment of design-time approaches, there are corresponding methods in that realm as well. As is the case with design time approaches, relevant system properties are expressed using formal specification languages and formalisms, but here the focus is on sequences of state observations. These sequences of observation are known as *traces*, and they may be discrete or continuous, depending on the nature of the application [77]. Leucker et al. note that one of the distinguishing features of runtime verification is that it opens up the possibility of performing mitigation strategies whenever incorrect or unsafe behavior is detected [87]

Falcone et al. note that there are four major steps involved in the RTA/RV process [88]. The first step is the creation of monitors from formal properties. The second step involves system instrumentation, which is the process of outlining the relevant events to be provided to the monitor. The third step involves the execution of the system with the relevant observations passed to the monitor. Finally, the monitor produces determinations about these observations and may potentially also provide feedback to the system in terms of corrective actions that must be taken [88]. RTA/RV approaches span several research communities. In the sections that follow, we outline key areas relevant to the safety assurance of CPS, and we refer interested readers to the following surveys for a more detailed treatment of this exciting realm of research [88, 87, 89, 77].

II.3.2 Online Monitoring via Temporal Logics

The most common family of specification languages used within RTA/RV techniques are temporal logics [77]. Temporal logics are a system of reasoning about properties qualified in terms of time. The most basic and usual variant is linear temporal logic (LTL) which allows for the application of modal operators

such as “Next”, “Until”, “Always”, and “Eventually” to be applied to analyses of a system’s traces [88]. Such analyses are often conducted by tools that are direct extensions of model checking algorithms, and there is a rich set of approaches for dealing with temporal specifications [90]. LTL specifications often necessitate reasoning about infinite length traces, however in the context of runtime verification, we are typically concerned with finite traces. Consequently, there have been extensions to LTL to reason about finite length traces [77]. Moreover, there have been extensions to LTL that allow for the consideration of a system’s past behavior [89].

Beyond LTL, several other temporal logic formals have been proposed such as interval temporal logic (ITL), metric temporal logic (MTL), signal temporal logic (STL), and spatial-temporal logics (STTL) [90, 88]. ITL allows for the analysis of traces over intervals of time rather than distinct points. This allows for the investigation of questions such as whether sequences of events overlap, or if the start and end of events are ordered [77]. MTL approaches extend LTL notions of ordered sequences of events to a quantitative consideration of where these events occur on the timeline. Specifically, in MTL, modal operators are associated with discrete or continuous time intervals. Signal temporal logic expands the domain of LTL techniques to consider traces that are not sequences of discrete events, but rather collections of real-valued continuous signals [77]. Finally, Spatial Temporal Logics allow for the consideration of systems that are spatially distributed in nature, or for properties that require the consideration of spatio-temporal properties. There have also been extensions to these standard logics to include the consideration of properties related to sets of traces. Such properties are known as hyperproperties and several methods have been proposed for their analysis [77]. We refer readers to the following papers for an in depth consideration of the nuances of these logics [91, 88, 89, 92]

II.3.3 Online Reachability

Given the power of design-time model checking through reachability analysis, researchers have begun to develop extensions of these approaches that are amenable to runtime operation [92]. In addition to set-based reachability regimes, approaches such as viability kernels that determine if a set of states remain within a predefined region [93, 48], as well as optimization-based Hamilton-Jacobi (HJI) reachability techniques have been proposed. These approaches have demonstrated an ability to deal with systems with a wide range of dynamics and disturbances within the context of dynamic and uncertain environments [94, 95, 96, 97, 98, 99, 100, 101, 102, 103]. The majority of these approaches deal with the reachability problem for a single system. However, approaches that can handle multiple agents have also been proposed [104].

Within the context of autonomous CPS, online reachability techniques have been used extensively for safe motion planning and have been validated extensively on both simulation and hardware platforms [96, 48].

One of the ways in which this has been done is by incorporating reachability regimes into model predictive control regimes. In this context, the results of reachability analysis can be leveraged to solve an optimal control problem over a finite time horizon [48]. While this has displayed efficacy for systems with linear dynamics, the problem is much more difficult for non-linear systems [48]. While methods for non-linear systems do exist, they are often too computationally expensive to be used online [105].

II.3.4 The Simplex Architecture

In some cases, the individual components utilized within CPS may be too large or too complex for formal analysis. However, the behaviour of the overall system relies on obtaining guarantees about these components [106]. One of the most popular paradigms for assuring systems with unverified components is the *Simplex Architecture*. In this framework, an unverified component is wrapped with a safety controller and a corresponding switching logic designed to transfer control to the safety controller in certain situations [107, 108]. A useful analogy for this architecture is a driving instructor's car with two steering wheels and two sets of brakes. As long as the instructor is capable of intervening in dangerous situations, the capricious student is allowed to drive. Typically, the complex controller has better performance with respect to the design metrics, whereas the safety controller is designed with simplicity and verifiability in mind [109].

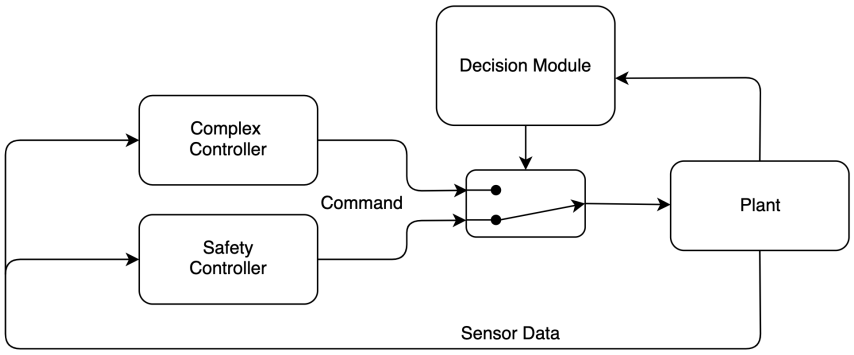

Figure II.2: Overview of the Traditional Simplex Architecture

Typically, in simplex architectures, the switching logic is primarily designed either from a control theoretic perspective through the solution of linear matrix inequalities (LMI) [110], or using a formal analysis hybrid-systems reachability technique [13]. As Bak et al. note, it is easy to design a safe decision logic; one can simply always use the safety controller [108]. However, this is unsatisfactory since the performance responsibilities of the system might be forfeited or unreasonably delayed [108]. The key challenge in this regime is to design a switching logic that allows the dynamic capabilities of the unverified complex controller to be exploited as much as possible without compromising safety. Thus, the central focus is the design of verified decision logic that is minimally conservative. Numerous effective approaches have been proposed in

recent years, resulting in the simplex architecture being utilized within numerous contexts. These contexts include aerospace systems [110], fleets of remote controlled cars [111], industrial embedded infrastructure [13, 28], and distributed mobile robotics applications [112, 104].

II.4 Verification of Machine Learning Components

Machine learning has been lauded as one of the most influential and disruptive set of methodologies of our era. These methods are now at levels of accuracy to be competitive with human levels of performance at many tasks [113]. Consequently, there has been a large push to use them within autonomous CPS for sensing, actuation, and control. Much of the success with this realm can be attributed to recent advancements in *deep learning*, where neural networks have revolutionized how we approach complex problems [114]. The challenge however remains that traditional formal analyses struggle to cope with the complexity of these models, since they are often characterized by millions or even billions of parameters. Due to the promise exhibited by these models, in recent years we have witnessed numerous promising verification methods proposed towards reasoning about the correctness of their behavior [114, 9].

Before outlining the techniques, let us first consider some preliminaries. Neural Networks consist of a number of interconnected neurons, where each neuron can be perceived as a processing element that reacts to the weighted sum of the inputs it receives. The neurons are typically structured into three types of layers: an input layer, an output layer, and one or multiple hidden layers. Each connection between neurons is typically labeled with a real-valued weight that is determined during a training process that seeks to maximize the network's prediction accuracy [115]. The overall structure of the network varies greatly depending on the specific architecture being considered. There are numerous architectures such as recurrent networks [116], radial basis function networks [117], long-term short-memory networks [118], self-organizing maps [119], feed-forward [120] and convolutional neural networks [121]. An example feed-forward architecture is shown in Figure II.3.

II.4.1 Neural Network Verification

Neural networks are often criticized as lacking transparency, since their underlying operation is often incomprehensible. This makes general observations about their behavior meaningless. One has to look no further than the rich literature of *adversarial* machine learning to get a sense of the unexpected and errant ways in which these models fail [122]. Traditionally, analyses of neural networks have largely focused on evaluating the network on large collections of points in the input space and assessing whether the outputs correspond to expected values [114]. Characteristic techniques include: *test coverage methods*, and *concolic testing*. Test coverage methods, which draw inspiration from software engineering, are white-box testing methodologies

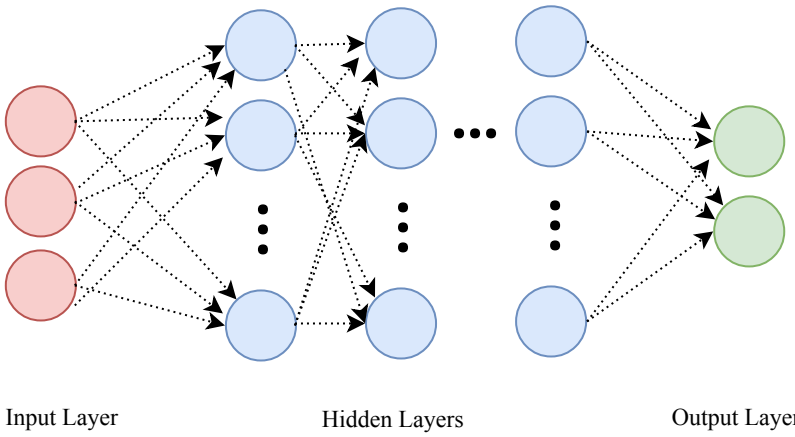

Figure II.3: Simple Feed-forward Architecture

aimed at generating exhaustive test cases to gauge the correctness of a neural network's behavior [123]. For large input spaces, however, testing can become infeasible.

Verification efforts aim to discern whether particular properties about a neural network's behavior hold over large regions of the input space that it is tasked with handling. Intuitively, this process corresponds to showing that the output of a network the desired property for every choice of input within a bounded set [124]. An example of a property that is frequently considered within classification settings, is that all the points that are similar to a particular training example, as measured by some metric, should belong to the same class as that example. Classically, the verification problem is posed as a search for an input that causes the negation of a desired property to be true. If the search is successful, then we conclude that the property does not hold and the obtained solution serves as a counter-example demonstrating a property violation [125]. Otherwise, the search fails, and we conclude that the underlying property holds. From an analysis standpoint, it has been demonstrated that neural network verification is an NP-complete problem [12]. Thus, one of the main challenges in this realm is designing algorithms that can scale to the complexity of neural networks used in state-of-the-art settings.

Challenges around scalability have led to the rise of two classes of verification methods; *Sound* methods and *Complete* methods. Soundness is a feature of the majority of approaches and denotes that an algorithm will only report that a property holds, if it actually holds (no false positives) [114]. Completeness on the other hand refers to the ability of an algorithm to state that a property holds when the underlying property is indeed true [114]. To promote computational efficiency, some algorithms sacrifice completeness through the use of approximations. Thus, it is possible under these regimes for the algorithm to report that it is unknown whether a particular property holds for the specific verification problem being considered and that further analysis must be conducted [125]. However, in practice, these methods have demonstrated great efficacy in

numerous contexts. In contrast to incomplete methods, Katz et al. assert that techniques which are sound and complete are typically limited in terms of scalability [125].

Bearing the above in mind, the majority of verification efforts can be broadly classified into three main categories. Those that treat the verification problem as a reachability problem [57, 126, 127, 128, 129, 130], methods that make use of optimization [131, 132, 133, 134, 135, 136, 137], and methods that make use of search [138, 139, 140, 141]. This realm of research has received significant attention in recent years, and we refer readers to the following surveys for a more detailed treatment of the intellectual progression of this field [114, 113, 9]. The aforementioned survey by Liu et al. offers a particularly enlightening analysis of the main drawbacks and limitations of these approaches.

II.4.2 Closed Loop Neural Network Verification

Beyond techniques that reason about neural networks in isolation, in recent years several methods have been proposed for verifying neural network control systems (NNCS) [126, 142, 143, 144, 145]. Neural network control systems commonly appear in settings where a neural network controller is obtained through the use of reinforcement learning, behavioral cloning, or learning by demonstration to accomplish a complex task [142]. In this realm, the safety verification problem is often posed as a reachability problem, where the chief concern is to ascertain whether a neural network controller will cause the system to enter into an unsafe scenario. Thus, the task is to obtain the set of reachable states for the combined system, which consists of a physical system (plant) and a neural network controller [142]. Despite significant progress in neural network control systems verification, developing scalable techniques remains a key challenge.

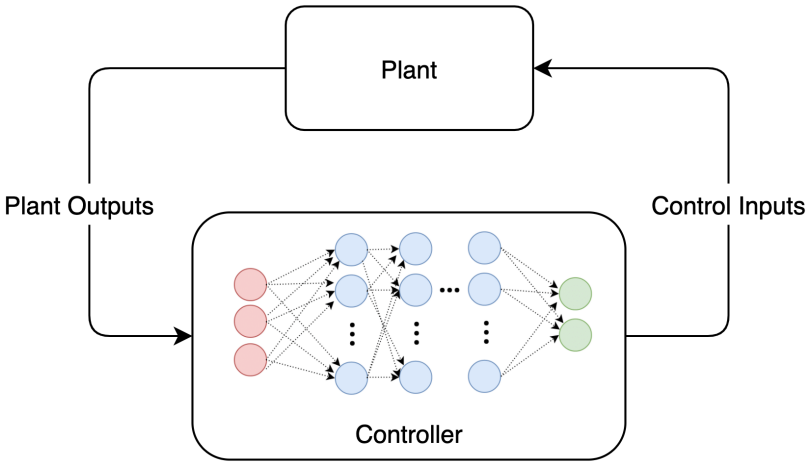

Figure II.4: Illustration of a closed-loop system with a neural network controller.

Apart from scalability concerns, as Dutta et al. note, generating reachsets for NNCS involves more than a straightforward combination of existing reachability tools, and neural network verification schemes due

to the well known *wrapping effect* [142]. The wrapping effect is a phenomenon in which the error derived from representing the set of reachable states through various formalisms accumulates as the reachability computations are carried out. This can lead to either extremely conservative approximations, or even result in the error blowing up [146]. Several approaches have been proposed that circumvent this issue, such as rule generation, mixed-integer programming techniques, and interval analysis. A nice summary of these approaches can be found in the category reports of the Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid System Plants friendly competitions [147, 148].

II.4.3 Safe Reinforcement Learning

The safety problem for reinforcement learning approaches has also received significant attention in recent years. Primarily because RL has demonstrated versatility in solving complex problems within multiagent robotics, swarm intelligence, optimal control theory, and game theory [83, 98]. One of the most attractive aspects of RL approaches are their ability to produce optimal results with simplistic models of their environment and the system that they control [98]. Moreover, by programming agents via punishments and rewards, these approaches allow for the formation of intelligent behavior without needing to specify how the task is to be achieved [149]. However, these benefits come at the cost of interpretability, making correctness guarantees hard to come by. As a result, this has led to the rise of *safe-reinforcement learning*, which refers to the process of learning behaviors that maximize performance while respecting safety constraints during learning and deployment [150, 151].

Apart from the challenges arising from the opaque nature of RL approaches, safely obtaining optimal policies is equally difficult. Often RL agents are expected to learn from trial and error, exploring *any* behavior during the learning process. In many real-world settings, this level of freedom is unacceptable. Consider for example an expensive robotic platform such as an autonomous vehicle. One must not only seek reward maximization, but also avoid damage to the system during the learning process [152]. This mandates ensuring that the system can recover safely from unsafe actions. Consequently, many state-of-the-art RL approaches avoid such considerations by training agents in simulation and subsequently transferring learned policies into the real world. However, this transfer presents incurs its own unique challenges. Thus, we can consider the safety assurance problem from two perspectives: the first perspective would be to assure RL agents at runtime, while the second would be to guarantee safety during learning. It is primarily this latter perspective that the field of Safe RL embodies [153].

Safe exploration methods for RL have often made use of runtime monitoring formalisms such as temporal logics to detect unsafe actions [154, 155]. Other approaches have altered the optimization criteria utilized by RL agents to incorporate notions of risk in long-term reward maximization [156, 157, 158, 159, 160]. To

do so, many approaches design heuristics to identify safe regions of the reward space that are updated as the agent learns more about the environment [152, 161, 162, 163]. In some cases, the exploration process can be modified by including prior knowledge of the environment and acceptable behaviors for the underlying task [164, 165]. The challenge however is balancing the desire for safe exploration with a desire to maximize the set of possible actions that can be undertaken in order to synthesize an optimal policy. Many efficient approaches exist within this space, and García et al. provide an insightful consideration of these approaches in the following survey [152].

II.5 Real Time Systems

Finally, Baskiyar et al. define a real-time system as "one whose correctness involves both the logical correctness of its outputs and their timeliness," [166]. These systems are often classified into three distinct classes with respect to timing constraints: hard, firm, and soft. In hard real-time systems, failure to meet timing requirements can result in system failure. Firm real-time systems have strict timing deadlines, but allow for small probabilities of timing failures. Finally, soft-real time systems view timing constraints in terms of performance. Failure to meet timing deadlines in this context results in degraded performance, but often will not result in system failure [166].

The role of time in CPS plays a major role in the consideration of its design and requirements, and many systems such as automobiles have strict timing deadlines [167]. This often requires the consideration of unique software and hardware specifications, and often mandates the use of real-time operating systems whose aim is to schedule tasks in a fashion that satisfies strict timing requirements. While scheduling approaches are outside the scope of this document, there is a rich literature of real-time scheduling techniques such round-robin scheduling [168], preemptive scheduling [169, 170], and rate monotonic scheduling [171] that ensure that timing criteria for software processes are fulfilled [172]. For soft-real time systems, however, it may be sufficient to conduct rigorous analyses of the statistical distributions of execution times [167].

Several paradigms for the creation of real-time CPS have been proposed in the research literature. Due to their combination of physical, networking, and software components, this requires reasoning about execution paradigms, faults, and conducting rigorous determinations of the worst-case execution time of tasks [173]. In this realm, the majority of real-time system design methods utilize over-approximations of the worst-case execution time of tasks within a CPS to ensure that deadlines are always satisfied. Moreover, many approaches forecast task execution times and resource needs within these systems in an effort to enable more predictable operation at runtime [173]. A nice summary of these techniques can be found in [167].

II.6 Summary

There are few technologies that hold as much promise as autonomous CPS in re-orienting the way we move around, explore new environments, distribute resources, and conduct complex missions. To bring forth these benefits, we must ensure that CPS meet rigorous standards of correctness both at design time and during operation. This mandates the development of modeling tools and algorithms that can deal with the complexity exhibited by CPS and their environments. Since CPS generally operate in unstructured and dynamic environments, their design often incorporates opaque data-driven or machine learning methods. Thus, assurance tools must be able to provide guarantees for these methods as well. It is within this context that we propose our work.

CHAPTER III

Zero-Shot Policy Transfer for Machine Learning Controllers in Autonomous Racing: Reinforcement Learning vs Imitation Learning

This chapter is adapted from the material presented in [17].

There are few technologies that hold as much promise in achieving safe, accessible, and convenient transportation as autonomous vehicles. However, as recent years have demonstrated, safety and reliability remain the most obstinate challenges, especially in complex domains. Autonomous racing has demonstrated unique benefits in that researchers can conduct research in controlled environments, allowing for experimentation with approaches that are too risky to evaluate on public roads. In this chapter, we compare two leading methods for training neural network controllers, Reinforcement Learning and Imitation Learning, for the autonomous racing task. We compare their viability by analyzing their performance and safety when deployed in novel scenarios outside their training via zero-shot policy transfer. Our evaluation is made up of numerous experiments in simulation and on our real-world hardware platform that analyze whether these algorithms remain effective when transferred to the real-world. Our results show reinforcement learning outperforms imitation learning in most scenarios. However, the increased performance comes at the cost of reduced safety. Thus, both methods are effective under different criteria.

III.1 Introduction

Autonomous Racing is a growing topic of interest, ranging from small-scale academic competitions (e.g. F1/10 [174]) to full-scale competitions (e.g. Roborace, AWS DeepRacer[175] and the Indy Autonomous Challenge[176]). These racing competitions are integral to the development of *Autonomous Vehicles* (AVs) as they help promote general confidence and societal acceptance of a novel emerging technology. Moreover, they allow researchers to conduct explorations of possible solutions to difficult scenarios such as high-speed obstacle avoidance and other risky maneuvers that may be too dangerous to consider in urban settings [2].

Within this realm, one classical approach of constructing these systems involves a decomposition of tasks into four main areas: perception, planning, control, and system supervision [177]. Confining our focus to the control of these vehicles, many platforms favor classical or model predictive control techniques for their predictably safe performance. However, in recent years, many researchers have proposed the use of machine learning for control tasks, as these methods have shown significant potential in solving optimal

control problems for highly nonlinear systems with varying degrees of uncertainty [177]. This prowess has made these types of regimes particularly attractive for autonomous vehicle development.

One of the most successful frameworks for solving machine learning control problems has been *Reinforcement Learning* (RL). RL is a branch of machine learning that focuses on software agents learning to maximize rewards in an environment through experience. The general idea is similar to training a dog to do tricks by giving it treats when it performs the desired task. Thus, an optimal controller can be synthesized using data evaluated by key performance criteria through trial and error[178]. Many RL approaches leverage neural networks due to their advantages in dealing with complex data. These approaches can be referred to as *Deep Reinforcement Learning* (also referred to as RL) techniques, and recent successes such as OpenAI's *OpenAI Five* outperforming pro-level players at Dota 2[179], and Microsoft's *MuZero*[180] mastering Atari, Go, Chess and Shogi have helped bring RL to the forefront of AI discussion.

Despite their success in numerous realms, RL approaches, can be costly to train, especially as systems become more complex and dynamic. Additionally, RL allows agents to learn via trial and error, exploring *any behavior* during the learning process. In many realistic domains, this level of freedom is unacceptable, thus training in simulation is standard. Therefore, the challenge becomes how to minimize the inherent mismatches between real-world settings, and the simulation environments used to train RL agents [181].

Training agents in simulation and then deploying them on real-world hardware platforms, known as a *sim2real* transfer, is a challenging problem. In many cases, the agents do not perform as expected in the real world, sometimes resulting in unsafe or catastrophic behavior [182, 183]. Their performance can be improved with further training in the new environment, but that is only possible if the behavior policy is safe from the outset. Transferring a learned policy and evaluating before any additional training is done is referred to as a *zero-shot policy transfer*.

In this chapter we focus on zero-shot policy transfer since active learning, i.e. learning during evaluation, is impractical for real-time systems because updates to the neural network control policy are computationally expensive and time-consuming. Instead, we evaluate trained policy networks as they are. This is standard practice in industry, to deploy a trained model and release updates intermittently.[1]

One way to achieve high performance with a zero-shot policy transfer is by leveraging external or expert knowledge. *Imitation Learning* (IL) utilizes expert demonstrations to train an agent to mimic a given behavior. Using IL, an agent can be trained to mimic a human or a complicated array of computationally intensive classical control methods that perform optimally in different scenarios. In this way, complicated algorithms and/or human experience can be boiled down to one neural network capable of replicating their behaviors.

While the last several years have witnessed a significant number of approaches for addressing these chal-

[1]The rate at which these updates occur depends highly on the application.

lenges, there have been few in-depth empirical studies comparing the efficacy of different learning frameworks for learning robust agent behavior [184]. In [184], Gros et al. note that RL approaches generally outperform IL. However, this performance comes at a cost of significant reward shaping. While this work provides an enlightening discussion, the authors consider only discrete environments and do not address *sim2real* challenges.

In light of the lack of empirical comparisons of IL and RL, in this work, we experiment with and compare *Neural Network Controllers* (NNCs) trained using these approaches for the control of a 1/10 scale autonomous vehicle. The performance of these trained NNCs are compared through a number of experiments, testing their ability to handle scenarios outside their training environment via zero-shot policy transfer. These experiments include changing the vehicle's constant speed, adding unknown obstacles to the track, and evaluating on different tracks. These experiments culminate in a *sim2real* transfer and evaluation of the controllers on our hardware platform.

III.2 Background

III.2.1 Imitation Learning

Imitation learning seeks to replicate the behavior of a human or other expert on a given task [185]. These approaches fall within the field of *Expert Systems* in Artificial Intelligence, and in recent years the demand for these approaches has increased substantially. The surge in interest is spurred on by two main motivations. (1) The number of possible actions needed to execute a complex task is too large to cover by explicit programming. (2) Demonstrations show that having prior knowledge provided by an expert is more efficient than learning from scratch [185].

In this work, we employ one of the most common methods of IL is *Behavior Cloning*, which was first introduced to train a modified van to navigate paths at speeds up to 20 miles per hour [186, 120]. The work was later replicated with an updated convolutional neural network architecture in [187] with great success.

III.2.2 Reinforcement Learning

Reinforcement learning seeks to find the optimal behavior function for completing a given task through experimental trials. An agent converges on this optimal behavior function, or learned policy $a = \pi(s)$, by learning what results from executing action a when in state s. The result is the next state, s', and a reward, r, determined by a given reward function. This information is stored as a tuple, $\{s, a, r, s'\}$, often referred to as an *experience*. In this work we utilize two well-known, state-of-the-art off-policy deep reinforcement learning algorithms *Soft Actor-Critic* (SAC)[188], and its predecessor *Deep Deterministic Policy Gradient* (DDPG)[149].

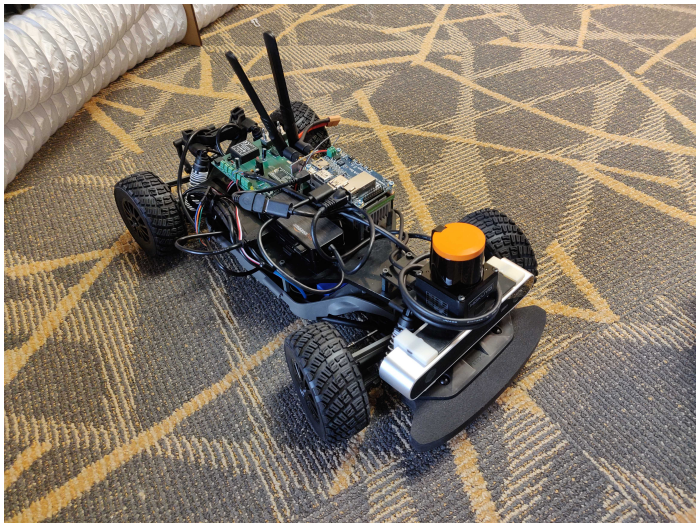

Figure III.1: Visualization of our experimental F1/10 hardware platform. This platform is a one-tenth scale RC car that has been altered to operate autonomously with the support of a sensor and compute architecture for autonomous decision-making [2].

III.2.3 F1/10

For our experiments, we utilize the F1/10 simulation and hardware platform [174]. The platform was designed to replicate the hardware and software capabilities of full scale autonomous vehicles. The hardware platform is equipped with a standard suite of sensors including stereo cameras, LiDAR (light detection and ranging), and inertial measurement units (IMU). The car is controlled by an NVIDIA Jetson TX2, and its software stack is built on the *Robot Operating System* (ROS) [189]. In the Gazebo simulation environment, all the sensors are replicated, so the transition from simulation to the real-world and back is straightforward without hours worth of re-configuring.

III.3 Experimental Setup

In order to make the comparisons as fair as possible, all the controllers we trained have the same neural network architecture and are trained on the *Porto* track shown in III.2 unless otherwise specified. The trained NNCs selected for our experimental evaluations are the best performing of at least 3 NNCs trained the same way using different random seeds[2]. Additionally, the control output has been limited to only steering and the car travels at a constant speed of $1m/s$ during training.

[2]The random seed used for training has a large impact on the training process and resulting policy, as demonstrated in [190, 191]

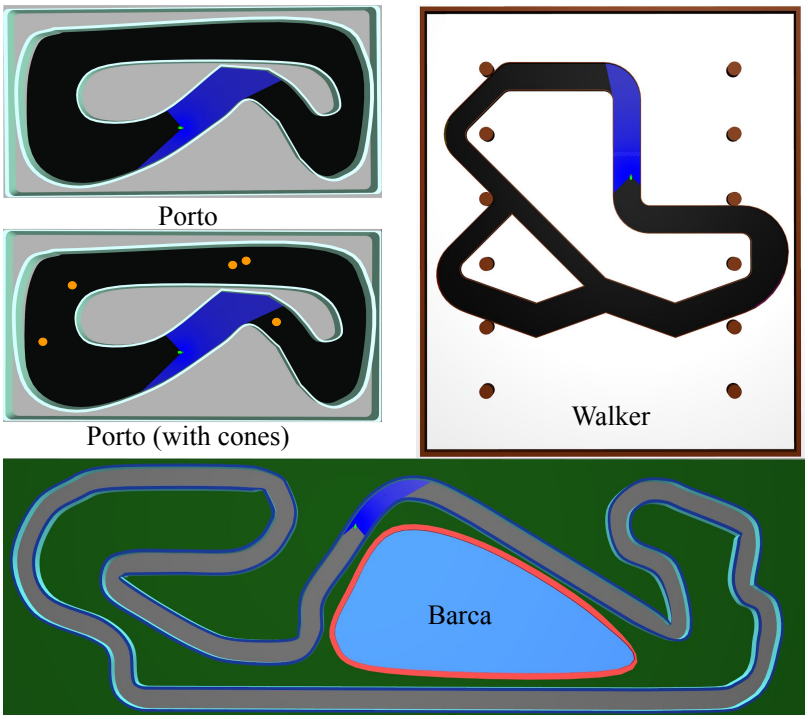

Figure III.2: The different tracks we use in our simulation experiments

III.3.1 Neural Network Architecture

In this work, we utilize a common architecture found in RL work. The simple multi-layer perceptron network consists of an input layer, 2 fully connected hidden layers of 64 nodes with ReLU activation functions, and a fully connected output layer with a tanh activation function. The input layer accepts nine range values collected from the LiDAR at $-90°$, $-60°$, $-45°$, $-30°$, $0°$, $30°$, $45°$, $60°$, and $90°$ from forward. The range values are clipped between $[0m, 10m]$. The output layer provides a single value between $[-1, 1]$, which is scaled up linearly for the desired steering angle between $[-34°, 34°]$.

III.3.2 Training the Agents

III.3.2.1 Imitation Learning

We trained the imitation learning agent using a procedure that is a simplification of the seminal work by Dean Pomerleau, in which a neural network was trained to control an autonomous vehicle [120]. The agent in this work was trained on sensor-action pairs collected during experiments where the vehicle was controlled using a path following algorithm on the racetrack. The path we used for training lead around the middle of the track, ensuring safe operation. The path following algorithm we utilized, *Pure Pursuit* [192], does a quick search for a waypoint that it can safely reach governed by a specified look-ahead horizon, it then steers the car towards that waypoint. The pure pursuit algorithm has been used in numerous contexts and has been shown

to be a robust method for efficiently and accurately following a path. This was our main motivation in using this controller.

The first imitation learning agent, IL, was trained only on data collected from the *Porto* track. This makes the training process more like what the RL agents will see, since they are also only trained on the *Porto* track. The second agent, IL-3, was trained using data collected from the *Porto* track as well as the two other tracks, *Walker* and *Barca* shown in III.2. We include IL-3 to highlight one of the main advantages of using IL to train NNCs: any recorded data of the expert can be used for training.

III.3.2.2 Deep Reinforcement Learning

Both RL controllers, DDPG and SAC, were trained using common hyperparameters, which are provided in Section III.9. The agents optimize performance according to a dense reward function that assigns a positive reward for counterclockwise progress around the track. The reward is calculated using a reference path that runs through the middle of the track. The value of the reward is the positive arc length between the previous and current closest point along the path. This reward function encourages the agent to complete as many laps as possible as quickly as possible.

We trained the agents according to their respective algorithms. We halted the training process to evaluate performance after every 500 training steps. The performance is measured by how many laps the agent can complete within 100 seconds. This is more than enough time to complete 2 laps in the training track (*Porto*). We chose 2 laps because completing 1 lap is not enough to show the controller is capable of completing multiple laps. The car always starts in the same position, but may not return to the same position at the end of the first lap. However, the starting position of laps 2+ will be about the same. Thus, if the controller is able to complete 2 laps, it is likely capable of completing any number of laps.

The evaluation is repeated up to 10 times, and training stops when the agent is able to complete at least 2 laps 10 times in a row. Once the agent is able to complete at least 2 laps 10 times, the training process is halted and control policy is saved for our experiments.

III.3.3 Evaluating Performance

We evaluate the controllers through a variety of scenarios that test their ability to maintain optimal performance in scenarios outside their training environment. These scenarios include changing the constant speed value, adding obstacles to the track, evaluating on a different track, and a real-world evaluation on our hardware platform. We compare the performance of the controllers according to three metrics we refer to as *track distance*, *efficiency*, and *safety*.

Efficiency is calculated as the distance the car travels around the track divided by the amount of time it

took to get there. Each test runs for a maximum of 60 seconds and cuts off sooner if the car collides with a wall or obstacle. We refer to this as a measure of efficiency because the distance is not measured by the direct distance the car traveled. Instead, the distance is measured in relation to the arc length of a path going through the center of the track, which we refer to as the *track distance*. The closer the car stays to following the center path, the closer the efficiency value will match the constant speed. However, if the car takes sharp turns around the corners, the efficiency will increase since the car covers the same track distance in less time.

Safety is a measure of how prone to collisions the controller is at a specific track. The safety value corresponds to the percentage of runs that ended with no collision regardless of the time or distance traveled, i.e. if safety = 100%, there were no collisions encountered in the experiments.

III.4 Experiments and Results

Our experiments were designed to test the performance of both the RL and IL controllers in challenging scenarios. The first experiment demonstrates the ideal test conditions, evaluating in the same environment the controllers were trained in. The following three experiments introduce changes to the environment that test the robustness of the learned control policies, building up towards the final experiment, deploying on the real-world hardware platform.[3]

All simulation experiments test each controller 30 times in the designated scenario. Each test lasts for a maximum of 60 seconds, stopping early in the event of a collision.[4]

III.4.1 Training Environment (Porto)

Our first experiment evaluates the performance of the controllers in the environment they were trained in. This provides a baseline that we can compare to as we test these controllers in scenarios outside their training. The results in Table III.1 show all the controllers operate safely without any recorded collisions. Additionally, the results show both RL controllers operate more efficiently and travel further than the IL controllers.

Table III.1: Performance on Porto With and Without Obstacles

	No Obstacles			Obstacles		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	121.44 ± 14.41	1.94 ± 0.25	100%	41.80 ± 0.88	1.93 ± 0.02	0%
IL-3	125.23 ± 0.31	2.01 ± 0.00	100%	40.79 ± 0.26	1.92 ± 0.02	0%
DDPG	142.74 ± 10.52	2.29 ± 0.16	100%	40.33 ± 0.37	2.23 ± 0.03	0%
SAC	144.04 ± 0.47	2.31 ± 0.01	100%	182.98 ± 2.92	2.94 ± 0.01	96.66%

[3]The hardware experiments are summarized at: **https://youtu.be/rgVb46RMMvE**

[4]A video summarizing the simulation experiments can be found at: **https://tinyurl.com/2bjwpcxs**

III.4.2 Varying Speed

In our second experiment, we explore how changing the constant speed of the car impacts performance. This subtle change tests the robustness of the controllers with respect to a change in speed. The control policies were trained with the assumption the car moves at $1.0m/s$. Moving at different speeds, especially faster than expected, might reveal unsafe behaviors. Additionally, this experiment provides some insight into how well the controllers will handle a *sim2real* transfer. Unlike in simulation, the hardware platform can experience fluctuations in speed caused by a poorly-tuned speed regulator, wheel slippage, etc.

Table III.2: Performance on Porto Varying Constant Speed

	0.5 m/s			1.0 m/s			1.5 m/s		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	64.73 ± 0.36	1.04 ± 0.01	100%	121.44 ± 14.41	1.94 ± 0.25	100%	171.24 ± 41.24	2.90 ± 0.06	93.33%
IL-3	64.60 ± 0.09	1.04 ± 0.00	100%	125.23 ± 0.31	2.01 ± 0.00	100%	178.54 ± 20.94	2.92 ± 0.03	96.67%
DDPG	73.60 ± 0.22	1.18 ± 0.00	100%	142.74 ± 10.52	2.29 ± 0.16	100%	18.09 ± 0.29	2.57 ± 0.08	0%
SAC	72.97 ± 10.05	1.20 ± 0.03	93.33%	144.04 ± 0.47	2.31 ± 0.01	100%	174.11 ± 62.58	3.21 ± 0.15	80.0%

We tested the controllers on the *Porto* track with constant speeds $0.5m/s$ and $1.5m/s$. We expected the efficiency and track distance of the controllers to be cut in half when run at half speed. We also expected the controllers would remain safe at half speed. For the tests at a faster speed, we expected the efficiency to increase by a factor of 1.5, but experience more collisions.

The results in Table III.2 show that cutting the speed in half leads the efficiency and track distance to be reduced by about half for every controller. Since the efficiencies and recorded track distances of the controllers at $0.5m/s$ are slightly above the expected half, the controller's efficient behaviors are more impactful at slower speeds. Every controller except for SAC maintained their safe performance. In the one trial that the SAC controller collided with the wall, it was during the first left turn. The controller turned too early while driving close to the wall, resulting in a collision.

Furthermore, the results in Table III.2 show that increasing the speed reduces the safety of all the controllers. In our experiments, none of the controllers were safe for all evaluated runs. In particular, DDPG was unable to complete any runs without colliding after the first curve. However, despite the increase in collisions, all the controllers operated more efficiently. IL, IL-3, DDPG, and SAC saw a $1.5x$, 1.45, $1.12x$, and $1.39x$ increase respectively. The IL controller was the only one able to meet the $1.5x$ increase we expected to match the speed increase.

III.4.3 Obstacles

Our third experiment introduces unknown obstacles, orange traffic cones, to the *Porto* track as shown in III.2. This experiment tests the controllers beyond what they were trained to do. Not only does the controller have

to steer the car along the optimal path while avoiding the walls, there are now additional obstacles to avoid. Thus, it provides a measure of each controller's ability to mimic the driving task, rather than robust pattern matching. The IL controllers failed to generalize to this scenario, and failed to complete a single lap without a collision failing around the last cone. DDPG was similar in nature, however, it maintained its higher level of efficiency over the IL controllers. SAC was the only controller able to handle obstacles and successfully navigated the cones in 96.66% of our evaluations. Interestingly, the obstacles improved SAC's performance. The last cone on the track was positioned just right to direct the controller to steer sooner, finding a more optimal path.

III.4.4 Alternate Race Tracks (Walker and Barca)

In our fourth experiment, we examined how well the controllers perform when used on two different, more complicated tracks, *Walker* and *Barca* shown in III.2. *Walker* introduces a choice between two paths, which we anticipated would cause issues because none of the controllers, except IL-3, have experience with that scenario. We also anticipated that *Barca*'s long straightaways and sharp turns would cause more collisions for controllers trying to cut corners. The results for this experiment, and the lengths of the tracks for comparison, are shown in Table III.3

Table III.3: Performance Across Different Racetracks

	Porto (57.5m)			Walker (73.25m)			Barca (221.14m)		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	121.44 ± 14.41	1.94 ± 0.25	100%	33.50 ± 1.55	1.89 ± 0.02	0%	108.54 ± 33.26	1.96 ± 0.03	83.33%
IL-3	125.23 ± 0.31	2.01 ± 0.00	100%	123.38 ± 0.33	1.98 ± 0.01	100%	124 ± 0.29	2.00 ± 0.00	100%
DDPG	142.74 ± 10.52	2.29 ± 0.16	100%	130.08 ± 0.34	2.09 ± 0.00	100%	31.54 ± 0.03	1.80 ± 0.01	0%
SAC	144.04 ± 0.47	2.31 ± 0.01	100%	62.64 ± 38.31	2.11 ± 0.25	0%	24.27 ± 4.09	1.76 ± 0.02	0%

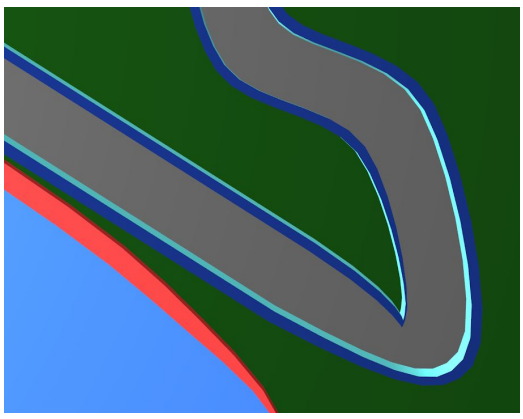

Figure III.3: Difficult sharp turn on Barca track.

On the *Walker* track, both the IL and SAC controllers were unable to consistently navigate the junction. Instead of picking a direction to pursue, the IL controller drove the car directly into the corner of the junction

in every test while the SAC controller managed to avoid that fatal mistake in some cases, but rarely passed it in the second lap. In contrast, the DDPG controller was able to successfully navigate the divergent track without prior experience. Additionally, both RL controllers navigated the track more efficiently than the IL-3 controller, which had prior experience on the track.

On the *Barca* track, only the IL controllers were able to safely navigate the sharp turn highlighted in III.3. Both DDPG and SAC collide with the track wall at the sharp turn by either turning too soon or not turning at all.

III.4.5 Real-World, Hardware Platform

In our final experiment, we test how well these controllers handle an actual *sim2real* transfer on our hardware platform. The experiments were conducted on our track, shown in III.4, which has a middle-of-the-track path length of 13.08m. Because we could not reliably record the time for runs that resulted in a collision, we do not compare the controllers' efficiency. Instead, we compare the distance traveled around the track in 60 seconds. We averaged the results across 10 runs and kept a count of how often the controllers drove the car along the side of the track, bumping into it (Bump), as well as how many times it drove directly into the side of the track (Collision). We halted the run in the event of a collision and recorded the final position as the total distance traveled. While bumps in our simulated results counted as collisions, we decided to allow them in the hardware experiments because they did not harm the track and would have been allowed in the F1/10 competition.

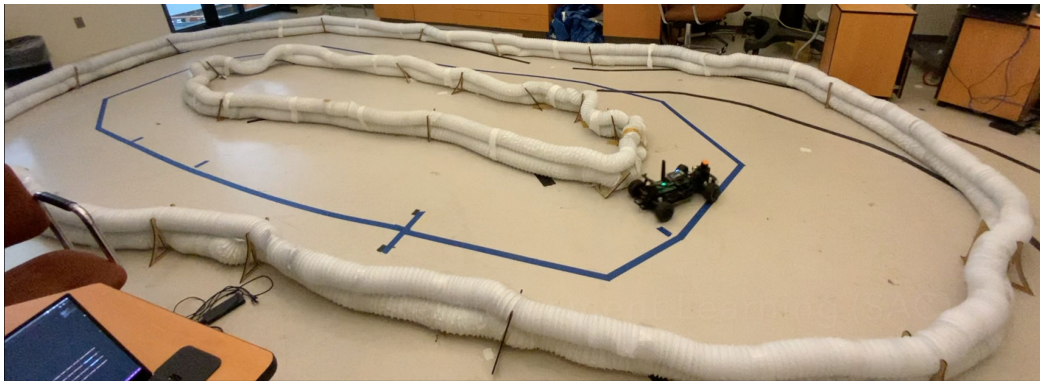

Figure III.4: Our real-world track with the reference path used for measuring the distance travelled marked in blue.

The results in Table III.4 show DDPG and IL-3 were unable to complete a lap, instead colliding with the side of the track before completing the first turn. However, both IL and SAC were able to complete over 4 laps in the allotted 60 seconds.

Table III.4: Performance On Hardware Platform

Algorithm	Track Distance	Bumps	Collisions
IL	53.86 ± 0.47	0	0
IL-3	5.81 ± 0.00	0	10
DDPG	2.00 ± 0.00	0	10
SAC	61.83 ± 0.37	4.7 ± 0.67	0

III.5 Discussion

Our experiments highlight two main challenges to the *sim2real* problem, *model mismatch* and *domain mismatch*. Model mismatch centers around the output not having the expected outcome. We highlight this challenge in our experiments with varying speed. Domain mismatch centers around the input being out of scope, or not what is expected. In other words, the real inputs do not match the training inputs. We highlight this challenge in our experiments with the obstacles and alternate racetracks. In this section, we discuss which controllers handled each type of mismatch best and theorize why that might be the case.

III.5.1 Model Mismatch

Model mismatch is a result of the output not having the expected outcome. This could be the result of noisy actuators, inaccurate model dynamics, etc. In our experiments, we highlight this challenge by testing the controllers at varying speeds in Table III.2.

Our results show the IL controllers handled this challenge better than the RL controllers. We attribute this result to how the controllers were trained. The IL controllers were trained to imitate the behavior of our expert control that balanced safety and efficiency by trending towards the middle of the track. In contrast, the RL controllers were trained solely to optimize efficiency. As a result, the RL controllers cut corners sharply and drove close to the walls. Because of this, changes to the speed had a larger impact on safety. Turning close to the walls has a smaller margin for error than turning in the middle of the track. Despite this greater challenge, the SAC controller was almost as safe as the IL controllers, with much better performance. If we compare the track distance of only safe trials, the SAC controller traveled an average of $201.25m$ and the IL controllers traveled an average of $182.3m$. The difference between the two is about $1/3$ of a lap.

III.5.2 Domain Mismatch

Domain mismatch is a result of the input data not matching the training input. This could be the result of noisy sensors, unexpected obstacles, or a change in environment. In our experiments, we highlight this challenge in or experiments by introducing obstacles (Table III.1) and testing on alternate racetracks (Table III.3).

For this challenge, there is not a clear victor since the results were more varied. The IL-3 controller

performed well across all the racetracks, but failed at obstacle avoidance. The SAC controller, on the other hand, successfully avoided colliding with obstacles in almost all our tests, but struggled when evaluated on alternate racetracks.

III.5.3 Sim2real

The experiments on our hardware platform help emphasize why *sim2real* is such a challenging problem. While the IL-3 controller maintained performance across all three racetracks and was the safest controller when we varied the speed, it failed to complete a single lap in the real world. Meanwhile, the IL controller, which had similar results with varied speed but struggled more on the different racetracks, successfully navigated our real world track. Because the IL-3 controller failed despite the IL controller's success, we theorize IL-3's failure was a result of overfitting. Overfitting occurs when the learned policy too closely or exactly matches the training data, and fails to generalize well to new data reliably. Training the IL-3 controller across multiple racetracks helped it perform well on all three tracks and improved its performance on *Porto*. However, all the extra training data in simulation, across varied racetracks, caused the controller to overfit to the simulation domain where the car can safely maintain a $1m$ distance from the left wall without colliding.

The varied training data that negatively impacted the IL-3 controller is likely what caused the SAC controller to succeed. While DDPG and SAC are similar RL approaches, they differ greatly in how they collect training data. In DDPG, new data is collected by adding random noise to the output of the learned policy. As the learned policy improves, the data collected starts to repeat. This repetition can cause undesirable effects on the learned policy, like *catastrophic forgetting* [193]. In contrast, SAC collects new data using an entropy maximizing function. This means that throughout the training process, new, unique, and varied data is prioritized. The result is a more robust learned policy with optimal performance.

III.5.4 Lessons Learned

III.5.4.1 Reinforcement Learning vs Imitation Learning

From the data and observations we collected throughout the training and evaluation processes, we found that reinforcement learning has a greater potential to learn robust and optimal control policies. However, the potential is lost without a well-defined reward function. Since RL focuses solely on optimizing performance, when we changed the track, many of the optimal performance strategies backfired and lead the car into collisions. We expect that this problem could be mitigated if we defined a reward function that incorporated an additional aspect, like a punishment for moving away from the center of the track. The result would be a more robust control policy that avoids colliding with walls, even in new tracks.

On the other hand, IL is still a valuable method, particularly when creating a well-defined reward function

is not possible. However, one of the main challenges with imitation learning lies in synthesizing a dataset that allows the agent to truly mimic the expert behavior. Although the training regime for these approaches resembles standard supervised learning regimes, the i.i.d assumption may no longer be valid [194]. Often, the current state of the system prompts the next state. Thus, if the agent makes a mistake in carrying out an action, it may eventually reach a state that the agent has never been trained on. For example, if the training data only contained state-action pairs where the agent was following a path in the center of the track, any deviation from this path could result in states outside the training data and suboptimal actions that lead the car straight into a wall. Therefore, while imitation learning is extremely effective in numerous applications, it can also fail spectacularly, like shown in our *sim2real* experiments.

III.5.4.2 Low Error is Not Necessarily a Good Indicator of Success

One commonly held principle within machine learning is that accuracy alone is generally a poor measure of evaluating a model's performance. In classification tasks, this can be addressed by using a metric such as an F1-Score, which balances the precision and recall of a model. However, it is not as straightforward for imitation learning tasks. In our experiments, we utilized mean-squared error to measure the effectiveness of our controllers. Curiously, some of the models that had very low error-rates, both on the test and validation set, could not complete a single lap. While other models, with a lower measured performance, did better on the driving task. This illustrates the need for better metrics for evaluating imitation learning tasks. There has been a large body of work towards this end over the last several years [195].

III.5.4.3 General Recommendations

We recognize that it is difficult to issue broad recommendations on a limited set of experiments. However, we believe the observations we made will translate to other platforms. Thus, we propose the following suggestions for those who wish to apply these techniques to other platforms:

- In general, we believe that RL approaches will fare better at *sim2real* tasks, since their inspiration is more conducive to exploring a wide range of state-action pairs than those considered in behavior cloning paradigms. However, reward shaping for these approaches is still extremely challenging. Therefore, one needs to weigh the cost of reward shaping against synthesizing expansive datasets for imitation learning models.
- Our experiments did not evaluate training or fine-tuning models in the real world. This choice was motivated by a desire to ensure fairness in the evaluation process between the two approaches. While it is straightforward to train imitation learning models on real-world data, training RL approaches in

the real world remains a challenge within the machine learning literature [196]. Our future work would like to consider an analysis of training and/or fine-tuning RL- and IL-trained models in the real world.

While imitation learning and reinforcement learning approaches are not widely used within production-ready, state-of-the-art autonomous vehicles, they have enjoyed significant success within industrial robotics applications. One such example of this success, is the rise of robotics companies leveraging these approaches, such as Alphabet's Intrinsic AI, Veo Robotics, Symbio, and Covariant. Still, there are few works comparing the success of imitation learning versus reinforcement learning approaches within these contexts. This work serves to motivate these types of studies in the community at large.

III.6 Related Work

There is a large body of work developing methods to improve reinforcement learning and overcome its shortcomings. These methods include ways to cut back on costly data collection and training time, reduce over-specialization, and improve the safety of the system during and after training is over. In this section, we highlight some of the promising methods we found in the literature. For an in-depth overview of how RL is being used in autonomous driving, we recommend Kiran et al.'s 2021 survey [197].

III.6.1 Offline Reinforcement Learning and Inverse Reinforcement Learning

Offline Reinforcement Learning, which is best described in [198], and Inverse Reinforcement Learning [199], are both similar to a combination of imitation learning and reinforcement learning.

Like IL, the training data is collected once and goes unaltered during the training process. Additionally, the agent does not interact with the environment at all during the training process until it is deployed after training is complete. This method is very beneficial to settings where data collection is slow, expensive, and/or dangerous like in robotics, autonomous driving, or healthcare.

The key aspect that allows both of these approaches to perform better than the policy used to collect the data is the use of a reward function. The reward function allows the agent to better infer what should be done in unexplored states, guiding the agent to perform optimally. In Offline RL, the reward function is known, but in Inverse RL, the reward function is inferred from observing expert behavior.

III.6.2 Meta Reinforcement Learning

Meta Reinforcement Learning (Meta-RL), best represented by model-agnostic meta-learning [200] and RL^2 [201], seeks to reduce over-specialization by training across multiple environments. In addition to making the agent more robust, the agent is able to learn to solve new tasks quickly. Some promising works in the area include *Joint PPO* [202] and *POET* [203].

III.6.3 Safe Reinforcement Learning

Safe Reinforcement Learning is grouped into two main styles of approach, (1) modification of the optimality criterion and (2) modification of the exploration process [204].

The first, often referred to under the broader term *reward shaping*, uses cleverly designed reward functions that incorporate risk in order to discourage unsafe behavior during training and ensure they avoid those unsafe behaviors when deployed[205].

The second style, often referred to simply as *safe exploration*, leverages external knowledge in the form of a *safety monitor*, a *shield*, *control barrier functions* (CBF), or some other form of *runtime assurance* (RTA) to ensure the agent remains safe while training [206, 207, 208, 209, 210].

III.6.4 Runtime Assurance

The most effective way to ensure safety after training, no matter the learning algorithm, is with *Runtime Assurance* (RTA). Especially in safety critical settings like autonomous driving, it is imperative that system designers prevent catastrophic failures that can result from biased or limited training data [211]. In recent years, numerous RTA approaches have been proposed, ranging from approaches that are statistical in nature [79, 80, 83, 84, 86], to more rigorous formal proof regimes [212, 213, 214, 215, 100, 102, 103]. Formally demonstrating the correctness of modern machine learning models is a difficult task that often suffers from the well-known state explosion problem [24]. While there has been a recent influx of formal methods capable of being run in real-time, statistical methods are the current leaders at circumventing scalability issues, though without the formal guarantees.

III.7 Future Work and Conclusions

In this work, we experimented with neural network controllers trained using imitation and reinforcement learning to compete in autonomous racing. We compared how the trained networks performed in new scenarios via zero-shot policy transfers. These scenarios tested the controllers' performance despite changes made to the operation of the vehicle and the track it was racing on. These changes were then combined by testing the controllers on our hardware platform.

The results show the RL controllers had more efficient performance even in new environments. SAC in particular was robust to the introduced static obstacles as well as the *sim2real* transfer. However, the RL controllers' more efficient performance led to more collisions. Therefore, unless work is done to train the RL controller to account for safety constraints, IL should be considered a competitive option for scenarios like this.

In future work, we would like to explore how the input space impacts performance by conducting the same

experiments in this work with a new neural network architecture that utilizes convolutional layers. Cameras are a standard sensor on most autonomous vehicles due to their ability to sense color and other fine-grained details in the environment. This makes them particularly useful for tasks such as traffic light recognition, and identifying possible road work. Moreover, there are numerous, remarkable machine learning algorithms within the computer vision community that can deal with images at high levels of accuracy.

III.8 Summary of Contributions

In summary, the contributions of this chapter are as follows:

1. We train a NNC using IL to imitate a path following algorithm that effectively balances efficiency and safety.
2. We train 2 NNCs using state-of-the-art RL algorithms, DDPG and SAC.
3. We compare their performance in a series of zero-shot policy transfer experiments in simulation.
4. We compare their performance in a *sim2real* zero-shot policy transfer experiment.

III.9 Neural Network Architectures and Hyperparameters

Below, we outline the neural network architectures and hyperparameters used in synthesizing the controllers considered in this chapter.

IL and IL-3 Hyperparameters:

- Network Architecture : (64, relu, 64, relu, tanh)
- Optimizer: Stochastic Gradient Descent, Nesterov Momentum
- Learning Rate (LR): 0.01
- Decay: 0.002
- Epochs: 100
- Loss: Mean Average Error

DDPG Hyperparameters:

- Policy Network (Actor): (64, relu, 64, relu, tanh)
- Q Network (Critic): (64, relu, 64, relu, linear)
- Actor LR: 0.0001

- Critic LR: 0.001
- Noise type: Ornstein-Uhlenbeck Process Noise $\sigma = 0.3$, $\theta = 0.15$
- Soft target update: $\tau = 0.001$
- $\gamma = 0.99$
- Critic L2 reg: 0.01
- buffer size: 10^6
- batch size: $B = 64$
- episode length: $T = 500$
- maximum number of steps: 45000

SAC Hyperparameters:

- Policy Network (Actor): (64, relu, 64, relu, tanh)
- Q Networks (Critics): (64, relu, 64, relu, relu)
- learning rate: 0.0001
- Soft target update: $\tau = 0.001$
- $\gamma = 0.99$
- $\alpha = 0.01$
- buffer size: 10^6
- batch size: $B = 64$
- episode length: $T = 500$
- maximum number of steps: 45000

CHAPTER IV

Online Safety Assurance for Machine learning Controllers with Real-Time Reachability

This chapter is adapted from the material presented in [16].

Over the last decade, advances in machine learning and sensing technology have paved the way for the belief that safe, accessible, and convenient autonomous vehicles may be realized in the near future. Despite the prolific competencies of machine learning models for learning the nuances of sensing, actuation, and control, they are notoriously difficult to assure. The challenge here is that some models, such as neural networks, are "black box" in nature, making verification and validation difficult, and sometimes infeasible. Moreover, these models are often tasked with operating in uncertain and dynamic environments where design time assurance may only be partially transferable. Thus, it is critical to monitor these components at runtime. One approach for providing runtime assurance of systems with unverified components is the simplex architecture, where an unverified component is wrapped with a safety controller and a switching logic designed to prevent dangerous behavior. In this chapter, we propose the use of a real-time reachability algorithm for the implementation of such an architecture for the safety assurance of a 1/10 scale open source autonomous vehicle platform known as F1/10. The reachability algorithm (a) provides provable guarantees of safety, and (b) is used to detect potentially unsafe scenarios. In our approach, the need to analyze the underlying controller is abstracted away, instead focusing on the effects of the controller's decisions on the system's future states. We demonstrate the efficacy of our architecture through experiments conducted both in simulation and on an embedded hardware platform.

IV.1 Introduction

The vision of a "driverless" future has riveted many technology enthusiasts, researchers, and corporations for decades [216]. The prevailing conviction is that there are relatively few technologies that hold as much promise as autonomous vehicles (AVs) in bringing about safe, accessible, and convenient transportation. Particularly, when the status-quo is considered, far too many individuals lose their lives to traffic fatalities each year [216]. As Koopman et al. write, "The question is not whether autonomous vehicles will be perfect. The question is when [will] we be able to deploy a fleet of fully autonomous driving systems that are actually safe enough to leave the human completely out of the driving loop [216]."

The two fundamental challenges widely regarded as limiting the arrival and widespread adoption of AVs are safety and reliability [217]. Reasoning about safety requires an understanding of the joint dynamics of

computers, networks, and physical dynamics in uncertain and variable environments, making it a notoriously difficult problem [8]. To handle the complexities of their environments, many AVs make use of *Machine Learning* (ML) components to decipher the information observed from an ever-evolving configuration of on-board sensors [8]. Despite the impressive capabilities of these components, there are reservations about using them within safety-critical settings due to their largely opaque nature. Utilizing a "black-box" model within a system that is safety-critical constitutes the highest form of technical debt [218] and, as a result, the last several years have witnessed a significant increase in the development of techniques that seek to reason about the safety and robustness of machine learning methods [9].

Unfortunately, despite numerous works proposed in the past few years for the formal analysis of machine learning methods, the vast majority of these efforts have not been able to scale to the complexity found in real world applications, where models such as neural networks may be characterized by millions or even billions of parameters [121]. Thus, designing solutions that are both practical and rigorous is extremely challenging. One approach that has enabled the assurance of systems with unverified components is the *simplex architecture* [219]. In this framework, an unverified component is wrapped with a safety controller and switching logic designed to transfer control to the safety controller in certain situations [108]. The key challenge in this regime is to design a switching logic that allows the dynamic capabilities of the unverified, complex controller to be employed without compromising safety. In this chapter, we extend the real-time reachability algorithm from [108, 106] to design a simplex architecture for a $1/10$ scale autonomous racing car called the F1/10 platform.

To put our work into context, this work falls within the *runtime verification* or *runtime assurance* realm. Our aim is to construct an architecture that allows us to ensure that an autonomous vehicle, controlled using machine learning strategies, never enters unsafe states as it navigates an environment. Specifically, the set of control strategies presented herein were synthesized using deep reinforcement learning and imitation learning, which have generated a considerable amount of excitement in recent years [187]. One strength of our approach is it abstracts away the need to analyze the underlying nature of these controllers and instead observes the influence of their decisions on the system behavior at runtime.

To perform the verification, we first identify a dynamical model of the car and assume that the car operates within an *a priori* known environment. Next, we synthesize controllers using data collected from a series of experiments with the F1/10 vehicle, as well as through the execution of a series of deep reinforcement learning training campaigns. Using the obtained controllers, we aim to verify that the car does not crash into static obstacles within its environment in addition to the environment boundaries. To do this, we extend a real-time reachability algorithm of Bak et al. [108, 106] to compute the set of reachable states for a finite time-horizon and check for potential collisions. This safety checking forms the basis of the switching scheme

in our simplex architecture, and we evaluate the merits of this approach both in simulation and on the F1/10 hardware platform using a variety of controllers, number of obstacles, and runtime configurations.

IV.2 Background: The Simplex Architecture and Real-Time Reachability

IV.2.1 Simplex Architecture

As modern autonomous systems grow in complexity, so do the challenges in assessing their reliability and correctness [13]. Moreover, any arguments about the reliability and safety of the system rely on assertions about the individual components that make it up [108]. However, in recent years, with the growth of increasingly autonomous systems [220], individual components may be designed using machine learning methods, such as neural networks, that are opaque to traditional formal analysis. Despite the recent years' surge in the development of formal analysis techniques for these types of models [114, 9], most techniques are incapable of dealing with the scale of models deployed in state-of-the-art systems.

One paradigm for dealing with untrustworthy components is the *simplex architecture* [107]. In the simplex architecture, the unverified component, or *complex controller*, is wrapped with a *safety controller* and a switching logic used to ensure safety [108]. A useful analogy for this architecture is a driving instructor's car with two steering wheels and two sets of brakes. As long as the instructor is capable of intervening in dangerous situations, the capricious student is allowed to drive. Typically, the complex controller has better performance with respect to the design metrics, whereas the safety controller is designed with simplicity and verifiability in mind. Thus, by using this architecture, one can utilize the complex controller while still maintaining the formal guarantees of the safety controller. The key challenge when designing a system with the simplex architecture is properly designing the switching logic [106]. One must be able to clearly delineate safe states from unsafe states.

Typically, in simplex architectures, the switching logic is primarily designed either from a control theoretic perspective through the solution of linear matrix inequalities (LMI) [110], or using a formal analysis hybrid-systems reachability technique [13]. In this manuscript, our simplex design necessitates computing the set of reachable states online through the use of a real-time reachability algorithm for short time horizons.

IV.2.2 Real-Time Reachability

Reachability algorithms have traditionally been executed offline and are typically computationally intensive endeavors [60, 52, 147]. However, in [108, 106], Bak et al., and Johnson et al. presented a reachability algorithm, based on the influential mixed face-lifting algorithm [221], capable of running in real-time on embedded processors. The algorithm is implemented as a standalone C-package that does not rely on sophisticated (non-portable) libraries, recursion, or dynamic data structures and is amenable to the anytime

computation model in the real-time scheduling literature [167]. In this regime, each task produces a partial result that is improved upon as more computation time is added [106].

The controllers used in our experiments are designed to sample sensor data and compute control actions at fixed time intervals as typically done in the control community [222]. During each control period, we take the corresponding control action and compute the reachable set of states into the future as defined by the current state and a specified finite-time horizon. An example of this computation is shown in IV.1. We assume a fixed control action throughout the reachable set computation. Based on the obtained reachable set, we determine if the system will collide with objects in its environment and, if necessary, switch to a safety controller optimized for obstacle avoidance. If the system falls back to using a safety controller, we only allow a switch back to the complex controller if the complex controller has demonstrated safe behavior for a fixed number of control periods[1]. This prevents arbitrary switching and incorporates a sense of hysteresis into our control strategy. Additionally, by not switching back until consistently safe behavior has been demonstrated, we enforce a notion of dwell time, which reduces instabilities caused by switching too frequently.

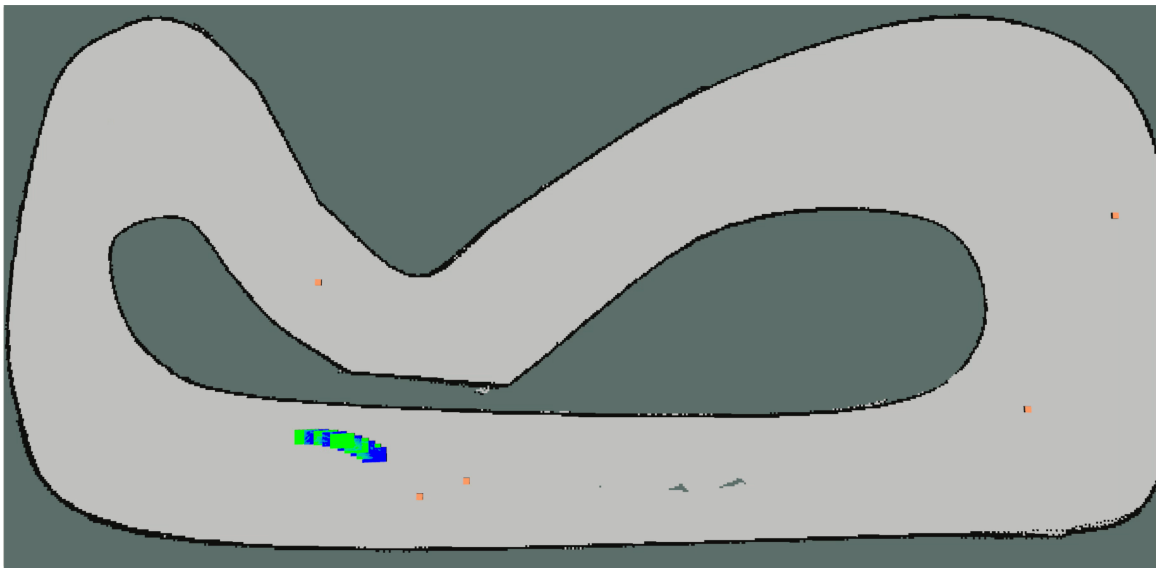

Figure IV.1: Visualization of the set of reachable states using the current control action. This example corresponds to a safe scenario, as there is no intersection with obstacles or the the racetrack walls. The orange squares represent the location of cones and their corresponding bounding box.

IV.3 Experimental Overview

To build and assure the safety of our system at runtime, we perform the following steps. First, we construct a mathematical model of the F1/10 car's physical dynamics using system identification techniques. We then deploy one of our trained *Machine Learning* (ML) controllers in the control architecture. These controllers

[1] In our experiments we allowed a switch back to the safety controller after 25 control periods. This corresponds to 1.25 seconds using a 20 Hz control period.

are: (1) *Imitation Learning* (IL) controllers trained to mimic driving behavior using data collected from a series of experimental runs of driving with a baseline controller and (2) *Reinforcement Learning* (RL) controllers trained using multiple RL algorithms. The controllers use sensor information to determine the desired steering angle for the vehicle. At runtime, the mathematical model obtained through system identification is used within the reachability algorithm to reason about safety of the control actions selected by the ML controllers. The simplex architecture provides the framework for ensuring safe operation of the F1/10.

IV.3.1 The F1/10 Autonomous Platform

The F1/10 platform proposed by Matthew O'Kelly et al. in [174] was originally designed to emulate the hardware and software capabilities of full scale autonomous vehicles. The platform is equipped with a standard suite of sensors such as stereo cameras, LiDAR (light detection and ranging), and inertial measurement units (IMU). The platform uses an NVIDIA Jetson TX2 as its compute platform, and its software stack is built on the *Robot Operating System* (ROS)[2] [189]. The result is a platform that allows researchers to conduct real-world experiments which investigate planning, networking, and intelligent control on a relatively low-cost, open-source test-bed [174]. Additionally, in order to promote rapid prototyping and consider research questions around closing the simulation to reality gap[224], Varundev Suresh et al. designed a Gazebo-based simulation environment [225] that includes a realistic model of the F1/10 platform and its sensor stack [226]. We utilize this simulation environment for a number of experiments and training our controllers.

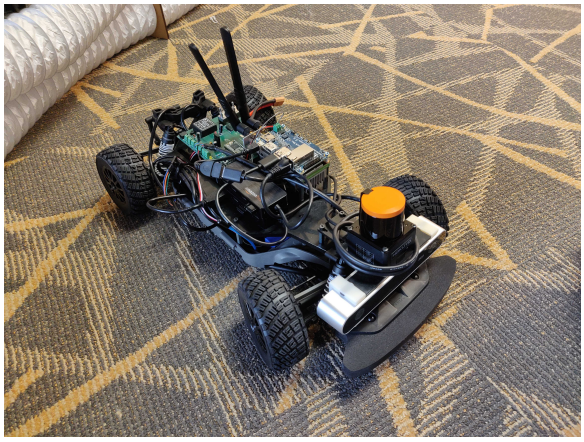

Figure IV.2: Visualization of our experimental F1/10 hardware platform [2].

IV.3.2 Vehicle Dynamics Model and System Identification

The physical dynamics of the F1/10 vehicle are modeled using a kinematic bicycle model [227], which is described by a set of four-dimensional nonlinear ordinary differential equations (ODEs). The kinematic

[2]It is worth noting that ROS is not an operating system in the traditional sense but rather a meta-operating system that primarily provides the message passing interface for various components within robot software development [223].

bicycle model is characterized by relatively few parameters and tracks reasonably well at low speeds.[3] The model has four states: Euclidean positions x and y, linear velocity v, and heading θ. The dynamics are given by the following ODEs:

$$\begin{aligned}
\dot{x} &= v\cos(\theta+\beta) \\
\dot{y} &= v\sin(\theta+\beta) \\
\dot{v} &= -c_a v + c_a c_m (u - c_h) \\
\dot{\theta} &= \frac{v\cos(\beta)}{l_f + l_r}\tan(\delta) \\
\beta &= \tan^{-1}\left(\frac{l_r \tan(\delta)}{l_f + l_r}\right)
\end{aligned}$$

where v is the car's linear velocity, θ is the car's orientation, β is the car's slip angle, x and y are the car's position, u is the throttle input, δ is the steering input, c_a is an acceleration constant, c_m is a motor constant, c_h is a hysteresis constant, and l_f and l_r are the distances from the car's center of mass to the front and rear respectively [228]. For simplicity, since the slip angle is fairly small at low speeds, we assume that $\beta = 0$. Using MATLAB's Grey-Box System Identification toolbox, we obtained the following parameters for the simulation model: $c_a = 1.9569$, $c_m = 0.0342$, $c_h = -37.1967$, $l_f = 0.225$, $l_r = 0.225$. The model was validated using a series of experimentswith an average *Mean Squared Error* (MSE) of 0.003. A sample experimental simulation is shown in IV.3. For the hardware platform, we obtained the following parameters: $c_a = 2.9820$, $c_m = 0.0037$, $c_h = -222.1874$, $l_f = 0.225$, $l_r = 0.225$, with a validation MSE of 6.75×10^{-4}.

IV.4 Controller Construction

Modern data-driven or machine learning methods have become increasingly scalable and efficient in dealing with complex problems in numerous contexts. In this section, we provide a high-level introduction to imitation learning and reinforcement learning and describe the construction of the controllers used within our simplex architecture.[4]

IV.4.1 Imitation Learning

Imitation learning (IL) seeks to reproduce the behavior of a human or domain expert on a given task [185]. These methods fall under the branch of *Expert Systems* in AI, which has seen a surge in interest in recent years. The increased demand for these approaches is spurred on by two main motivations. (1) In many settings, the number of possible actions needed to execute a complex task is too large to cover using explicit

[3] The kinematic bicycle model typically tracks well under $5m/s$ [228]

[4] All the artifacts used to train the controllers can be found in the following repository **https://zenodo.org/record/5879646**.

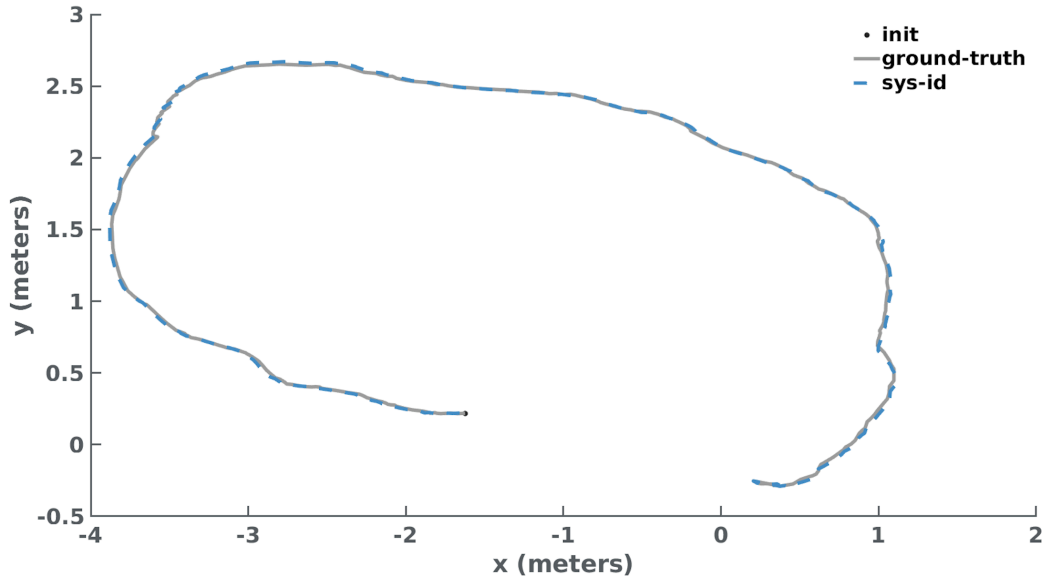

Figure IV.3: Illustrative visualization of a scenario used in validating the F1/10 Hardware Model. The model validation process was performed using a sizeable set of diverse experiments. *Init* corresponds to the starting position of the vehicle in the experiment illustrated above.

programming. (2) Demonstrations show that having prior knowledge provided by an expert is more efficient than learning from scratch [185]. While these approaches have demonstrated great efficacy in fixed contexts, there are concerns regarding their ability to generalize to novel contexts where the operating conditions are different from those seen during training, providing a need for effective runtime verification like the one explained in this work [185].

One of the most common imitation learning methods is *Behavior Cloning*, whereby a controller is constructed by learning a mapping from sensor-action pairs collected either from a baseline controller or through human-in-the-loop control. While these approaches have demonstrated great efficacy in fixed contexts, there have been concerns about their ability to generalize to novel contexts where the operating conditions are different from those seen during training [217].

In this work, we utilize behavior cloning to train two neural network controllers to produce steering angles from sensor inputs. The first controller is trained on camera-images and leverages an architecture used by Bojarski et al. to control a real-world autonomous vehicle [187]. The second controller utilizes a much smaller neural network model and is trained on a discrete sampling of LiDAR distance measurements. This network allows for more fair comparisons to the reinforcement learning approaches.

IV.4.1.1 Vision-Based Navigation (VBN)

Since the seminal work of Krizhevsky et al. [229] in the ImageNet Large Scale Recognition Challenge, *Convolutional Neural Networks* (CNNs) have revolutionized the field of computer vision. Within the context of autonomous vehicles, CNNs have demonstrated efficacy for driving tasks such as lane following, path planning, and control, simultaneously, by computing steering commands directly from images [187].

We utilized the CNN architecture, DAVE-2, initially proposed by Bojarski et al. to drive a 2016 Lincoln MKZ, in order to control the F1/10 model. The data we used to train DAVE-2 was collected from a set of simulation experiments where the sensor-action pairs were generated by a path tracking controller optimized to keep the F1/10 in the center of the track in the absence of obstacles. Such an environment is shown in IV.1.

IV.4.1.2 Lidar Behavior Cloning (LBC)

The second network considered for behavior cloning was a standard multi-layer perceptron network that consisted of an input layer, 2 fully connected hidden layers of 64 nodes with ReLU activation functions, and a fully connected output layer with a tanh activation function. The input layer accepts nine range values collected from the LiDAR at -90°, -60°, -45°, -30°, 0°, 30°, 45°, 60°, and 90° from forward. The range values are clipped between $[0m, 10m]$. The data used to train this controller was collected in the same fashion as the end-to-end regime.

IV.4.2 Reinforcement Learning Control

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are branches of machine learning that focus on software agents learning to maximize rewards in an environment through experience. Despite the growing success of DRL approaches in many contexts, these methods are mainly leveraged within simulation due to challenges with ensuring safe training in real-world systems, designing reward functions that deal with noisy and uncertain state information, and ensuring trained controllers are able to generalize beyond fixed scenarios [196]. While training a controller in simulation and moving it into the real world is possible, a process known as *sim2real* transfer, it often results in undesired, poor, and/or dangerous behavior [196].

In this chapter, we used two well-known state-of-the-art reinforcement learning algorithms, an off-policy DRL algorithm known as *Soft-Actor-Critic* (SAC), [188], and an on-policy RL algorithm, known as *Augmented Random Search* (ARS), [191]. In line with the imitation learning experiments, the agents were trained on the racetrack shown in, IV.1 with no obstacles and no backup controller. For both algorithms, the agent optimizes performance on a dense reward function that assigns a positive reward for counterclockwise progress around the track. The reward is calculated using a reference path that runs through the middle of the track. The reward value is the positive arc length between the previous and current closest point along the path. This

reward function encourages the agent to complete as many laps as possible as quickly as possible.

IV.4.2.1 Soft Actor Critic (SAC)

This algorithm was first introduced in 2018 as an improvement to *Deep Deterministic Policy Gradient* (DDPG) that tackled RL's major challenges: high sample complexity and brittle convergence properties, i.e. a heavy dependence of hyperparameters being "just right" in order to effectively learn [188]. In this work, the SAC controller was trained using the same architecture as the LBC controller described in Section IV.4.1.2

IV.4.2.2 Augmented Random Search (ARS)

This algorithm was first proposed in 2018 as a random search method for training static, linear policies for continuous control problems [191]. Their simple method was able to match state-of-the-art sample efficiency on the benchmark MuJoCo locomotion tasks,[5] demonstrating that deep neural networks might not be necessary for some complex control tasks. We chose to highlight this RL algorithm because it allowed us to experiment with a different control architecture. Both LBC and SAC are the same NN architecture, differentiated by how the networks are trained. In contrast, ARS focuses on the use of a linear policy, i.e. a weight matrix. Instead of passing the input through multiple layers with non-linear activation functions, the input is multiplied by a single weight matrix to generate the control output.

Similar to the SAC architecture, the output control signal of the ARS policy, $\delta = \pi(s)$, is the desired steering angle clipped between $\pm 34^{\circ}$. However, the input, s, consists of 271 LiDAR range values, clipped between $[0m, 10m]$, collected from between $\pm 90^{\circ}$ from forward.

IV.5 Online Reachability Computation

Before outlining the algorithm, let us define two key terms relevant to our approach.

Definition 1 *(REACHTIME). The reachtime, T_{reach}, is the finite time horizon for computing the reachable set.*

Definition 2 *(RUNTIME). The runtime, $T_{runtime}$, is the duration of (wall) time the algorithm is allowed to run.*

Using the dynamics model obtained for the F1/10, the crux of the real-time reachability algorithm is computing the set of reachable states from the current time t up until $(t + T_{reach})$. The algorithm utilized within this work is based on mixed face-lifting, which is part of a class of methods that deal with *flow-pipe construction* or *reachtube computation* [106]. This is done using snapshots of the set of reachable states that are enumerated at successive points in time. To formalise this concept, we define the reachable set below.

[5] These benchmark tasks are described in more detail at **https://gym.openai.com/envs/#mujoco**

Definition 3 *(REACHABLE SET). Given a system with state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$, input vector $\boldsymbol{u}(t) \in \mathbb{R}^m$, and dynamics $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where t is time, and the initial states $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ and inputs $\boldsymbol{u}_0 = \boldsymbol{u}(0)$ are bounded by sets, $\boldsymbol{x}_0 \in \chi_0$, $\boldsymbol{u}_0 \in U$. The reachable set of the system for a time interval $t \in [0, T_{reach}]$ is:*

$$R_{[0,T_{reach}]} = \left\{ \psi(\boldsymbol{x}_0, \boldsymbol{u}_0, t) \;\middle|\; \boldsymbol{x}_0 \in \chi_0,\ \boldsymbol{u}_0 \in U,\ t \in [0, T_{reach}] \right\},$$

where $\psi(\boldsymbol{x}_0, \boldsymbol{u}_0, t)$ is the solution of the ODE at time t with initial state $\boldsymbol{x}_0$ under control input $\boldsymbol{u}_0$.[6]

For general nonlinear systems, it is not possible to obtain the exact set of reachable states $R_{[0,T_{reach}]}$, so we customarily compute a sound over-approximation such that the actual system behavior is contained within the over-approximation [52, 58, 221]. The algorithm utilized in this work utilizes n-dimensional hyper-rectangles ("boxes") as the set representation to generate reachtubes [106]. Over long reachtimes, the over-approximation error resulting from the use of this representation can be problematic. However, for short reachtimes, it is ideal in terms of its simplicity and speed [108].

The over-approximation error and the number of steps used in generating the reachable set can be controlled by a reachtime step size h. This parameter defines the level of discretization of the time interval $[0, T_{reach}]$ and can be used to tune the runtime of the reachability computation. Bak et al. leverage the step size to make the reachability algorithm amenable to the anytime computation model in the real-time scheduling literature. Given a fixed runtime, $T_{runtime}$, we compute the reachable set $R_{[0,T_{reach}]}$. If there is remaining runtime, we restart the reachability computation with a smaller step size. In both this work and [108], the step-size is halved in each successive iteration, leading to more accurate determinations of the reachable set. The relationship between the over-approximation error and the step size is demonstrated in IV.4. We refer readers to the following papers for an in depth treatment of these procedures [221, 108, 106].

IV.6 Safety Checking

We define the notion of safety considered in this work below.

Definition 4 *(SAFETY). Let Λ represent the set of unsafe states. A system is considered safe over the finite time horizon, T_{reach}, if $R_{[0,T_{reach}]} \cap \Lambda = \emptyset$. Here, Λ, consists of all static obstacles within the environment and the boundaries of the racetrack.*

In our autonomous racecar scenario, Λ, consists of all static obstacles within the environment, described by a bounding-box, and the boundaries of the racetrack, characterized by a list of finely separated points.

[6] Our assumption is that f is globally Lipschitz continuous. This property guarantees the existence and uniqueness of a solution for every initial condition in χ_0.

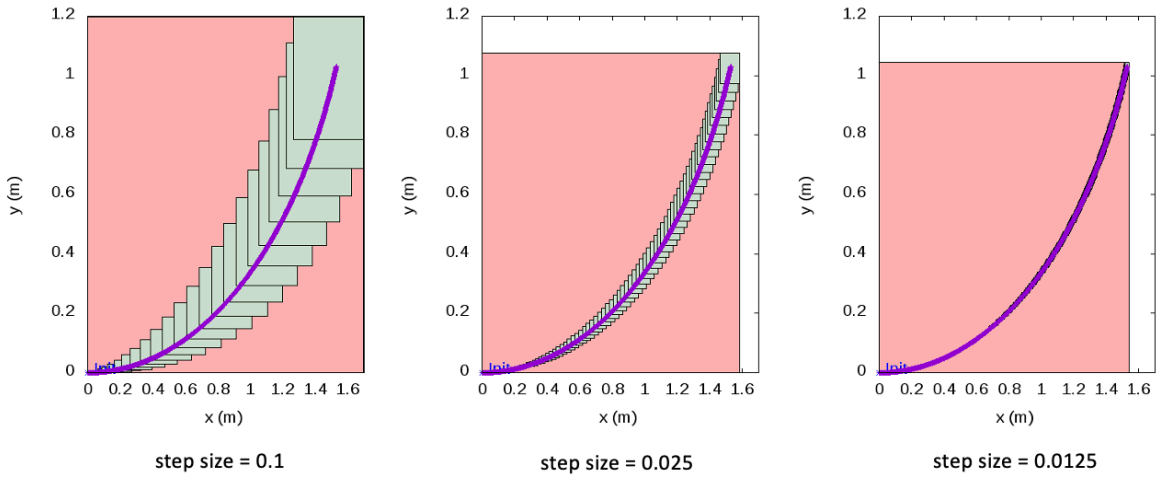

Figure IV.4: The real-time reachability algorithm always returns an over-approximation of the reachable set of states. The over-approximation error decreases with successive iterations, provided that there is enough runtime for re-computations. The above images demonstrate this aspect by simulating a left hand turn control action for a reachtime of two seconds. The green boxes represent the set of reachable states, the red rectangle represents the interval hull of the reachable states, and the purple points are points obtained from a simulation of the vehicles physical dynamics.

These representations are then converted into their hyper-rectangle formulations that make up Λ. IV.1 provides a visualization of the obstacles, and IV.6 displays our discretization of the racetrack. If there are no intersections between $R_{[0,T_{reach}]}$ and Λ, then we conclude that the system is safe. However, since our approach computes an over-approximation of $R_{[0,T_{reach}]}$, it may lead to conservative observations of unsafe behavior. This occurs when the error in the over-approximation of $R_{[0,T_{reach}]}$ results in intersections with the set of unsafe states, despite these intersections not occurring with the exact reachable set. By refining the reachable set in successive iterations, our regime seeks to mitigate the occurrence of falsely returning *unsafe*.The two chief considerations in the anytime implementation of the safety checking procedure are (1) the overall soundness of our approach, and (2) the real-time nature of our scheme. Satisfying both requirements constitutes the novel extensions of the aforementioned algorithm. For the results of the verification to be sound, the safety checking process must be carried out in its entirety before a safety result is issued. At the same time, this requirement must be balanced alongside the real-time stipulation that tasks operate within pre-defined and deterministic time spans. Thus, our implementation ensures soundness properties while maintaining a low-likelihood of missing timing deadlines.

Ideally, to ensure there were no missed deadlines, we would build our system in a *Real-Time Operating System* (RTOS), which allows for the specification of task priorities, executing them within established time frames. However, our implementation does not make use of an RTOS, and instead depends on native Linux and ROS to handle task management. To combat this shortcoming and reduce the number of missed dead-

lines, we estimate how the time required to compute the next reachability loop. If our estimate exceeds the remaining allotted time, the process terminates. There is an inherent tradeoff between the conservativeness of our runtime estimates and the conservativeness of the resulting reachable set. In this work, we chose to maximize the number of iterations used in constructing the reachable set at the risk of occasionally missing deadlines. Our experiments demonstrate that we were successful in minimizing the number of missed deadlines during operation.

Let k denote the number of hyper-rectangles used in representing the reachable set. k is characterized by the following equation: $k = T_{reach}/h$.[7] Since each successive iteration decreases the step size by half, the number of hyper-rectangles that make up $R_{[0,T_{reach}]}$ doubles. Thus, the complexity of the safety checking process is $O(2^k)$.[8] Therefore, we can estimate that a subsequent iteration of the algorithm will take twice as long as the current one and bloat this estimate to be conservative. To ensure that our estimates are accurate in our implementation, rather than deriving $R_{[0,T_{reach}]}$ and then checking whether the system has entered an unsafe scenario, the safety checking is done during the computation with intermediate hyper-rectangles as outlined in [108]. This prevents us from needing to dynamically store the reachable set and allows us to restart the computation sooner if an unsafe state is detected.

IV.7 ROS Simplex Architecture

Our simplex architecture for the F1/10 is designed using ROS [189], and an overview of the design is shown in Figure IV.5. There are two considerations that play a major role in designing this architecture: (1) the finite time horizon, T_{reach}, over which we are reasoning about safety, and (2) the amount of time, $T_{runtime}$, allocated for the computation of the reachsets. In our experiments, we use $T_{reach} = 1.0s$ and $T_{runtime} = 25ms$, unless otherwise specified. These values were determined considering the empirical results of how long it took the F1/10 to come to a stop at speeds less than $1.5m/s$, and the control period, $20Hz$, which the reachability computation needs to finish within in order to not miss a deadline.[9]

Within this architecture, the primary sensors we rely on are a LiDAR and Stereo Labs' Zed Depth Camera. The messages from the LiDAR are published at $40Hz$, and the camera messages are published at $20Hz$. Additionally, we rely on odometry information, published at $40Hz$, in order to ascertain the state of the F1/10 vehicle. In our design, we decouple the control of the car's steering and throttle control. The steering control, δ, is governed by the ML controller, and the throttle control is designed to maintain a constant speed, u, when the learning-based controller is in use.

[7] We begin the flow-pipe construction with an initial time step of $h = T_{reach}/10$.

[8] This analysis neglects consideration of the obstacles and the points used to represent the racetrack boundaries. Since these do not change between iterations, reasoning only about the hyper-rectangles is sufficient.

[9] We limit velocities to $1.5m/s$ because a lap on our physical track is approximately $13.08m$. Races held by the F1/10 community are around $30-50m$ per lap with larger distances between the track walls, allowing for much faster operating speeds. The rules are described in more detail here: **https://f1tenth.org/misc-docs/rules.pdf**

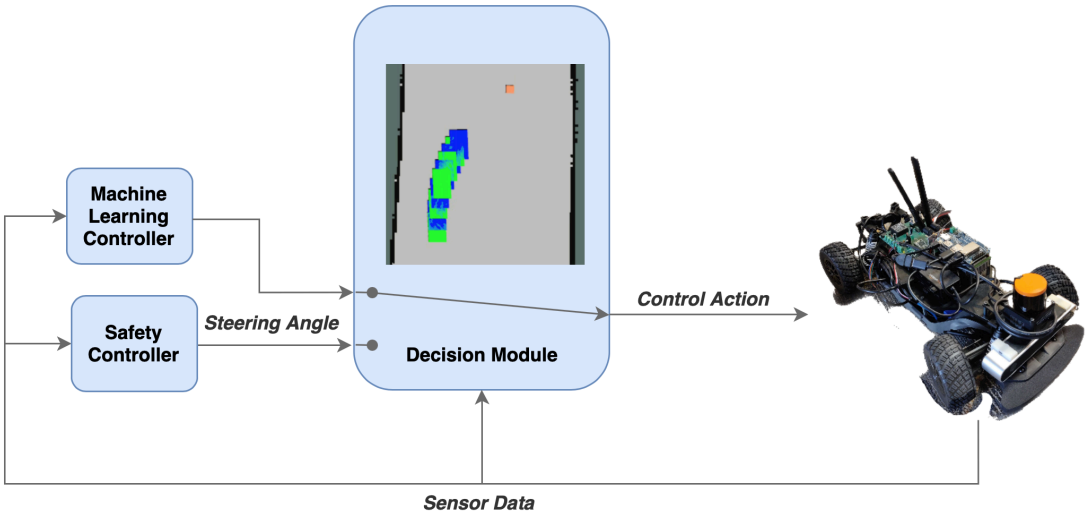

Figure IV.5: Overview of the Simplex architecture deployed on the F1/10 System as described in Section IV.7. The switching logic consists of monitoring the intersection between the F1/10 reachable set and the positions of static obstacles within the environment.

In the traditional simplex architecture, both the decision module and the safety controller must be verified for the system to be verified as correct [108]. While this is straightforward for relatively simple controllers, it is significantly more challenging for many classes of controllers, especially when real-time execution is considered [228]. However, the main focus of this work is evaluating the use of the reachability algorithm as a switching logic for the simplex architecture. Thus, we opted not to develop a "formally verified" safety controller. Instead, we selected a controller based on a gap-following algorithm optimized to avoid collisions with obstacles. A detailed description of the gap-following algorithm can be found in the following report [230]. It was primarily selected due to its robust collision avoidance ability and simplicity.

IV.8 Experimental Evaluation

Having described the details of our reachability algorithm, controller construction, and the simplex architecture construction, we now present the results of our empirical evaluations both in simulation and on the hardware platform. We ran our experiments on platforms running Linux (Ubuntu 16.04 LTS). The simulation experiments were conducted on a Dell XPS-15 (9570) with 32GB RAM, a six-core Intel Core i7-8750 @4.1GHz processor, and an Nvidia GeForce GTX 1050Ti 4GB graphics card. The motivation for conducting simulation experiments stemmed from a desire to ensure fair comparisons over numerous experiments and to promote reproducibility for those without hardware access. The hardware experiments were done on an Nvidia Jetson TX2 with a Dual-core Nvidia Denver 64-bit CPU (ARM), a quad-core ARM A57 Complex, and an NVIDIA Pascal Architecture GPU with 256 CUDA cores. The latter configuration validates our

claims that our safety architecture admits minimal resource requirements.

The evaluation included a sizeable diversity of experiments with respect to the speed set-point utilized by the ML controller, u, the wall-time utilized by the real-time reachability algorithm, $T_{runtime}$, the presence and configuration of obstacles, and an examination of how each controller performed in each context. This allowed for an enlightening analysis on the various trade-offs that exist within our safety architecture. For context, the speed and wall-time analysis was evaluated on 48 different combinations of 30 experimental executions.

For benchmarking purposes, we recorded the mean execution-times (Mean ET) of our real-time reachability algorithm, as well as the average number of iterations utilized in constructing the reachable set (Mean Iters). While a typical discussion of upper bounds on execution times involves a discussion of the *Worst-Case Execution Time* (WCET), we instead report the Mean ET. In general, the WCET is unknown or difficult to derive without the use of static analysis proofs [231]. Since our safety regime relies on ROS, which is highly dynamic and distributed, it is prohibitively difficult to perform an exhaustive exploration of the space of all execution times and thus derive the WCET. However, we provide a rough proxy of the WCET by reporting the *Maximal Observed Execution Times* (MOET) [231] in Table IV.3. Additionally, we report the *Percentage of Missed Deadlines* (PMD) that result from our soundness requirements; as we execute on a regular operating system and not on a RTOS, this is possible, and performing the runtime measurements may result in variance due to changing load and scheduling. We demonstrate that this value is low across all experiments.

In the tables that follow, ML refers to the machine learning controller, and all summary statistics are reported alongside their corresponding standard deviations. Additionally, to analyze the conservativeness of the regime, we report the percentage of the time in which the machine learning controller was utilized during an experimental run (ML Usage).

IV.8.1 Simulation

In the considered simulation scenarios, the F1/10 vehicle is tasked with navigating a racetrack environment that has 6 traffic cones placed at random locations before the start of the experiment. The locations of the cones are known *a priori*, and IV.1 provides a snapshot of this setup. Utilizing the control architecture discussed in Section IV.7, we ran 1440 simulation episodes with a timeout of 60 seconds each. That is roughly enough time to complete 2 laps at a speed of $1m/s$.

Each of the controllers was trained with an assumption that the F1/10 moves at a speed $u = 1.0m/s$. Thus, the experiments at this speed provide a baseline as we consider variations in speed, obstacles, and runtime. From inspecting IV.1, one can see that moving at speeds faster than $1.0m/s$ was correlated with lower levels of safety. In particular, the SAC controller was the least tolerant to increases in speed. Since the SAC controller

was trained to complete laps as quickly as possible, it was often aggressive around turns, resulting in higher declarations of unsafe behavior. In contrast, the VBN controller was the most robust to speed changes. Still its performance varied significantly as displayed by the standard deviation of the controller usage at speeds of $1.5m/s$. It is worth noting, however, that the VBN has 340 times more parameters than the SAC and LBC controllers.[10]

Out of the 1440 experiments conducted in simulation, we observed collisions in 93 of them. This corresponds to a 93.5% success rate in preventing unsafe behavior. Of these collisions, 53 of them occurred at a speed set point of $1.5m/s$ and when using the IL controllers. We were able to eliminate these collisions in subsequent experiments by increasing the reach time horizon.[11] However, in practice in order to provably eliminate all collisions, one must also verify the decision module and the safety controller utilized within the simplex regime.

Table IV.1: Machine Learning Controller Use in the Simplex Architecture: Simulation Platform

		Obstacles	No Obstacles
ML Controller	u (m/s)	Mean ML Usage (%)	Mean ML Usage (%)
VBN	0.5	78.26 ± 12.74	94.08 ± 2.92
	1.0	53.98 ± 15.51	78.56 ± 4.34
	1.5	37.97 ± 14.11	53.07 ± 9.89
LBC	0.5	83.23 ± 14.25	99.32 ± 1.07
	1.0	61.40 ± 21.96	94.83 ± 6.95
	1.5	32.74 ± 12.36	43.67 ± 16.92
SAC	0.5	42.59 ± 17.80	50.32 ± 10.29
	1.0	13.20 ± 8.42	11.13 ± 8.36
	1.5	9.47 ± 4.19	8.87 ± 3.94
ARS	0.5	66.62 ± 9.72	69.01 ± 8.98
	1.0	46.18 ± 7.57	49.99 ± 3.85
	1.5	26.50 ± 7.56	30.79 ± 5.02

IV.8.2 Hardware

While the simulation experiments allowed us to evaluate our regime on a diverse set of scenarios, the hardware experiments validate our claim that our safety architecture admits minimal resource requirements. Additionally, these experiments allowed us to consider the seminal problem of *sim2real* transfer, which is a challenging problem within ML and robotics at large [196]. Training ML controllers in simulation and deploying them on real-world hardware platforms, *sim2real*, is a challenging problem and policies that are learned in simulation

[10]The DAVE model that we utilized has 1,595,511 trainable parameters. The multilayer perceptron is characterized by 4685 trainable parameters.

[11]The friction model used in the simulator is quite different from our hardware setup. This requires a longer reach-time to ensure safety. However, to maintain consistency across hardware and simulation experiments, we present the results with $T_{reach} = 1.0s$.

Table IV.2: Analysis of Wall-Time and Speed Variation (Without Obstacles): Simulation Platform

ML Controller	u (m/s)	$T_{runtime}$	MOET (ms)	Mean ET (ms)	Mean Iters	ML Usage (%)	PMD(%)
VBN	0.5	10	10.71 ± 0.97	7.00 ± 0.13	3.95 ± 0.03	94.52 ± 2.18	0.13 ± 0.15
		25	28.49 ± 5.23	15.42 ± 0.88	5.07 ± 0.12	93.64 ± 3.67	2.62 ± 2.71
	1.0	10	11.10 ± 1.25	6.85 ± 0.16	4.08 ± 0.23	77.15 ± 6.40	0.27 ± 0.24
		25	28.36 ± 0.77	14.86 ± 0.14	5.06 ± 0.10	79.97 ± 2.27	2.83 ± 1.64
	1.5	10	11.36 ± 0.60	6.83 ± 0.16	4.04 ± 0.14	50.56 ± 11.11	0.77 ± 1.13
		25	28.19 ± 0.53	15.00 ± 0.25	5.06 ± 0.03	55.59 ± 8.68	3.45 ± 1.59
LBC	0.5	10	10.78 ± 0.62	6.87 ± 0.10	3.93 ± 0.02	98.64 ± 2.14	0.12 ± 0.09
		25	28.31 ± 0.91	15.13 ± 0.22	5.03 ± 0.03	100.00 ± 0.00	2.40 ± 1.37
	1.0	10	10.68 ± 0.79	6.90 ± 0.09	3.91 ± 0.02	94.55 ± 6.40	0.12 ± 0.10
		25	28.69 ± 0.73	15.15 ± 0.26	5.06 ± 0.08	95.10 ± 7.49	3.56 ± 1.08
	1.5	10	15.03 ± 20.75	6.92 ± 0.18	3.97 ± 0.29	44.27 ± 16.79	0.27 ± 0.22
		25	28.46 ± 2.45	15.34 ± 0.76	5.08 ± 0.19	43.07 ± 17.04	1.87 ± 1.53
SAC	0.5	10	11.98 ± 3.16	6.44 ± 0.12	4.11 ± 0.24	50.66 ± 9.99	0.36 ± 0.30
		25	28.41 ± 2.72	15.17 ± 0.51	5.31 ± 0.32	49.98 ± 10.58	4.73 ± 1.30
	1.0	10	11.89 ± 3.36	6.63 ± 0.28	4.55 ± 0.47	12.16 ± 8.13	0.43 ± 0.34
		25	29.03 ± 6.27	14.49 ± 0.39	5.57 ± 0.55	10.11 ± 8.59	3.20 ± 1.96
	1.5	10	11.97 ± 1.89	6.42 ± 0.31	4.92 ± 0.63	8.31 ± 4.48	0.66 ± 0.43
		25	28.38 ± 2.63	14.17 ± 0.50	6.12 ± 0.62	9.42 ± 3.40	3.55 ± 1.64
ARS	0.5	10	17.24 ± 10.48	6.35 ± 0.12	4.69 ± 0.04	69.16 ± 8.29	0.35 ± 0.18
		25	30.33 ± 8.79	14.51 ± 0.41	5.79 ± 0.08	68.85 ± 9.67	4.62 ± 1.47
	1.0	10	20.07 ± 12.63	6.35 ± 0.16	4.91 ± 0.12	49.55 ± 4.05	0.96 ± 0.33
		25	34.27 ± 10.09	14.85 ± 0.47	5.86 ± 0.14	50.43 ± 3.65	4.71 ± 1.34
	1.5	10	12.15 ± 3.54	6.55 ± 0.22	4.08 ± 0.19	29.78 ± 5.56	0.53 ± 0.35
		25	27.83 ± 0.63	15.12 ± 0.38	5.15 ± 0.08	31.81 ± 4.48	4.53 ± 1.61

usually do not perform as expected in the real-world. Due to the risk of unsafe or catastrophic behavior, it is critical to monitor these components during operation.

In comparison to the simulation experiments, our experiments on the F1/10 hardware platform differed in two minor ways. (1) In order to analyze the *sim2real* performance of the controllers in isolation, we did not include obstacles within our experimental racetrack, shown in IV.6.[12] (2) We chose to offload the ML components to a separate computer. Compared to the other controllers, the VBN required a prohibitively large amount of the computation resources on the Jetson TX2. It is possible to optimize these ML models to run more efficiently on the Jetson platform, but we elected not to do so, as our primary concern was to analyze the variation of execution times of our safety regime on the embedded platform and the real world performance of each controller. Additionally, offloading the expensive ML computations to another computer allowed us to evaluate our regime with a finer level of granularity, providing a fairer comparison of the controllers.

We ran a total of 80 experiments, 5 for each controller, with the same structure as the simulation episodes.

[12] Due to the space constraints of our laboratory environment, such an evaluation would have prohibitively skewed our results.

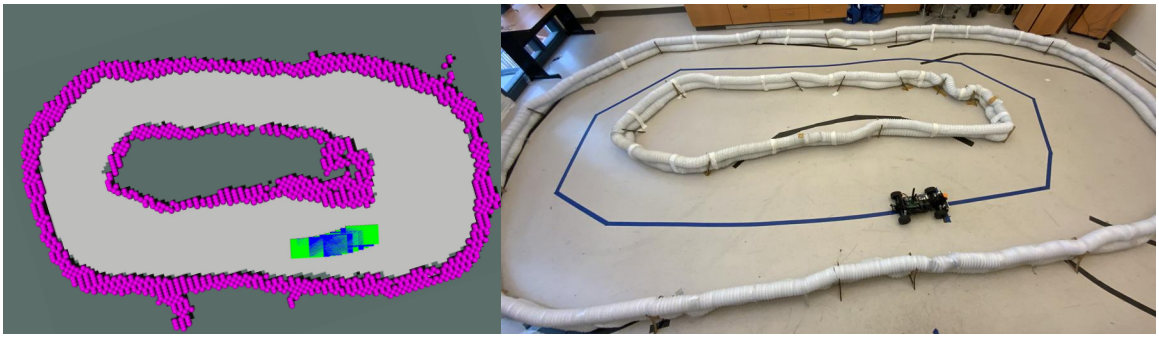

Figure IV.6: Visualization of the hardware experiments on the F1/10 Platform, the magenta points are the point-discretization of the wall boundaries (not all points are visualized). Videos of the experiments can be found at

A video of the experiments is available online.[13] The results of these experiments demonstrated that the ML models struggled to generalize to the real world and there was a large increase in the amount of time that the safety controller was used. This result can be explained by the small size of our racetrack, as well as the differences between the simulated and real world environments. On the other hand, the experiments demonstrate our regime was successful in maintaining safety, as there were no collisions observed in any of the hardware experiments.

With respect to the runtime performance of our approach, the experiments demonstrated that Mean ET of our regime fell well within our desired $T_{runtime}$. Though a few deviations were observed as displayed by the MOET, these deviations were minimal as displayed by percentage of missed deadlines shown in IV.3. These deviations can be attributed to our requirement that the safety checking process complete.

Finally, our hardware experiments demonstrated an average execution time that was significantly lower than the set wall-time. This result can be explained by the frequency of unsafe determinations and the nature of the reachability algorithm. Within the real-time reachability algorithm, the reachable-set computations terminate whenever an unsafe state is detected. The algorithm then restarts with half the step-size in order to refine the over-approximation error and determine if the "unsafe" declaration is spurious. This strategy continues until the step size falls below a pre-specified threshold specified to guarantee numerical stability. On the hardware platform, this threshold was met consistently, which demonstrates the frequency of unsafe declarations. Evidence for this observation is provided in Table IV.3. Experiments with higher levels of safety utilize fewer iterations in constructing the reachable set and generally have higher execution times. The numerical stability termination conditions are discussed in more detail in [106].

[13] **https://youtu.be/F42PF9ET7eA**

Table IV.3: Analysis of Wall-Time and Speed Variation: Jetson TX2

ML Controller	u (m/s)	$T_{runtime}$	MOET (ms)	Mean ET (ms)	Mean Iters	ML Usage (%)	PMD (%)
VBN	0.5	10	10.87 ± 0.36	5.07 ± 0.53	5.39 ± 0.83	11.70 ± 6.90	0.58 ± 0.29
		25	24.42 ± 0.49	10.41 ± 0.63	7.56 ± 0.97	12.38 ± 2.79	0.00 ± 0.00
	1.0	10	14.36 ± 5.74	5.56 ± 0.58	5.12 ± 0.39	9.31 ± 5.07	1.61 ± 1.18
		25	31.60 ± 17.81	9.36 ± 0.79	8.69 ± 1.16	9.97 ± 2.92	0.03 ± 0.06
LBC	0.5	10	24.49 ± 17.76	5.35 ± 0.45	4.65 ± 0.23	20.61 ± 7.43	0.78 ± 0.62
		25	31.31 ± 14.46	9.65 ± 0.51	6.49 ± 0.82	24.28 ± 7.63	0.07 ± 0.07
	1.0	10	14.25 ± 1.81	5.18 ± 0.36	4.50 ± 0.17	17.83 ± 4.75	0.75 ± 0.40
		25	24.43 ± 0.59	9.35 ± 0.74	7.35 ± 1.02	14.38 ± 6.18	0.01 ± 0.03
SAC	0.5	10	10.98 ± 0.33	5.10 ± 0.16	4.97 ± 0.54	16.44 ± 9.43	1.74 ± 0.80
		25	26.78 ± 2.57	10.91 ± 0.78	6.06 ± 0.72	26.17 ± 8.21	0.07 ± 0.04
	1.0	10	11.46 ± 0.47	4.73 ± 0.54	5.65 ± 2.17	12.58 ± 3.40	0.61 ± 0.18
		25	27.16 ± 6.38	9.69 ± 1.17	8.10 ± 0.91	9.82 ± 3.07	0.06 ± 0.10
ARS	0.5	10	13.92 ± 6.32	4.94 ± 0.30	5.63 ± 0.38	15.88 ± 1.78	0.46 ± 0.35
		25	30.83 ± 15.55	9.55 ± 0.53	7.00 ± 0.66	21.17 ± 6.56	0.02 ± 0.04
	1.0	10	11.53 ± 1.22	5.32 ± 0.20	4.93 ± 0.55	19.95 ± 6.65	1.13 ± 1.13
		25	30.58 ± 9.60	8.89 ± 0.72	7.50 ± 0.76	23.58 ± 3.16	0.05 ± 0.05

IV.9 Discussion

Having evaluated the merits of our approach both in simulation and on an embedded hardware platform, we now present some observations based on our results. In particular, we focus on real-time systems, challenges we faced moving from simulation to the real-world, and the main limitations of our approach.

IV.9.1 Real-Time Evaluation and Missed Deadlines

The basic requirement for real-time systems is that tasks operate within pre-defined and deterministic time spans. Often, this is accomplished through the use of a real-time operating system (RTOS), which allows for the specification of task priorities so that they are executed within established time frames. Our implementation did not make use of an RTOS, thus task management was left to the native Linux implementation. While our experimental evaluation did demonstrate deviations from the specified wall-time, the mean percentage of missed deadlines on the Jetson TX2 was fewer than 2% across all of our experiments.

In our hardware experiments, shown in Table IV.3, the percentage of missed deadlines (PMD) remained consistent when the $T_{runtime}$ was increased from $10ms$ to $25ms$. In contrast, the simulation experiments, shown in Table IV.2, displayed a greater deal of variation when the runtime was increased. The difference in results can be attributed to undesirable context switching.

Our simulation evaluations were conducted on a single machine running all the ROS infrastructure as well as the simulation software. To prevent all the nodes from missing their publishing deadlines, the simulation

utilizes a slower simulation time, which is distinct from the wall time we use for our reachability method. Using simulation time allows all nodes to meet their publishing deadlines and increases the amount of context switching going on. If the computation load is low, each node can finish its task without being halted. However, in our experiments, this was not the case. Thus, when $T_{runtime}$ is increased, there is a higher chance of the reachability computation being paused for something else to run. Consequently, this results in an increase in percentage of missed deadlines in the relevant experiments.

In our hardware experiments, we alleviated this issue by limiting the number of tasks running on the Jetson TX2 to only the essential ROS nodes. The machine learning computations were conducted on a separate machine to reduce the computational load. Thus, we reduce the amount of context switching that occurs during the reachability process.

In future work, we could further reduce the percentage of missed deadlines by increasing the conservativeness of the estimate or adding a computation resource focused only on the safety decision, similar to how we moved the NN computations to a separate machine. Alternatively, as all evaluation was done on regular operating systems (Linux with ROS) and not real-time operating systems (RTOS) that would provide hard deadline guarantees, we could perform an evaluation on such a platform to enforce deadlines strictly.

IV.9.2 Challenges Moving from Simulation to the Real World

Moving from simulation to the real-world hardware platform saw a significant expansion in the frequency that the safety controller was used. This increased reliance on the safety controller is a direct result of *sim2real* challenges in machine learning. Because the real world is inherently more noisy and more complex than the simulation environments that machine learning models are typically trained in, when these models are deployed in the real world, they are tasked with making generalizations on data that may significantly differ from their training context. Furthermore, this challenge is exacerbated by the reality of imperfect dynamic and sensor perception models [228]. However, the main challenge that we faced during the transition from simulation to was related to the size of the track in our laboratory setup.

The track shown in IV.1 is much larger than the real world track we tested on, shown in IV.6. At the narrowest point, the simulated track is $2m$ wide.[14] In contrast, the widest part of our real-world track is scarcely larger than that. The bulk of the racetrack has a width of less than one meter. Since $T_{runtime}$ is constant across the simulation and real world experiments, the real world vehicle spends the majority of its time forecasting unsafe actions due to its close proximity to the track walls. The hardware experiments were quite illustrative in motivating our desire to extend the current methodologies to integrate closed-loop reachability techniques.

[14] Races held by the F1/10 community are around typically 30-50 meters in length with a minimum of 3 meters between the racetrack walls. The rules are described in more detail here: https://f1tenth.org/misc-docs/rules.pdf

Our current assumption that the control decision remains fixed throughout the reachset construction is quite conservative. Closed-loop reachability techniques would allow us to weaken this assumption and compute approximations of the real inputs during the reachset construction. However, the difficulty in performing closed-loop reachset generation lies in developing accurate sensor models. As an example, for the VBN it is not clear how to generate camera images based on the state of the system so as to provide a meaningful and useful reachable set.

IV.9.3 Environment-Induced Noise

Noisy sensor data is expected because no sensor is always perfectly accurate. However, environment-induced noise refers to additional noise that exists because of differences in the environments. This is best highlighted in the case of the RL controller. The environment the agent trained in had smooth, solid walls. Meanwhile, our track is made out of a series of flexible vinyl duct piping hose segments[15], which contain ridges and occasionally expose gaps in between the stacked structure.

While it is possible to try and model these variations in the simulator, this was not within the scope of our study. However, our future work will explore the relationship between simulator fidelity and the safety of each controller. Since the controllers are ”black box” in nature, we cannot know for sure how the policies will respond to these variations. Despite this challenge, we were able to get the F1/10 vehicle to successfully complete laps in the real world because of our simplex architecture. The overall performance might have been reduced, but we were able to ensure safe operation.

IV.9.4 Limitations

While the reachability algorithm presented in this work possesses provable guarantees, our architecture does not. Obtaining these guarantees requires developing a formally verified safety controller and switching logic, which was outside the scope of the work presented herein. Therefore, it is possible to enter a state in our framework in which all future trajectories will result in a collision. These states are known as *inevitable collision states* and have been well-studied within the motion planning literature [232]. In future work, we hope to address this limitation by leveraging approaches such as viability kernels, and dynamic safety envelopes that allow for the synthesis of provable safe control regimes [233].

Finally, as with many model based approaches, the quality of our reachability computations is dependent on the quality of the underlying model of the physical system. That is, guarantees around safety are only valid, provided that the system model is a good representation of reality. While the system identification results, presented in Section IV.3, show that our model is reasonably accurate, in practice the implementation

[15] https://www.amazon.com/dp/B01G198P8W?psc=1

of such an approach would also need to incorporate a rigorous uncertainty quantification analysis of the underlying model within the presence of imprecise sensor and state information [234]. However, it is worth noting the underlying reachability algorithm supports differential inclusions, allowing for the straightforward incorporation of uncertainty, provided that such an analysis has been done.

IV.10 Comparison to Other Approaches

As cyber-physical systems have become increasingly more complex in recent years, there has been a major thrust to develop techniques capable of assuring their safety at at runtime. This has led to the rise of *simplex*, *runtime assurance*, and *runtime-verification* strategies, and it is within this context that we present our work.

The simplex architecture has been used widely in the research literature to provide guarantees for systems with unverified components. The contexts in which it has been applied include aerospace systems [110], fleets of remote controlled cars [111], industrial embedded infrastructure [13, 28], and distributed mobile robotics applications [112, 104]. In [235], the authors utilize a reachability regime to guarantee the safety of an autonomous vehicle that makes use of a reinforcement learning controller for a way-point-following task. Most similar to this work in [104], the authors utilize a real-time reachability approach to verify that a group of quadcopters executing a distributed search mission is free from collisions. Their approach is implemented in simulation and can theoretically deal with over 64 quadcopters. These works primarily deal with developing provably correct motion planners, while the focus of this work is abstracting away the underlying nature of a set of ML components and reasoning about the consequences of their actions on overall system safety.

Table IV.4: Comparison of Online Reachability and Monitoring Methods

Tool Name	Real-Time Guarantee	External Libraries	Evaluation Platforms	ML Monitoring	Online Reachability
Rtreach-ML (this chapter)	✓	×	x86,ARM	✓	✓
Rtreach [106]	✓	×	x86, ARM, AVR	×	✓
BACH2 [236, 237]	✓	✓	x86	×	✓
ReachFlow (Flow*) [235]	×	✓	x86	✓	✓
PHaVer [238]	×	✓	x86	×	✓
FaSTrack [239]	×	✓	x86	×	×
ARMTD [240]	✓	✓	x86	×	×
CORA [48]	✓	✓	x86	×	✓
SafeTrafficWeaving [241]	✓	✓	x86	×	×
ROSRV [223]	×	✓	x86	×	×
SOTER [112, 242]	×	✓	x86	×	×
Modelplex/VeriPhy [84]	×	✓	x86, ARM	×	×
Black-Box Simplex [109]	×	✓	x86	✓	✓
FPGA HJR [243]	✓	✓	x86, ARM	×	×

The simplex architecture has been used widely in the research literature to provide guarantees for systems with unverified logic. The contexts in which it has been applied include aerospace systems [110], fleets of remote controlled cars [111], industrial embedded infrastructure [13, 28], and distributed mobile robotics applications [112, 104]. In [235], the authors utilize a reachability regime to guarantee the safety of an autonomous vehicle that makes use of a reinforcement learning controller for a way-point-following task. Most similar to this work in [104], the authors utilize a real-time reachability approach to verify that a group of quadcopters executing a distributed search mission defined by a series of waypoints is free from collisions. Their approach is implemented in simulation and can theoretically deal with over 64 quadcopters. These works primarily deal with developing provably correct motion planners, while the focus of this work is abstracting away the underlying nature of a set of ML components and reasoning about the consequence of their actions on overall system safety.

Closely related to simplex techniques are run-time assurance (RTA) methods and run-time verification (RV) methods. An intuitive explanation of these regimes can be summarized as follows. RTA techniques are tasked with ensuring the safe operation of systems with untrusted components, while run-time verification techniques monitor a system against presupposed formal properties at run-time [79, 80, 81, 82, 83, 84, 85, 86]. The distinction here is that while run-time assurance techniques may often utilize verification results, they may often also employ statistical techniques such as anomaly detection [78] or simulation based strategies [29]. One such example is the work by Allen et al.[105], in which they utilize various machine learning techniques, mainly support vector machines and linear regression models, to approximate the solution to the real-time reachability problem. Their work is capable of being applied in low-resource, real-time environments but suffers from the downside that theoretical guarantees cannot be made using these techniques, only statistical ones. However, it demonstrated impressive results in improving state-of-the-art execution times by four orders of magnitude.

Finally, in recent years, researchers have begun to integrate traditionally non real-time approaches within real-time systems, making these approaches more amenable to real-time execution. These include viability kernel approaches that determine if a set of states remain within a predefined region [93, 48], as well as Hamilton-Jacobi (HJI) reachability techniques that deal with dynamical systems with general nonlinear dynamics and disturbances in uncertain environments [94, 96, 97, 98, 102, 103]. Specifically, [96] and [48] were able to implement their techniques both in simulation and on hardware platforms. However, both papers used their respective real-time reachability results for safe real-time motion planning, while the work contained in this manuscript, primarily dealt with the construction and implementation of the simplex architecture.

Table IV.4 presents several state-of-the-art online reachability tools present within the literature, and summarizes their amenability to real-time operation. What distinguishes the methods presented in this work, is

that they possess real-time guarantees and have been extended from [106] for the monitoring of machine learning models. Particularly, our work is to the best of our knowledge the only work that deals with perception controllers. We realize that there are numerous other interesting works present within this area, and to the best of our knowledge the aforementioned works are the most relevant to the methods presented within this manuscript.

IV.11 Conclusions and Future Work

This chapter presents a simplex architecture for the safety assurance of a 1/10 scale open source autonomous vehicle platform known as the F1/10. The approach relies on a real-time reachability regime that is used to provide guarantees of safety and detect potential unsafe scenarios during operation. One of the motivations in utilizing real-time reachability is that it abstracts away the need to analyze the underlying controller and instead focuses on the effects of control decisions on the system's future states. Our experiments conducted both in simulation and on an embedded hardware platform validate the real-time aspects of our approach. Moreover, they demonstrate the efficacy of the simplex architecture in ensuring safety in different scenarios. Improving the over-conservativeness of the reachability framework, considering closed-loop reach-set generation, making use of real-time operating systems, and incorporating dynamic obstacles into our regime are left for future work

IV.12 Summary of Contributions

In summary, the contributions of this chapter are as follows:

- We modify the real-time reachability algorithm presented in [108] to handle static obstacles in a sound and real-time manner.
- We implement a simplex control architecture that uses real-time reachability for online collision avoidance.
- We show our method working with multiple machine learning controllers.
- We demonstrate success using our method to safely navigate through obstacles the trained controllers have no prior experience with.
- We evaluate the safety of machine learning components transferred to real-world hardware without additional training.

CHAPTER V

An Empirical Analysis of the Use of Real-Time Reachability for the Safety Assurance of Autonomous Vehicles

This chapter is adapted from the material presented in [244].

Recent advances in machine learning technologies and sensing have paved the way for the belief that safe, accessible, and convenient autonomous vehicles may be realized in the near future. Despite tremendous advances within this context, fundamental challenges around safety and reliability are limiting their arrival and comprehensive adoption. Autonomous vehicles are often tasked with operating in dynamic and uncertain environments. As a result, they often make use of highly complex components, such as machine learning approaches, to handle the nuances of sensing, actuation, and control. While these methods are highly effective, they are notoriously difficult to assure. Moreover, within uncertain and dynamic environments, design time assurance analyses may not be sufficient to guarantee safety. Thus, it is critical to monitor the correctness of these systems at runtime. One approach for providing runtime assurance of systems with components that may not be amenable to formal analysis is the simplex architecture, where an unverified component is wrapped with a safety controller and a switching logic designed to prevent dangerous behavior. In this chapter, we propose using a real-time reachability algorithm for the implementation of the simplex architecture to assure the safety of a 1/10 scale open source autonomous vehicle platform known as F1/10. The reachability algorithm that we leverage (a) provides provable guarantees of safety, and (b) is used to detect potentially unsafe scenarios. In our approach, the need to analyze an underlying controller is abstracted away, instead focusing on the effects of the controller's decisions on the system's future states. We demonstrate the efficacy of our architecture through a vast set of experiments conducted both in simulation and on an embedded hardware platform.

V.1 Introduction

For decades, the vision of deploying autonomous vehicles ubiquitously has enraptured technology enthusiasts, researchers, and corporations. The prevailing conviction is that there are relatively few technologies that hold as much promise as autonomous vehicles (AVs) in bringing about safe, accessible, and convenient transportation. Despite demonstrated success through efforts such as the Defense Advanced Research Projects Agency's (DARPA) Grand Challenges [245, 246], and the emergence of high profile autonomous vehicle

companies such as Alphabet's Waymo, Argo AI, Aptiv, Zoox, General Motors' Cruise, Tesla, Aurora, and Intel Corporation's Mobileye, the consensus remains that there are serious technical and safety challenges to be resolved.

The two fundamental challenges widely regarded as limiting the arrival and widespread adoption of AVs are safety and reliability [217]. Reasoning about safety requires an understanding of the joint dynamics of computers, networks, and physical dynamics in uncertain and variable environments, making it a notoriously difficult problem [8]. To handle the complexities of their environments, many AVs make use of *Machine Learning* (ML) components such as artificial neural networks to decipher the information observed from an ever-evolving configuration of on-board sensors [8]. However, despite the impressive capabilities of these components, there are reservations about using them within safety critical settings. This is primarily due to the difficulty of interpreting the inner workings of these models, which prevents any meaningful explanation of their behavior from being made.

Utilizing a model that lacks transparency with respect to how its decisions are made within a system that is safety critical, constitutes the highest form of technical debt [218] and, as a result, the last several years have witnessed a significant increase in developing methods that seek to reason about the safety and robustness of machine learning methods [247, 114, 9]. Unfortunately, while numerous works have been proposed over the past few years for the formal analysis of machine learning methods, the vast majority of these efforts have not been able to scale to the complexity found in real world applications, where models, such as neural networks, may contain millions or even billions of parameters [248, 121]. Further exacerbating this challenge are the unanticipated environmental conditions that cannot be captured at design time. Testing, while rather effective, is also infeasible, as this requires a prohibitively large amount of tests to be executed in order to demonstrate a sufficient amount of reliability [32].

Since design-time testing and formal analysis are not sufficient alone to demonstrate the safety of complex systems, verifying systems at runtime is often required. These regimes are classically referred to as *runtime verification* or *runtime assurance* approaches, and they broadly consist of observing the execution of a system at operation-time and checking whether relevant safety properties are preserved. The system model under consideration may take on many forms, including models of the physical dynamics of the system or even models of the underlying software governing its behavior. These models are then used to consider lightweight yet rigorous considerations of presupposed formal properties [79, 80, 81, 82, 249].

A crucial question that must be answered when a runtime assurance approach to verification is used, is what happens when a problem is discovered by a monitor [90]. Many runtime verification approaches classically involve the ability of invoking recovery actions in response to safety or property violations. Within this context, one of the most popular runtime assurance architectures is the *Simplex Architecture*, and it has

demonstrated significant success in enabling the assurance of systems with components that may be too complex, or too large, for complete design-time analysis [219]. This makes this regime particularly attractive for the assurance of machine learning and AI-based components. In this framework, an unverified component is wrapped with a safety controller and a switching logic designed to transfer control to the safety controller in the event of property or safety violations [108]. A useful analogy for this architecture is a driving instructor's car with two steering wheels and two sets of brakes. As long as the instructor is capable of intervening in dangerous situations, the capricious student is allowed to drive [108].

Typically, in simplex architectures, the switching logic is primarily designed either from a control theoretic perspective through the solution of linear matrix inequalities (LMI) [110], or using a formal analysis hybrid-systems reachability technique [13]. As Bak et al. note, it is easy to design a safe decision logic; one can simply always use the safety controller [108]. However, this is unsatisfactory since the performance responsibilities of the system might be forfeited or unreasonably delayed [108]. The key challenge in this regime is to design a switching logic that allows the dynamic capabilities of the unverified complex controller to be exploited as much as possible without compromising safety.

In this chapter, we extend the real-time reachability algorithm from [108, 106] to design a simplex architecture for a 1/10 scale autonomous racing car called the F1/10 platform. The central idea behind our framework lies in computing the set of reachable states of the F1/10 system and ensuring that it never enters unsafe states as it navigates an environment. Specifically, this entails checking that the vehicle's trajectories are free from collisions with both static and dynamic obstacles within its environment. Rather than performing an analysis of the underlying controller governing the behavior of the system, the crux of our approach lies in monitoring the influence of a controller's decisions on the overall evolution of the system by reasoning about the set of reachable states over a finite-time horizon. This set can then be used to forecast potential collisions. Thus, this safety checking procedure forms the basis of the switching scheme in our simplex architecture. In our work, the nature of the underlying controller is immaterial, and our experiments consider a variety of control strategies ranging from machine learning, path tracking, and gap following regimes.

One of the key benefits of utilizing reachability analysis to construct our simplex architecture is that reachability analysis is quite adept at handling uncertainty. This makes the approach particularly attractive for autonomous systems, whose correct operation is dependent on an effective treatment of uncertain sensor measurements, predictions about the behavior and intent of dynamic environmental participants, and the modeling assumptions defining its control regimes. Bearing the above in mind, reachability techniques can be used to compute *all* the possible states that a system may attain from large bounded sets of initial conditions, disturbances and system parameter variations [39]. Thus, as Asarin et al. note in their work, such an analysis "provides knowledge about the system with a completeness or coverage that a finite number of simulations

cannot deliver," [39]. We evaluate the merits of using reachability methods for the safety assurance of the F1/10 platform both in simulation and on an embedded hardware platform using a variety of controllers, number of obstacles, and runtime configurations. Furthermore, we present an analysis of the effects of various sources of uncertainty on the conservativeness of our safety regime. Finally, we also present a robust runtime characterization of the real-time reachability regime that forms the basis of our work.

V.1.1 Statement of Contributions

The contributions of this article can be summarized as follows. (1) We present a runtime verification technique that abstracts away the need to analyze the nature of the underlying controller governing the behavior of our system, and instead focuses on the effects of the control decisions of these controllers on the overall system during operation. This is accomplished by obtaining the set of reachable states of the system over a finite time horizon and checking for potential collisions with objects within the environment. The approach presented in this chapter has the ability to reason about safety in the presence of both static and dynamic obstacles. (2) Leveraging the reachability framework, we implement a simplex control architecture in order to maintain safety during operation. (3) We present a safety analysis of a diverse set of controllers through a series of experiments with varying speeds and number of opposing vehicles in order to explore the tradeoffs of our approach. (4) We present a rigorous empirical analysis, accounting for various classes of uncertainty within our regime. Our analysis includes a study of the effects of uncertainty on the conservativness of our safety regime. (5) Finally, we present a runtime characterization of the real-time reachability regime, enabling our work over a broad set of experiments.

A preliminary version of this work appeared in Chapter IV. In this enhanced and extended version, we provide the following additional contributions (1) An extension of our safety framework to account for dynamic obstacles, (2) further commentary on the physical dynamics models used within our approach and a deeper discussion of handling uncertainty, (3) an empirical study of the effects of uncertainty on our overall safety regime, and (4) additional experimental results including an evaluation of our approach in contexts where two 1/10 scale embedded open-source autonomous vehicle testbeds interact within a racing context.

V.2 Related Work

The increasing ubiquity of software in numerous domains, particularly in safety critical domains, has heightened the need to ensure the correct and reliable operation of deployed software. Over the last several years, there has been a wealth of approaches proposed towards proving the correctness of autonomous systems prior to fielding them in their respective operational design domains [25, 26, 27, 28, 29, 30]. However, very few approaches exist that can provide strict formal guarantees about their behavior, due to the challenges associ-

ated with reasoning about large and complex systems that are tasked with operating in dynamic and uncertain environments. Moreover, these systems often leverage machine learning components to deal with the diverse data obtained from the system's sensors. Applying formal assurance techniques to machine learning systems has only been considered recently and poses unique challenges [250]. While there has been significant progress within this realm, there is still a significant gap between the machine learning models that these approaches can handle, and the models deployed in state-of-the-art systems.

In a similar vein, while significant efforts have been devoted to developing approaches that can deal with the complexities of autonomous systems, only a few of them can be leveraged at runtime [251, 236, 237, 235, 238, 96, 240, 48, 241, 223, 112, 242, 84]. The motivation for utilizing runtime assurance approaches stems from a recognition that in certain environments, complete or even partial verification may be infeasible due to the well-known state-explosion problem [24] and the reality that the complete analysis of certain components may be infeasible at design time. As an example, for an autonomous vehicle, it is imperative that collisions are avoided while the system carries out its high-level goals. This requires monitoring the vehicle's state during operation, as design time considerations cannot feasibly consider all the possible scenarios that the system may encounter [61]. In light of these challenges, this has led to the rise of *runtime assurance*, *runtime-verification* and *simplex* strategies, and it is within this context that we present our work.

Runtime assurance (RTA) and runtime verification (RV) methods broadly consist of techniques that allow for observing the execution of a system at runtime and checking whether relevant correctness properties are preserved. An intuitive explanation of these regimes can be summarized as follows. RTA techniques are tasked with ensuring the correct operation of systems with untrusted components, while runtime verification techniques monitor a system against presupposed formal properties at runtime [79, 80, 81, 82, 83, 84, 85, 86]. The distinction here is that while runtime assurance techniques may often utilize verification results, they may often also employ statistical techniques such as anomaly detection [78] or simulation based strategies [252]. One such example is the work by Allen et al.[105], in which they utilize various machine learning techniques, mainly support vector machines and linear regression models, to approximate the solution to the real-time computation of the set of reachable states of an underlying system. Their work is capable of being applied in low-resource, real-time environments, but suffers from the downside that theoretical guarantees cannot be made using these techniques. Only statistical guarantees may be obtained. However, the approach demonstrated impressive results in improving state-of-the-art execution times in the assurance process by four orders of magnitude.

Within the context of runtime-assurance approaches, the simplex architecture has been used widely in the research literature to provide guarantees for systems with unverified components. The contexts in which it has been applied to include aerospace systems [219, 110, 249], fleets of remote controlled cars [111], in-

dustrial embedded infrastructure [13, 28], and distributed mobile robotics applications [112, 104]. In [235], the authors utilize a reachability regime to guarantee the safety of an autonomous vehicle that makes use of a reinforcement learning controller for a way-point-following task. Most similar to this work in [104], the authors utilize a real-time reachability approach to verify that a group of quadcopters executing a distributed search mission is free from collisions. Their approach is implemented in simulation and can theoretically deal with over 64 quadcopters. These works primarily deal with developing provably correct motion planners, while the focus of the work presented in this chapter, is abstracting away the nature of an underlying controller, and instead reasoning about the consequences of its actions on overall system safety.

Finally, in recent years, researchers have begun to integrate traditionally non-real-time verification approaches within real-time systems by making these approaches more amenable to real-time execution. These include viability kernel approaches that determine if a set of states remain within a predefined region [93, 48], as well as Hamilton-Jacobi reachability (HJR) techniques that can deal with dynamical systems with general nonlinear dynamics in uncertain environments [94, 96, 97, 98, 102, 103]. One of the major benefits of Hamilton-Jacobi reachability is that it allows for the specification of an optimal control problem characterized by a differential game, where a controller must maintain system safety under the influence of disturbances [253]. Therefore, this framework allows for a robust analysis of safety under uncertainty. Specifically, [96] and [83] were able to implement these techniques on both simulation and hardware platforms. However, both papers used their respective reachability results for safe motion planning, and their approach does not possess rigorous real-time guarantees. The work contained in this chapter, however, primarily deals with the construction and implementation of a safety assurance architecture based through use of a real-time reachability regime.

What distinguishes the methods presented in this article is that they possess real-time guarantees and have been extended from [106] to consider the safety of a set of controllers under varying levels of uncertainty. Moreover, this work considers the safety question in the presence of dynamic and static obstacles and is evaluated over a wide range of experiments. We realize that there are numerous other interesting works present within this area, and to the best of our knowledge, the aforementioned works are the most relevant to the methods presented within this chapter.

V.3 Preliminaries

V.3.1 The Simplex Architecture

As modern autonomous systems grow in complexity, so do the challenges in assessing their reliability and correctness [13]. Moreover, any arguments about the reliability and safety of the system rely on assertions about the individual components that make it up [108]. However, in recent years, with the growth of increas-

ingly autonomous systems [220], individual components may be designed using machine learning methods, such as neural networks, that are opaque to traditional formal analysis. Despite the recent years' surge in the development of formal analysis techniques for these types of models [114, 9], most techniques are incapable of dealing with the scale of models deployed in state-of-the-art systems.

One paradigm for dealing with untrustworthy components is the *simplex architecture* [219, 108, 110]. In the simplex architecture, the unverified component, or *complex controller*, is wrapped with a *safety controller* and a switching logic used to ensure safety [108]. A useful analogy for this architecture is a driving instructor's car with two steering wheels and two sets of brakes. As long as the instructor is capable of intervening in dangerous situations, the capricious student is allowed to drive. Typically, the complex controller has better performance with respect to the design metrics, whereas the safety controller is designed with simplicity and verifiability in mind. Thus, by using this architecture, one can utilize the complex controller while still maintaining the formal guarantees of the safety controller. The key challenge when designing a system with the simplex architecture is properly designing the switching logic [106]. One must be able to clearly delineate safe states from unsafe states.

In a typical implementation of the simplex architecture, the switching logic is primarily designed either from a control theoretic perspective through the solution of *Linear Matrix Inequalities* (LMI) [110], or using a formal analysis hybrid-systems reachability technique [13]. In this chapter, our simplex design requires computing the set of reachable states online through the use of a real-time reachability algorithm for short time horizons.

V.3.2 Reachability Analysis

Reachability analysis is a model checking technique that involves rigorously computing the set of all states that a system can attain over a finite time horizon, and it is commonly used as a method for demonstrating that a system satisfies relevant safety properties [39]. One of the major strengths of these approaches is they are able to provide knowledge about an underlying system with a level of completeness that a finite number of simulation analyses cannot deliver [39]. Primarily, because the set of reachable states obtained using these approaches can describe the system's trajectories from all possible initial conditions, and under all admissible disturbances and variations in the parameter values of the underlying model [39]. In deriving such a set, the safety assurance problem often consists of determining whether there is an intersection between the reachable set of a system and a set of *undesirable states*. As an example, for an autonomous vehicle this analysis can be leveraged to investigate whether the vehicle remains within lane boundaries, and if static and dynamic obstacles are avoided as the vehicle navigates within its environment [48].

Generating the set of reachable states involves a combination of numerical analysis techniques, graph

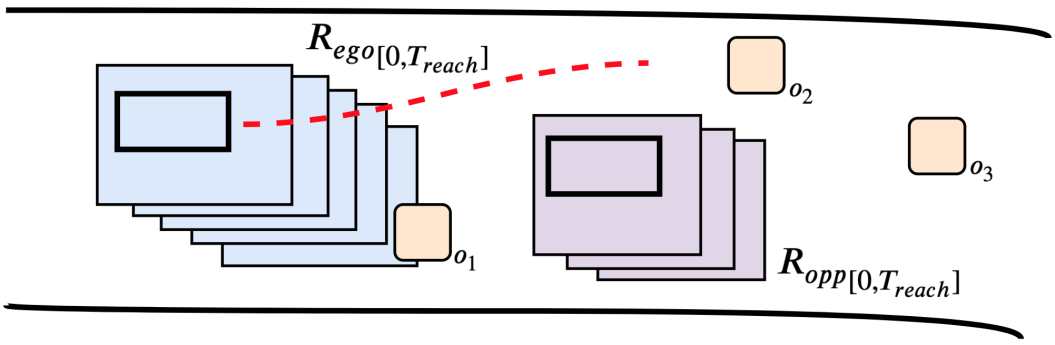

Figure V.1: Overview of our runtime safety assurance framework. In this figure, the blue rectangles correspond to the reachable set of the ego vehicle, while the purple rectangles correspond to the reachable set of a dynamic opponent. Static obstacles are shown in orange, and the racetrack boundaries are the curved solid black lines. The red dotted line corresponds to the trajectory that would be obtained through the exclusive use of the safety controller. In the above figure, the reachable set of the ego vehicle, $R_{ego_{[0,T_{reach}]}}$, projects the effects of using a control action issued by the complex controller leveraged by the system, while the reachable set of the dynamic opponent is obtained by assuming that the opponent vehicle will maintain its velocity and direction over a short time horizon $R_{opp_{[0,T_{reach}]}}$. If the reachable set of the ego vehicle intersects with any obstacle, o_1, in the environment, or with the reachable set of an opponent vehicle, then our simplex approach switches to using a safety controller optimized to avoid collisions (red trajectory).

algorithms, and computational geometry [39, 51] and there is a rich set of literature and software tools available for the reachability analysis of systems with continuous, discrete, and hybrid dynamics. While, in this article, we confine our focus to those with continuous dynamics, the reachability analysis of discrete systems has been extensively considered since the early 1960s and is a well studied problem [49, 50]

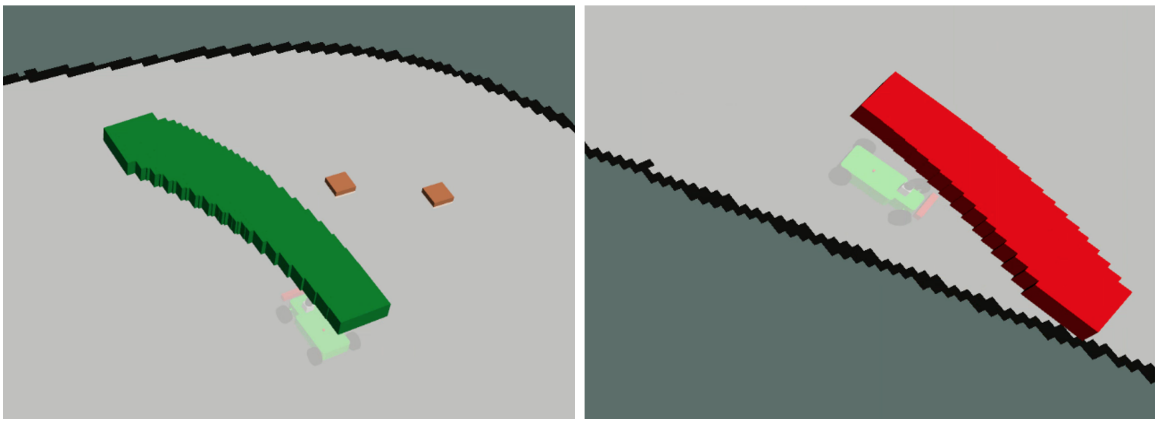

Figure V.2: Visualization of the set of reachable states derived by projecting a control action forward over a finite time horizon in our simulation evnironment. For illustration purposes, we display only a subset of the hyper-rectangles in the above images. *Left*: (green boxes) Example of an action labeled as safe, since there are no intersections between the reachable set and obstacles in the vehicle's environment or the racetrack walls (black). *Right:* (red boxes) This example corresponds to an unsafe scenario, as following the issued control action would result in a collision between the vehicle and the racetrack boundaries. In the above images, the orange squares represent the location of cones and their corresponding bounding box.

Traditionally, reachability methods have been executed offline, at design-time, because they are computationally intensive endeavors [52, 39, 58]. However, in [108, 106], Bak et al. and Johnson et al. present a reachability algorithm based on the seminal mixed face-lifting algorithm [221], capable of running in real-time on embedded processors. The algorithm is implemented as a standalone C-package that does not rely on sophisticated (non-portable) libraries, recursion, or dynamic data structures and is amenable to the anytime computation model in the real-time scheduling literature. In this regime, each task produces a partial result that is improved upon as more computation time is added [106]. The standalone nature of this approach allowed us to utilize this scheme in implementing our safety architecture on an embedded hardware platform.

V.3.3 Safety Architecture

The controllers in our experiments are designed to sample sensor data and compute control actions at fixed time intervals, as typically done in the control community. During each control period, we take the corresponding control action and compute the reachable set of states into the future as defined by the current state, assumptions around disturbance and uncertainty, and a specified finite-time horizon. An example of this computation is shown in V.2. We assume a fixed control action throughout the reachable set computation.[1] Based on the obtained reachable set, we determine if the system will collide with objects in its environment and, if necessary, switch to a safety controller optimized for obstacle avoidance. If the system falls back to using a safety controller, we only allow a switch back to the complex controller if the complex controller has demonstrated safe behavior for a fixed number of control periods.[2] This prevents arbitrary switching and incorporates a sense of hysteresis into our control strategy. Additionally, by not switching back until consistently safe behavior has been demonstrated, we enforce a notion of dwell time, which reduces instabilities caused by switching too frequently. An overview of our approach is shown in Figure V.1.

V.3.4 Handling Uncertainty

As with all model-based approaches, the quality of our safety declarations, and thereby the results of our reachability computations, are highly dependent on the quality of the models of the underlying system and environment. It is imperative that our derived model is a good representation of reality. Otherwise, any predictions about the behavior of our system may be invalid. The reality, however, is that deriving good representations of each of these elements is quite challenging [254]. Developing an exact model of a system, for example, is extremely difficult due to the presence of complex physical interactions that may be hard to describe precisely [253]. These interactions include phenomena such as drag forces, non-observable exter-

[1] We discuss the merits of this assumption in Section V.7.

[2] In our experiments, we allowed a switch back to the safety controller after 30 control periods. This corresponds to 1.5 seconds using a $20Hz$ control period.

nal disturbances, friction, non-deterministic model parameters, non-observable states, and other stochastic elements [253].

In many cases, things that cannot be modeled explicitly are often aggregated together as uncertainty, and rigorous analyses aimed at quantifying levels of uncertainty in the underlying model are frequently conducted [253, 255]. Sources of uncertainty within a system can broadly be classified into two categories, *Aleatoric Uncertainty*, and *Epistemic Uncertainty*. Aleatoric uncertainty, also referred to as irreducible random uncertainty, is characterized by the natural variation of physical systems due to random effects [256]. Epistemic Uncertainty, however, refers to systematic uncertainty, that is attributed to a lack of knowledge of the dynamics of the system [257]. As an example, epistemic uncertainty can be attributed to a lack of knowledge of how to model quantities that are hard to measure or represent, and unlike aleatoric uncertainty, it can be reduced by more comprehensive experimentation and modeling [257].

Before making use of a model within model-based design approaches, it is imperative to identify and define assumptions about all the known sources of uncertainty in the system, and if through rigorous analyses one can obtain bounds on the uncertainty associated with a model's parameters, then one can conduct exhaustive analyses of the behavior of the system under a large set of bounded parameter variations and initial conditions [32]. In doing so, the system designer can gauge whether the system satisfies the required levels of consistency, robustness, and quality governed by its requirements under the assumptions about the underlying levels of uncertainty [255].

While stochastic simulation techniques, such as the Monte Carlo Paradigm, have allowed for efficient explorations of the parameters of formal models, the number of simulations needed to gain full confidence in an underlying system is prohibitively large [32]. As Beg et al. note, in general, the total number of Monte Carlo Simulations aimed at matching a model to experimental data would need to be increased one hundredfold to achieve an additional decimal place of precision. Thus, theoretically, having total confidence in the results of Monte Carlo analyses would require an infinite number of simulations [32].

Within this context, set-based reachability regimes have displayed significant success in rigorously capturing the behavior of a system under a large set of initial conditions, disturbances, and system parameter values [51]. Thus, reasoning the correctness of the model can be reduced to verifying whether the set of all reachable states satisfy key properties across all possible model parametrizations [32].

V.4 Problem Formulation

This section reviews modeling dynamical systems, and then presents this work's main contribution, a runtime assurance framework leveraging real-time reachability and the simplex architecture.

V.4.1 System Dynamics Model

Suppose the dynamics of the system can be described by an ordinary differential equation (ODE) of the form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{d}) \tag{V.1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}^n$ describes the dynamics of the system, $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{u} \in \mathbb{R}^m$ is input to the system, and $\mathbf{d} \in \mathbb{R}^n$ is a disturbance input. Assuming that f is globally Lipschitz continuous, a solution to (V.1) describing the evolution of the system with initial condition $\mathbf{x}_0 \in \mathbb{R}^n$, initial input $\mathbf{u}_0$, and disturbance $\mathbf{d}_0$, is any differentiable function $\psi(t)$, where $\psi\colon \mathbb{R}^+ \to \mathbb{R}^n$, such that $\psi(0) = \mathbf{x}_0, \mathbf{u}_0, \mathbf{d}_0$ and $\dot{\psi}(t) = f(\psi(t))$ [258]. Under our Lipschitz assumptions, the solution to the above differential equation is unique.

Suppose now that we wish to consider a family of solutions for the dynamics of a particular system, in order to characterize the uncertainty in the underlying model. In this realm, we can formulate the dynamics as a *differential inclusion*. A differential inclusion can be written as:

$$\dot{\mathbf{x}} \in F(\mathbf{x}) \tag{V.2}$$

where F is a set-valued map from $\mathbb{R}^n$ to $\mathbb{R}^n$. That is $F(\mathbf{x}) \subset \mathbb{R}^n$. Maintaining our Lipschitz assumptions, a solution to (V.2), with initial condition $\mathbf{x}_0 \in \mathbb{R}^n$, initial input $\mathbf{u}_0$, and disturbance $\mathbf{d}_0 \in \mathbb{R}^n$ is any differentiable function $\psi_1(t)$, where $\psi_1\colon \mathbb{R}^+ \to \mathbb{R}^n$, such that $\psi_1(0) = \mathbf{x}_0, \mathbf{u}_0, \mathbf{d}_0$ and $\dot{\psi}_1(t) \subset F(\psi(t))$ [258]. Whereas in (V.1), the solution to the ODE had a unique solution, in this realm a differential inclusion has a family of solutions.

Bearing the above in mind, many classes of uncertainty, such as environmental uncertainty and modeling discrepancies, can be modeled as differential inclusions [259]. In this realm, one way of describing the uncertainty with respect to an underlying model is allowing the state, input, and parameters to be described by sets. As an example, if we assume that we can obtain bounds on the set of disturbances D, that represent all the things that are not explicitly modeled by f, then the behavior of the system resulting from interactions with any admissible disturbance can be rigorously obtained via the solution of the differential inclusion with $\mathbf{d} \in D$ [259].

V.4.2 Online Reachability Computation

Before outlining the reachability framework leveraged in our work, let us define two key terms.

Definition 5 (REACHTIME) *The reachtime, T_{reach}, is the finite time horizon for computing the reachable set.*

Definition 6 (RUNTIME) *The runtime, $T_{runtime}$, is the duration of (wall) time given for constructing the reachable set.*

With that, deriving the set of reachable states for an underlying system works as follows. Assuming a dynamics model for the system, we utilize the mixed face-lifting algorithm proposed in [260], to compute the set of reachable states from the current time t up until $(t + T_{reach})$. The mixed face-lifting approach utilized here is part of a class of methods that deals with *flow-pipe construction* or *reachtube computation* [106]. This is done using snapshots of the set of reachable states that are enumerated at successive points in time. To formalize this concept, we define the reachable set below.

Definition 7 (REACHABLE SET) *Given a system with state vector $\boldsymbol{x} \in \mathbb{R}^n$, input vector $\boldsymbol{u} \in \mathbb{R}^m$, disturbance vector $\boldsymbol{d} \in \mathbb{R}^n$, and dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d})$, where the initial states $\boldsymbol{x}_0 = \boldsymbol{x}(0)$, disturbances $\boldsymbol{d}_0 = \boldsymbol{d}(0)$, and inputs $\boldsymbol{u}_0 = \boldsymbol{u}(0)$ are bounded by sets, $\boldsymbol{x}_0 \in \chi_0$, $\boldsymbol{d}_0 \in D_0$, $\boldsymbol{u}_0 \in U$. The reachable set of the system for a time interval $t \in [0, T_{reach}]$ is:*

$$R_{[0,T_{reach}]} = \left\{ \psi(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{d}_0, t) \;\middle|\; \boldsymbol{x}_0 \in \chi_0,\ \boldsymbol{u}_0 \in U\ \boldsymbol{d}_0 \in D,\ t \in [0, T_{reach}] \right\},$$

where $\psi(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{d}_0, t)$ is the solution of the ODE at time t with initial state $\boldsymbol{x}_0$ under control input $\boldsymbol{u}_0$ and disturbance $\boldsymbol{d}_0$.[3]

In practice, for systems with non-trivial continuous dynamics, obtaining the exact reachable set is often extremely difficult or undecidable. In fact, even for linear systems, obtaining the exact reachable set is only possible if the matrices that describe the differential equations possess a specific eigen-structure [39]. Such a structure is outlined in [53]. Thus, for general nonlinear systems, deriving the reachable set involves obtaining a sound over-approximation of $R_{[0,T_{reach}]}$, such that the actual system behavior is contained within the over-approximation [52, 58, 221]. There are a variety of set representations for accomplishing this task, however, the algorithm utilized in this work uses n-dimensional hyper-rectangles ("boxes") to generate reachtubes [106]. Over long reachtimes, the over-approximation error resulting from the use of this representation can be problematic. However, for short reachtimes, it is ideal in terms of its simplicity and speed [108].

The over-approximation error and the number of steps used in generating the reachable set can be controlled by a reachtime (T_{reach}) step size h. This parameter defines the level of discretization of the time interval $[0, T_{reach}]$ and can be used to tune the runtime of the reachability computation. Bak et al. leverage the step size to make the reachability algorithm amenable to the anytime computation model in the real-time

[3] Our assumption is that f is globally Lipschitz continuous. This property guarantees the existence and uniqueness of a solution for every initial condition in χ_0.

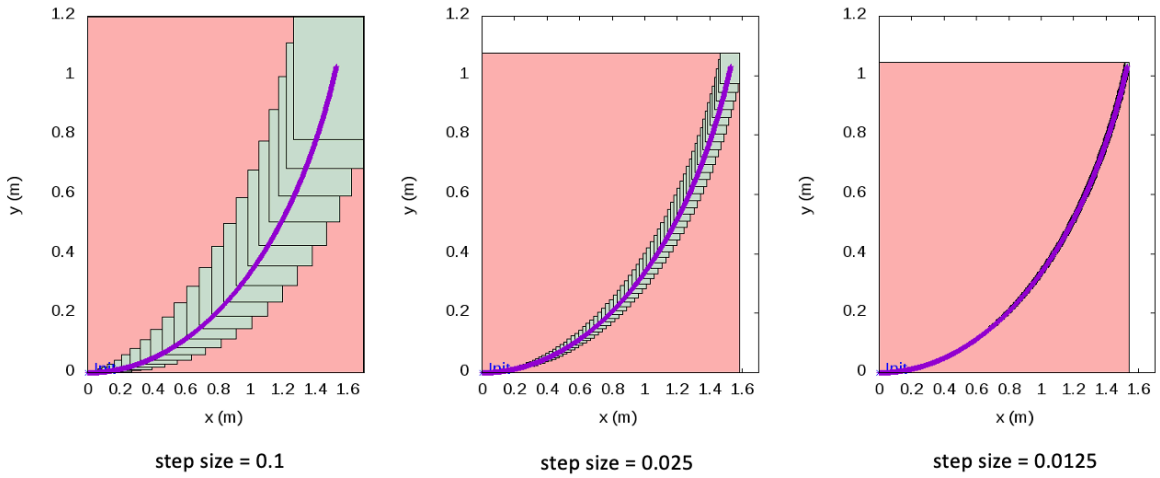

Figure V.3: The real-time reachability algorithm always returns an over-approximation of the reachable set of states. The over-approximation error decreases with successive iterations, provided that there is enough runtime for re-computations. The above images demonstrate this aspect by simulating a left-hand turn control action for $T_{reach} = 2$ seconds. The green boxes represent the set of reachable states, the red rectangle represents the interval hull of the reachable states, and the purple points are points obtained from a simulation of the vehicle's dynamics.

scheduling literature [167]. Thus, given a fixed runtime, $T_{runtime}$, we compute the reachable set $R_{[0,T_{reach}]}$, and if there is remaining runtime, we restart the reachability computation with a smaller step size. In both this work and [108], the step-size is halved in each successive iteration, leading to more accurate determinations of the reachable set. The relationship between the over-approximation error and the step size can be seen in V.3. We refer readers to the following papers for an in depth treatment of these procedures [221, 108, 106].

V.4.3 Safety Checking

The computation of the reachable set allows us to reason whether the system under consideration will enter an unsafe situation in the future. Furthermore, by supposing a dynamics model for the dynamic obstacles within the environment, one can reason about potential future collisions. While the work described in [108, 221, 106] made use of Lyapunov Stability theory in order to reason about the safety of systems, the following manuscript extended the approach to handle online collision avoidance queries. Thus, the safety checking in this work is formulated as checking whether an intersection between a set of unsafe states and $R_{[0,T_{reach}]}$ is empty.

We define the notion of safety considered in this work below.

Definition 8 (SAFETY) *Let Λ represent the set of unsafe states. A system is considered safe over the finite time horizon, T_{reach}, if $R_{[0,T_{reach}]} \cap \Lambda = \emptyset$.*

The unsafe set, Λ, consists of all static obstacles within the environment, described by a bounding-box, the

boundaries of the racetrack, characterized by a list of finely separated points, and the union of the reachable sets of the dynamic obstacles. These representations are then converted into their hyper-rectangle formulations that make up Λ.

Definition 9 (UNSAFE SET) *Given a set of N dynamic obstacles and a set O of static obstacles within the environment, let $R_{i[0,T_{reach}]}$ denote the reachable set of the i-th dynamic obstacle. The set of unsafe states Λ is:*

$$\Lambda = O \cup \bigcup_{i=1}^{N} R_{i[0,T_{reach}]}.$$

V.2 provides a visualization of the obstacles we considered in our simulation experiments, and V.7 displays one of the hardware experiments we conducted evaluating our approach. If there are no intersections between $R_{[0,T_{reach}]}$ and Λ, then we conclude that the system is safe. However, since our approach computes an over-approximation of $R_{[0,T_{reach}]}$, it may lead to conservative observations of unsafe behavior. This occurs when the error in the over-approximation of $R_{[0,T_{reach}]}$ results in intersections with the set of unsafe states, despite these intersections not occurring with the exact reachable set. By refining the reachable set in successive iterations, our regime seeks to mitigate the occurrence of falsely returning *unsafe*.

There are two chief considerations in the anytime implementation of the safety checking procedure, which are (1) the overall soundness of our approach and (2) the real-time nature of our scheme. Satisfying both requirements constitutes the *novel* extensions of the aforementioned algorithm. For the results of the verification to be sound, the safety checking process must be carried out in its entirety before a safety result is issued. At the same time, this requirement must be balanced alongside the real-time stipulation that tasks operate within pre-defined and deterministic time spans. Thus, our implementation ensures soundness properties while maintaining a low-likelihood of missing timing deadlines.

Ideally, to ensure there were no missed deadlines, we would build our system in a *Real-Time Operating System* (RTOS), which allows for the specification of task priorities, executing them within established time frames. However, our implementation does not make use of an RTOS and instead depends on native Linux and the Robot Operating System (ROS) to handle task management. To combat this shortcoming and reduce the number of missed deadlines, we estimate the time required to compute the next reachability loop. If our estimate exceeds the remaining allotted time, the process terminates. There is an inherent tradeoff between the conservativeness of our runtime estimates and the conservativeness of the resulting reachable set. In this work, we chose to maximize the number of iterations used in constructing the reachable set at the risk of occasionally missing deadlines. Our experiments demonstrate that we were successful in minimizing the number of missed deadlines during operation.

Let $k = T_{reach}/h$ denote the number of hyper-rectangles used in representing the reachable set.[4] Since each successive iteration decreases the step size by half, the number of hyper-rectangles that make up $R_{[0,T_{reach}]}$ doubles. Thus, the complexity of the safety checking process is $\mathcal{O}(2^k)$.[5] Therefore, we can estimate that a subsequent iteration of the algorithm will take twice as long as the current one and bloat this estimate to be conservative.

A high-level overview of the reach-set construction and safety checking procedures defining our safety assurance framework is presented in Algorithm 1. The *constructReachSet* function is defined by Definition 7, and realized through hyper-rectangle-based mixed-face-lifting methods. Notably, we extend the original reach-set construction process outlined in [108] to handle uncertain model parameters and set-based disturbances. The safety checking process is implemented as outlined in Section V.4.3. Computing the elapsed time, $computeElapsedTime()$, is done by leveraging functions of the underlying operating system. Finally, the function $estimateNextIterationRuntime(elapsedTime)$ is based on our requirement that the reach-set construction and safety checking process be carried out in their entirety, as outlined above.

Algorithm 1: Safety Assurance Leveraging Real-Time Reachability

```
INITIALIZE
Input: χ0, U, D, T_reach, T_runtime
Output: safe (boolean)

elapsedTime = 0
T_remaining = T_runtime
while T_remaining > 0 do
    safe = true
    R_[0,T_reach] = constructReachSet(χ0, U, D, T_reach, h)
    if R_[0,T_reach] ∩ Λ ≠ ∅ then
        safe = false
    elapsedTime = computeElapsedTime()
    nextIterationEstimate = estimateNextIterationRuntime(elapsedTime)
    T_remaining = T_runtime − elapsedTime − nextIterationEstimate
    h = h/2
return: safe
```

V.5 Experimental Overview

In this section, we detail the steps needed to implement our runtime assurance framework for the safety assurance of a 1/10 scale autonomous vehicle known as the F1/10. First, we construct a mathematical model of the F1/10 car's physical dynamics using system identification techniques. Next, we synthesize a series of controllers frequently used within autonomous racing whose control decisions will be analyzed at runtime.

[4] We begin the flow-pipe construction with an initial time step of $h = T_{reach}/10$.

[5] This analysis neglects consideration of the obstacles and the points used to represent the racetrack boundaries. Since these do not change between iterations, reasoning only about the hyper-rectangles is sufficient.

These controllers include a standard path tracking controller, a gap-following based collision avoidance controller, and a machine learning (ML controller) synthesized using imitation learning (IL). All the controllers use sensor information to determine the desired steering angle for the vehicle. At runtime, the mathematical model obtained through system identification is used within the reachability algorithm to reason about safety of the control actions selected by each of the controllers. Finally, the mathematical model of the F1/10 dynamics is augmented to include an interval based model of uncertainty and environmental disturbances. This allows us to reason about the effects that uncertainty imposes on the overall safety of the system. Finally, the simplex architecture provides the framework for ensuring safe operation of the F1/10 in the event of potential safety violations.

V.5.1 The F1/10 Autonomous Platform

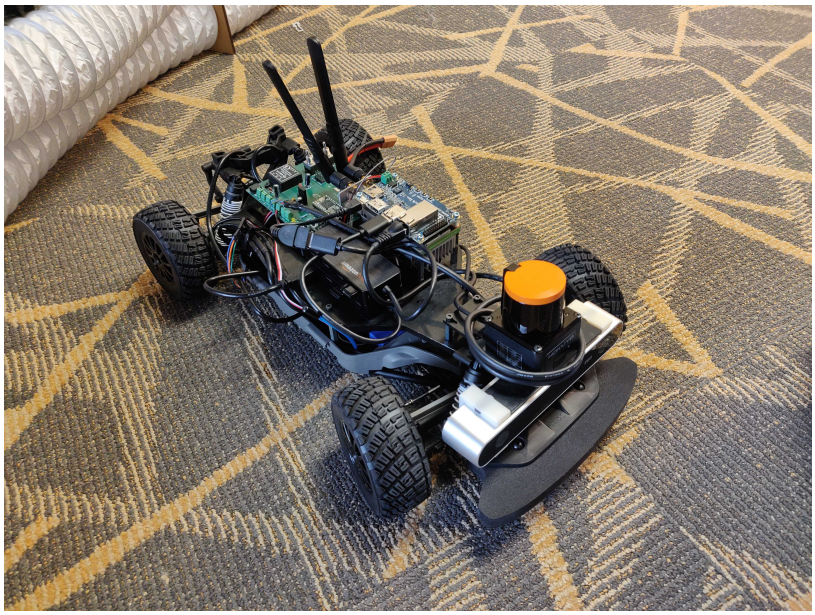

Figure V.4: Visualization of our experimental F1/10 hardware platform. This platform is a one-tenth scale RC car that has been altered to entertain autonomous control inputs as well as support a sensor and compute architecture for autonomous decision-making. [2].

The F1/10 platform of O'Kelly et al. [174] was originally designed to emulate the hardware and software capabilities of full-scale autonomous vehicles. The platform is equipped with a standard suite of sensors such as stereo cameras, LiDAR (Light Detection And Ranging), and inertial measurement units (IMU). The platform uses an NVIDIA Jetson TX2 as its compute platform, and its software stack is built on the *Robot Operating System* (ROS)[6] [189]. The result is a platform that allows researchers to conduct real-world experiments that investigate planning, networking, and intelligent control on a relatively low-cost, open-source

[6]It is worth noting that ROS is not an operating system in the traditional sense but rather a meta-operating system that primarily provides the message passing interface for various components within robot software development [223].

test-bed [174]. A picture of the platform is shown in V.4. Additionally, to promote rapid prototyping and consider research questions around closing the simulation to reality gap[224], Varundev Suresh et al. designed a Gazebo-based simulation environment [225] that includes a realistic model of the F1/10 platform and its sensor stack [226]. We utilize this simulation environment for a number of experiments and training our controllers.

V.5.2 System Identification and Model Validation

The physical dynamics of the F1/10 vehicle can be modeled using a kinematic bicycle model [227], which is described by a set of four-dimensional nonlinear ordinary differential equations (ODEs). The kinematic bicycle model is characterized by relatively few parameters and tracks reasonably well at low speeds.[7] The model has four states: Euclidean positions x and y, linear velocity v, and heading θ. The dynamics are given by the following ODEs:

$$\begin{aligned} \dot{x} &= v\cos(\theta+\beta) & \dot{\theta} &= \frac{v\cos(\beta)}{l_f+l_r}\tan(\delta) \\ \dot{y} &= v\sin(\theta+\beta) & \beta &= \tan^{-1}\Big(\frac{l_r\tan(\delta)}{l_f+l_r}\Big), \\ \dot{v} &= -c_a v + c_a c_m(u_v - c_h) \end{aligned} \tag{V.3}$$

where v is the car's linear velocity, θ is the car's orientation, β is the car's slip angle, x and y are the car's position, u_v is the throttle input, δ is the steering input, c_a is an acceleration constant, c_m is a motor constant, c_h is a hysteresis constant, and l_f and l_r are the distances from the car's center of mass to the front and rear respectively [228]. For simplicity, since the slip angle is fairly small at low speeds, we assume that $\beta = 0$.

While the kinematic bicycle model is an effective model for describing vehicle dynamics, in order to describe the dynamics of the F1/10 precisely, it must be parametrized with respect to measured data obtained from experiments that characterize the behavior of the F1/10 system in various contexts [261]. The process of parametrizing an underlying theoretical model of a system with experimental data is often referred to as *Grey-Box System Identification*, and in this work we utilized MATLAB's Grey-Box System Identification toolbox to obtain the acceleration constant, c_a, motor constant, c_m, and hysteresis constant, c_h, defining its dynamics. We obtained the following parameters for the simulation model of the F1/10: $c_a = 1.9569$, $c_m = 0.0342$, $c_h = -37.1967$, $l_f = 0.225$, $l_r = 0.225$. The model was validated using six experimental campaigns with an average *Mean Squared Error* (MSE) of 0.003. A sample experimental campaign is shown in V.5. For the hardware platform, we obtained the following parameters: $c_a = 2.9820$, $c_m = 0.0037$, $c_h = -222.1874$, $l_f = 0.225$, $l_r = 0.225$, with a validation MSE of 6.75×10^{-4}.

[7] The kinematic bicycle model typically tracks well under $5m/s$ [228]

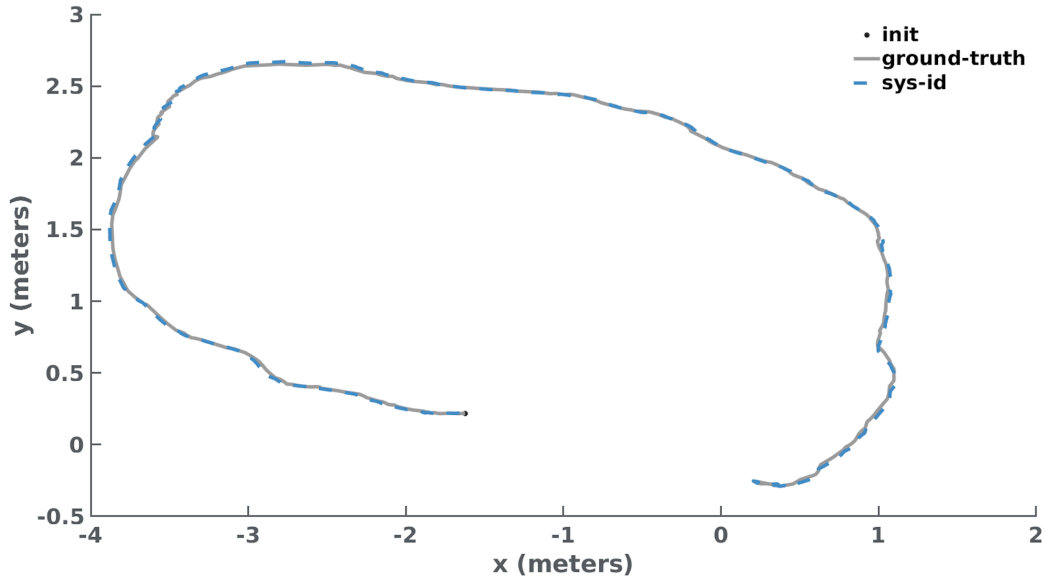

Figure V.5: Vehicle Position (map frame). Illustrative example of an experiment used in validating the F1/10 Hardware Model. The model was validated using data collected from six experimental runs. *Init* corresponds to the starting position of the vehicle in the considered experiment.

The system identification results demonstrate that our model is reasonably accurate. However, as discussed in Section V.3.4, developing an exact model is extremely difficult due to the presence of complex physical interactions that may be hard to describe precisely [253]. Thus, to allow for the modeling of uncertainty, we can extend the dynamics presented in Equation (V.3) to allow each of the state variables and parameters that define the dynamics to be described by sets. Additionally, we allow for the modeling of bounded set based disturbances with respect to the velocity and orientation variables of our model in order to capture phenomena such as drag forces and friction that are not explicitly captured by the kinematic bicycle model. Therefore, the dynamics of our system become:

$$\begin{aligned} \dot{X} &= V\cos(\Theta+\beta) & \dot{\Theta} &= \frac{V\cos(\beta)}{l_f+l_r}\tan(\delta)+D_2 \\ \dot{Y} &= V\sin(\Theta+\beta) & \beta &= \tan^{-1}\left(\frac{l_r\tan(\delta)}{l_f+l_r}\right) \\ \dot{V} &= -C_aV+C_aC_m(u_v-C_h)+D_1 \end{aligned} \tag{V.4}$$

where the operations displayed above are set based operations, X,Y,V, and Θ, are the equivalent set-based state variables, D_1 and D_2 denote the disturbance with respect to velocity and orientation, and C_a, C_m, and C_h are sets that describe the uncertainty with respect to the acceleration constant, motor constant, and hysteresis constant defining our model.[8]

[8] As an example, one could use an interval to describe the uncertainty associated with one of the parameters defining the kinematic bicycle model. That is $C_a = \{c_a \in \mathbb{R} : \underline{c_a} \leq x \leq \overline{c_a}\}$, where $\underline{c_a}$ and $\overline{c_a}$ are the upper and lower bounds defining the acceleration constant c_a

Accounting for increasing levels of uncertainty can lead to conservative behavior in the overall system that may potentially degrade system performance with respect to other objectives [253]. One of the challenges that we investigate in this work is the tradeoff between the conservativeness of uncertainty estimates and the performance of our system. Specifically, what we measure is the percentage of actions issued by a controller that are labeled unsafe, as well as the growth in size of the set of reachable states of the system. The details are presented in Section V.3.4.

V.5.3 Dynamic Obstacle Model

Additionally, to reason about the safety of the system in the presence of dynamic obstacles, a model of their behavior is needed. Specifically, we need to be able to compute how fast these agents are moving and the positions within the environment that they are likely to assume. This problem is frequently referred to as *the obstacle tracking problem* within robotics and is a well studied and challenging topic within the autonomous vehicle, computer vision, and robotics literature [262]. In our experiments, we assume that the only dynamic obstacles present within the environment are the other vehicles participating in the race.

Typically, some assumptions are required to constrain the obstacle tracking problem to best suit the context of the application. As an example, if one wishes to model the behavior of an opponent vehicle within an autonomous racing scenario, then it is quite reasonable to model a dynamic agent using a kinematic bicycle model. However, the challenge in this context is predicting the behavior of the opponent vehicle, or in this case the set of control commands that an opponent agent is likely to use. Any predictions with respect to the behavior of the opponent will necessarily be characterized by a great deal of uncertainty. Therefore, one must carefully consider the fidelity of the models used to describe the behavior of dynamic agents. Models that are imprecise may lead to spurious declarations of unsafe behaviors, while models that are too rigid may be unable to capture the range of dynamic behaviors that agents can assume [254].

In our framework, to avoid having to make distributional assumptions about the driving behavior of opponent vehicles, we assume that the obstacles can be described by a two-dimensional kinematic model and a corresponding bounding box. The assumptions made by this model are limited. Intuitively, our assumption is that dynamic agents will continue to follow their current trajectory over short time horizons. Thus, the equations describing the ODE are given as follows:

$$\begin{aligned} \dot{x} &= v_x, \\ \dot{y} &= v_y, \end{aligned} \tag{V.5}$$

where v_x and v_y are the velocities in the x and y direction, respectively. Additionally, we make the assumption

that we have access to the position and velocity of the other race participants. Similar to the kinematic bicycle model, the dynamics given by Equation (V.5) can be formulated as a differential inclusion, in order to capture the uncertainty associated with measuring the position and velocity of dynamic obstacles using the vehicle's onboard sensors.

V.5.4 Controller Implementation

One of the motivations in utilizing real-time reachability is that it abstracts away the need to analyze the underlying controller and instead focuses on the effects of control decisions on the system's future states. Thus, the nature of the underlying controller can vary quite significantly. This is particularly useful when the controller is a complex machine learning component such as a neural network that may be characterized by billions of parameters. To demonstrate the potential use cases for our methods as well as provide a broad picture of the application of our safety regime, we considered three different controllers within our experiments. These controllers include a standard path tracking controller, a gap-following based collision avoidance controller, and a machine learning controller synthesized via imitation learning (IL).

In this section, we provide a high-level introduction to the construction of the controllers used within our experiments.

V.5.4.1 Pure Pursuit Controller

The first controller considered within our experiments makes use of the Pure Pursuit algorithm. The Pure Pursuit algorithm is a widely used path-tracking algorithm that was originally designed to calculate the arc needed to get a robot back onto a path [263]. It has shown great success being used in numerous contexts, and in this work we utilize it to design a controller that allows the F1/10 vehicle to follow a pre-defined path along the center of the racetrack.

V.5.4.2 Gap Following Controller

Obstacle avoidance is an essential component of a successful autonomous racing strategy. Gap following approaches have shown great promise in dealing with dynamic and static obstacles. They are based on the construction of a gap array around the vehicle used for calculating the best heading angle needed to move the vehicle into the center of the maximum gap [2]. In this work, we utilize a gap following controller called the "disparity extender" by Otterness et al. that won the F1/10 competition in April 2019 [230].

V.5.4.3 Vision Based Imitation Learning

As modern data-driven and machine learning methods have become increasingly scalable and efficient, these methods have begun to be routinely used within autonomous applications. Particularly within perception

tasks, machine learning models are frequently used to gain a semantic understanding of the objects within a vehicle's environment [264]. Unfortunately, these models are notoriously difficult to analyze [114, 9].

One area of machine learning that has enjoyed wide success is Imitation Learning (IL). IL seeks to reproduce the behavior of a human or domain expert on a given task [185]. These methods fall under the branch of *Expert Systems* in AI, which has seen a surge in interest in recent years. The increased demand for these approaches is spurred on by two main motivations. (1) In many settings, the number of possible actions needed to execute a complex task is too large to cover using explicit programming. (2) Demonstrations show that having prior knowledge provided by an expert is more efficient than learning from scratch [185]. While these approaches have demonstrated great efficacy in fixed contexts, there are concerns regarding their ability to generalize to novel contexts where the operating conditions are different from those seen during training, providing a need for effective runtime verification like the one explained in this work [185].

Since the seminal work of Krizhevsky et al. [229] in the ImageNet Large Scale Recognition Challenge, *Convolutional Neural Networks* (CNNs) have revolutionized the field of computer vision. Within the context of autonomous vehicles, CNNs have demonstrated efficacy for driving tasks such as lane following, path planning, and control, simultaneously, by computing steering commands directly from images [187]. In this work, we utilized the CNN architecture, DAVE-2, initially proposed by Bojarski et al. to drive a 2016 Lincoln MKZ, to control the F1/10 model. The data we used to train DAVE-2 was collected from a set of simulation experiments where the sensor-action pairs were generated by a path tracking controller optimized to keep the F1/10 in the center of the track in the absence of obstacles. Such an environment is shown in V.2.

Our main motivation in featuring a machine learning controller based on IL is to highlight the monitoring of a component that may be too complex to analyze. Beyond the seminal work of Bojarski et al. [187], machine learning components are not typically used for control tasks in autonomous systems beyond dealing with systems with unknown dynamics. However, there is a significant amount of promising research in this realm [265].

V.5.5 ROS Simplex Architechture

Our simplex architecture for the F1/10 is designed using ROS [189], and an overview of the design is shown in V.6.

There are two considerations that play a major role in designing the simplex architecture: (1) the finite time horizon, T_{reach}, over which we are reasoning about safety and (2) the amount of time, $T_{runtime}$, allocated for the computation of the reachsets. In our experiments, we use $T_{reach} = 1.0s$ and $T_{runtime} = 25ms$, unless otherwise specified. These values were determined considering the empirical results of how long it took the F1/10 to come to a stop at speeds less than $1.5m/s$ and the control period, $20Hz$, which the reachability

computation needs to finish within in order to not miss a deadline.[9]

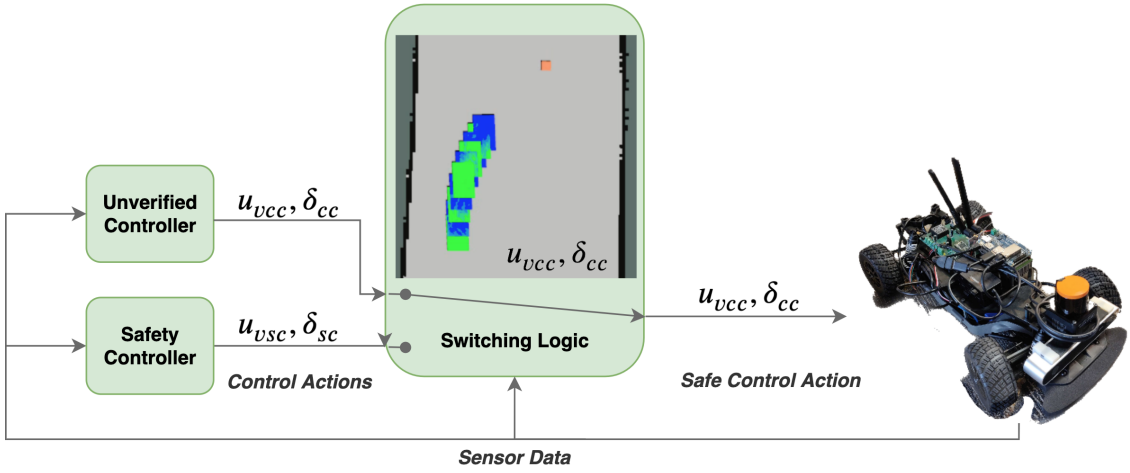

Figure V.6: Overview of the simplex architecture deployed on the F1/10 system described in Section V.5.5. The switching logic consists of monitoring the intersection between the reachable set of the F1/10 and the positions of static and dynamic obstacles within the environment. In the above figure, u_{vcc}, δ_{cc}, corresponds to the control action issued by the complex, or high performance controller, while u_{vsc}, δ_{sc} corresponds to the control action issued by the safety controller. The reachability regime, uses u_{vcc}, δ_{cc} to determine the set of states that the vehicle will assume over T_{reach}. In the above figure, the alternating blue and green rectangles correspond to the intermediate reachable states defining the vehicle's trajectory. Since there are no intersections with obstacles in the environment, the control action issued by the complex controller can safely be used by the F1/10.

Within this architecture, the primary sensors we rely on are a LiDAR and Stereo Labs' Zed Depth Camera. The messages from the LiDAR are published at $40Hz$, and the camera messages are published at $20Hz$. Additionally, we rely on odometry information, published at $40Hz$, to ascertain the state of the F1/10 vehicle. In our design, we decouple the control of the car's steering and throttle control. The controllers evaluated in our work are primarily concerned with the steering control, δ, and the throttle control is designed to maintain a constant speed, u_v. The primary reason we elected to use a constant speed in our experiments was to be able to evaluate the performance of each controller with respect to a single metric, thereby making comparisons across controllers easier.

In the traditional simplex architecture, both the decision module and the safety controller must be verified for the system to be verifiably safe [108]. While this is straightforward for relatively simple controllers, it is significantly more challenging for many classes of controllers, especially when real-time execution is considered [228]. However, the main focus of this work is evaluating the use of the reachability algorithm as a switching logic for the simplex architecture. Thus, we opted not to develop a "formally verified" safety controller. Instead, we selected a controller based on a gap-following algorithm optimized to avoid collisions

[9]We limit velocities to $1.5m/s$ because a lap on our physical track is approximately $13.08m$. Races held by the F1/10 community are around $30-50m$ per lap with larger distances between the track walls, allowing for much faster operating speeds. The rules are described in more detail here: **https://f1tenth.org/misc-docs/rules.pdf**

with obstacles. A detailed description of the gap-following algorithm can be found in the following report [230]. It was primarily selected due to its robust collision avoidance ability and simplicity.

V.6 Experimental Evaluation

Having described the details of our reachability algorithm, dynamics modeling, controller construction, notion of safety, and simplex architecture, we now present the experimental evaluation of our proposed approach. This section is concerned with four major experimental themes. Our first set of experiments presents a safety analysis of the controllers presented in Section V.5.4 under a diverse set of experimental scenarios. The second study describes the implementation of the simplex architecture, described in Section V.5.5, aimed at eliminating collisions and ensuring safety. Next, we present a runtime characterization of our reachability regime, evaluated using two separate platforms. Finally, we conclude this section with a presentation of a set of experiments analyzing the impact of various classes of uncertainty on the overall performance of our safety regime.

The experiments presented here were conducted both in simulation on the physical F1/10 hardware platform. The aim of the simulation experiments was to promote reproducibility for those without hardware access and allow for consideration of a vast set of experiments. The hardware trials validate our claims that our safety regime admits minimal resource requirements.[10]

V.6.1 Controller Safety Analysis

The first set of experiments that we considered were concerned with how the controllers, discussed in Section V.5.4, performed in a variety of contexts. Specifically, what we were interested in was the portion of actions labeled as safe during a particular experiment. Consequently, to solely examine this metric, the following experiments did not make use of the simplex architecture that we described previously. To evaluate the performance of the various controllers, we considered a sizeable diversity of experiments with respect to the speed set-point u_v, the presence and configuration of obstacles, and the number of opponents present within the racetrack.[11] The opposing vehicle's speed was set at $0.5m/s$ and utilized the disparity extender for navigation.[12] Our assumption is that the other vehicles in the environment are collision averse[13]. Each experimental configuration was studied over five experiments consisting of a minute in length. A summary of these experiments is presented in Tables V.1 and V.2.

[10]The simulation artifacts can be found at: https://zenodo.org/record/6418817.

[11]We limited the number of vehicles present within the racing environment to two or three vehicles, primarily because the simulator's performance decreases significantly with each additional car. The real-time factor for our Gazebo simulations, which is a parameter that communicates how fast time in the simulation environment is running relative to real-time, was on average about 0.61 on a desktop with 32 GB RAM and Intel Core i7-7700. Our future work aims to improve on this by utilizing a simulator with lower resource requirements.

[12]The reason we selected this speed was to guarantee that an overtaking action would be considered at least once during each experiment.

[13]This assumption however does not prohibit aggressive driving maneuvers

In general, the overall safety of each controller decreased as the number of static and dynamic obstacles present within the racing environment increased. The same effect was observed with an increase in the speed set-point utilized by each controller. Beyond these general trends, the overall performance of each controller varied significantly. For instance, since the pure pursuit controller was tasked with following a predefined path along the center line of the racing environment, it was in general the safest controller with respect to the number of actions labeled safe during an experiment. Provided that an opposing vehicle did not cross its path, or there were no obstacles located on the path that the pure pursuit controller was tasked with tracking, there was a low risk for collisions. In contrast, the pure pursuit controller performed poorly in experiments with obstacles, and it experienced the highest number of collisions among the controllers we studied.

The vision based controller, which was trained to mimic the behavior of the pure pursuit controller, displayed similar results in the experiments that we considered. However, its performance was much less robust. In some scenarios, the vision based controller out-performed the pure pursuit controller. As an example, the vision based controller displayed an ability to deal with dynamic and static obstacles at low speeds. At the same time, its use also resulted in numerous collisions in scenarios without obstacles at speeds of greater than 0.5 m/s. Thus, the vision based controller was the least tolerant to speed changes in the experiments that we considered. While broad generalizations based on these results cannot be made, these results embolden our belief that monitoring machine learning components is of utmost importance.

Finally, the results of the experiments with the disparity extender were the most idiosyncratic. One of the most peculiar results of these experiments is that the use of the disparity extender displayed relatively low levels of safety despite being designed for collision avoidance. However, this observation can be explained by the greedy nature of its design, as described in [230]. Since the underlying goal of the disparity extender is to situate itself within the direction of the maximum gap, in practice, this design causes it to frequently steer the vehicle close to the racetrack boundaries as well as the other race participants. Despite the number of unsafe declarations issued by our reachability regime, the disparity extender was by far the most consistent controller across all the experiments that we conducted, and it displayed the greatest robustness to the presence of static and dynamic obstacles as well as higher speeds. Moreover, it experienced the lowest rate of collisions amongst the controllers that we evaluated.

V.6.2 Mitigating Collisions via Simplex

In the previous section, we were primarily concerned with the number of safe actions produced by a controller during an experiment, and due to the lack of the implementation of any mitigation strategies in the event of a potential safety violation, collisions occurred with varying levels of frequency depending on the nature of the controller. Preventing the occurrence of these collisions warrants the use of our simplex architecture,

Table V.1: Controller Safety Analysis Without Use of the Simplex Architecture: Simulation Platform

			Obstacles		**No Obstacles**	
Controller	Opponents	u (m/s)	(%) Safe Actions	Collision Frequency (%)	(%) Safe Actions	Collision Frequency (%)
Dispartiy Extender	2	0.5	87.53 ± 0.50	0.0	87.64 ± 0.88	0.0
		1.0	71.02 ± 1.21	0.0	69.21 ± 13.32	20.0
		1.5	61.61 ± 11.26	60.0	71.71 ± 1.27	20.0
	3	0.5	90.99 ± 0.43	0.0	92.17 ± 0.62	0.0
		1.0	63.56 ± 1.06	0.0	65.84 ± 1.58	0.0
		1.5	65.11 ± 0.79	0.0	66.89 ± 1.16	20.0
Pure Pursuit	2	0.5	96.56 ± 0.29	100.0	100.00 ± 0.00	0.0
		1.0	72.35 ± 1.42	100.0	90.43 ± 11.93	20.0
		1.5	82.86 ± 4.77	40.0	94.85 ± 0.33	0.0
	3	0.5	96.79 ± 0.16	100.0	100.00 ± 0.00	0.0
		1.0	7.89 ± 2.87	80.0	12.14 ± 12.19	100.0
		1.5	33.86 ± 33.06	100.0	73.27 ± 34.94	40.0
Vision Based Network	2	0.5	95.36 ± 2.83	20.0	95.16 ± 5.24	20.0
		1.0	75.14 ± 9.97	100.0	16.85 ± 6.09	100.0
		1.5	62.47 ± 3.49	100.0	24.53 ± 2.26	100.0
	3	0.5	97.49 ± 0.56	0.0	99.71 ± 0.33	0.0
		1.0	37.13 ± 25.66	80.0	35.99 ± 8.47	100.0
		1.5	15.96 ± 9.86	100.0	11.09 ± 1.99	100.0

Table V.2: Controller Safety Analysis Using Simplex Architecture: Simulation Platform

			Obstacles		**No Obstacles**	
Controller	Opponents	u (m/s)	(%) Safe Actions	Collision Frequency (%)	(%) Safe Actions	Collision Frequency (%)
Dispartiy Extender	2	0.5	17.47 ± 0.11	100.0	17.61 ± 0.24	100.0
		1.0	25.11 ± 0.72	0.0	24.73 ± 0.97	0.0
		1.5	15.76 ± 0.40	0.0	15.87 ± 0.56	0.0
	3	0.5	74.63 ± 3.48	0.0	71.51 ± 5.38	0.0
		1.0	20.69 ± 0.28	0.0	21.12 ± 0.75	0.0
		1.5	16.93 ± 7.95	0.0	13.39 ± 0.27	0.0
Pure Pursuit	2	0.5	93.95 ± 0.11	0.0	100.00 ± 0.00	0.0
		1.0	74.37 ± 1.13	0.0	88.33 ± 1.46	0.0
		1.5	62.52 ± 1.95	0.0	59.94 ± 0.54	0.0
	3	0.5	94.17 ± 0.11	0.0	100.00 ± 0.00	0.0
		1.0	60.50 ± 17.63	0.0	67.24 ± 18.35	0.0
		1.5	21.04 ± 0.34	0.0	50.77 ± 0.88	0.0
Vision Based Network	2	0.5	80.24 ± 4.62	20.0	91.71 ± 4.68	0.0
		1.0	47.74 ± 3.81	40.0	43.36 ± 13.78	40.0
		1.5	27.89 ± 11.16	80.0	29.88 ± 6.18	60.0
	3	0.5	93.02 ± 2.13	0.0	97.78 ± 1.24	0.0
		1.0	33.66 ± 3.74	0.0	40.69 ± 0.26	0.0
		1.5	18.83 ± 3.12	0.0	38.25 ± 12.81	0.0

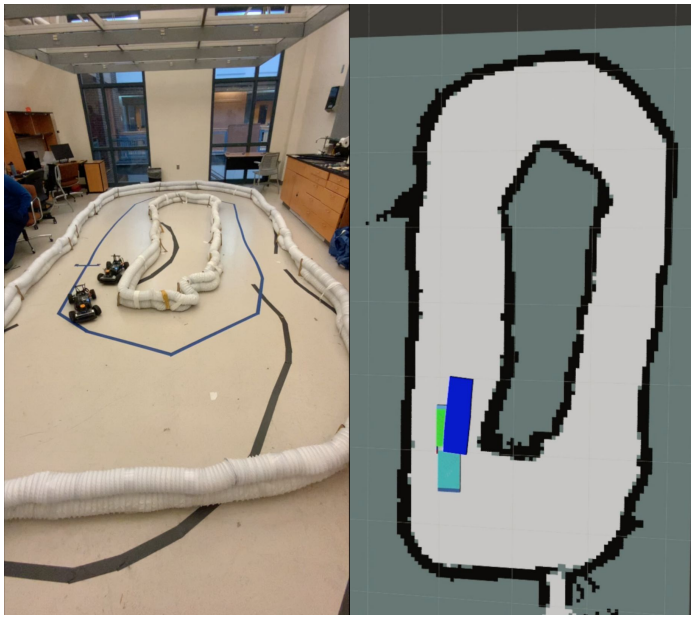

Figure V.7: Example of one of the hardware experiments we conducted evaluating the efficacy of our safety regime in multi-agent racing settings. In the above image, the reachable set of the ego vehicle (dark blue) intersects with the reachable set of an opponent vehicle (light blue). This overlap corresponds to an action that would be labeled unsafe and a switch to the safety controller in our simplex architecture would occur.

and here we present the results of utilizing our architecture under the same set of scenarios considered in Section V.6.1.

Using our simplex architecture, we were able to completely eliminate collisions from occurring when the pure pursuit controller was used to control the F1/10 vehicle. Moreover, the number of collisions that occurred when the other two controllers were utilized also meaningfully decreased, as shown in Table V.1. However, in practice, to provably eliminate all collisions from occurring within our experiments, one must also verify the decision module and the safety controller utilized within the simplex regime. Within our work, the safety controller that we utilized was a conservative version of the disparity extender.[14] Thus, while in practice it is quite effective in eliminating collisions, it is not perfect.

As stated earlier in this chapter, developing a formally verified safety controller is quite challenging in practice. The challenge within this realm is developing controllers whose behavior is not exceedingly limited and whose analysis is feasible. However, the main focus of this work is evaluating the use of the reachability algorithm as a switching logic for the simplex architecture. Thus, rather than developing a provably safe simplex regime, our assumption is that our underlying safety controller is safe. We refer the interested reader to works such as [228, 108] for an in-depth discussion of developing provably safe controllers.

[14]The underlying logic of the safety controller is the same as the disparity extender. However, we limit the maximum speed of this controller to be $0.3m/s$. Additionally, if the distance between an obstacle and the F1/10 falls below $0.5m$, the vehicle stops completely.

Finally, one of the greatest challenges in considering the question of safety within the context of dynamic obstacles is that collision avoidance strategies may not be enough to guarantee safety. For example, consider a scenario where a dynamic agent abruptly swerves into the path of the ego vehicle, thereby causing all the future trajectories of the ego vehicle to result in a collision. These scenarios are known as *Inevitable Collision States* (ICS) [266] and in recent years, there has been significant work towards avoiding ICS. Notably, one popular strategy of dealing with ICS is the use of reachability regimes which inherently satisfy two of the three conditions needed to avoid ICS. Specifically, reachability regimes inherently reason about the underlying dynamics of the ego vehicle, which is the first requirement of avoiding ICS. Secondly, they can be used to reason about the future state of the environment through the computation of the reachable sets of dynamic agents. This is the second condition for eliminating ICS. The final condition, requires reasoning about the previous two items over an infinite time horizon, which is infeasible in practice [266]. However, one can meaningfully decrease the probability of a collision through the selection of a sufficiently long time horizon. Thus, the regime presented in this work can meaningfully be used to mitigate ICS.

Bearing the above in mind, we were able to eliminate all collisions in subsequent experiments by significantly increasing the reachtime horizon used in the reachability regime. However, doing so caused the overall use of the safety controller to increase significantly. While this may be acceptable in certain contexts, it is not always a desirable solution.

V.6.3 Real-time Characterization of Reachability Regime

One of the main benefits of the reachability regime presented in this work is that it admits minimal resource requirements and possesses real-time guarantees [108]. In this section, we present, a runtime characterization of the real-time reachability regime outlined in Section V.4.2. We ran our experiments on platforms running Linux (Ubuntu 16.04 LTS). The runtime analysis of the simulation experiments was conducted on a Dell XPS-15 (9570) with 32GB RAM, a six-core Intel Core i7-8750 @4.1GHz processor, and an Nvidia GeForce GTX 1050Ti 4GB graphics card. The hardware experiments were evaluated on a Jetson TX2 with a Dual-core Nvidia Denver 64-bit CPU (ARM), a quad-core ARM A57 Complex, and an NVIDIA Pascal Architecture GPU with 256 CUDA cores. This latter configuration validates our claims that our safety architecture admits minimal resource requirements.

For benchmarking purposes, we recorded the mean execution-times (Mean ET) of our real-time reachability algorithm, as well as the average number of iterations utilized in constructing the reachable set (Mean Iters). While a discussion of upper bounds on execution times typically involves a discussion of the *Worst-Case Execution Time* (WCET), we instead report the *Maximal Observed Execution Times* (MOET). In general, the WCET is unknown or difficult to derive without the use of static analysis proofs [231]. Since our

safety regime relies on ROS, which is highly dynamic and distributed, it is prohibitively difficult to perform an exhaustive exploration of the space of all execution times and thus derive the WCET. However, we provide a rough proxy of the WCET by reporting the MOET [231] in Tables V.3 and V.4. Additionally, we report the percentage of missed deadlines (PMD) that result from our soundness requirements; as we execute on a regular operating system and not an RTOS, this is possible, and performing the runtime measurements may result in variance due to changing load and scheduling. We demonstrate that this value is low across all experiments.

In the tables that follow, all summary statistics are reported alongside their corresponding standard deviations. Each configuration was evaluated using 30 experiments that were a minute in length.

Table V.3: Analysis of Wall-Time and Speed Variation Simulation Platform

u_v (m/s)	$T_{runtime}$	MOET (ms)	Mean ET (ms)	Mean Iters	PMD(%)
0.5	10	10.71 ± 0.97	7.00 ± 0.13	3.95 ± 0.03	0.13 ± 0.15
	25	28.49 ± 5.23	15.42 ± 0.88	5.07 ± 0.12	2.62 ± 2.71
1.0	10	11.10 ± 1.25	6.85 ± 0.16	4.08 ± 0.23	0.27 ± 0.24
	25	28.36 ± 0.77	14.86 ± 0.14	5.06 ± 0.10	2.83 ± 1.64

Table V.4: Analysis of Wall-Time and Speed Variation: Jetson TX2

u_v (m/s)	$T_{runtime}$	MOET (ms)	Mean ET (ms)	Mean Iters	PMD(%)
0.5	10	10.87 ± 0.36	5.07 ± 0.53	5.39 ± 0.83	0.58 ± 0.29
	25	24.42 ± 0.49	10.41 ± 0.63	7.56 ± 0.97	0.00 ± 0.00
1.0	10	14.36 ± 5.74	5.56 ± 0.58	5.12 ± 0.39	1.61 ± 1.18
	25	31.60 ± 17.81	9.36 ± 0.79	8.69 ± 1.16	0.03 ± 0.06

The experiments demonstrated that Mean ET of our regime fell well within our desired $T_{runtime}$. Our estimates of future iterations of the reachability computations were quite conservative in nature. Though a few deviations were observed as displayed by the MOET, these did not significantly impact the performance of our approach. Rather, they are a result of our requirement that the safety checking process complete.

The runtime characterization of our approach done on the F1/10 hardware platform took on the same structure as the simulation experiments. Our experimental setup is shown in V.7, and a video demonstration of the results is available online.[15]. Similar to the simulation trials, the experiments demonstrated that the Mean ET of our regime fell well within our desired $T_{runtime}$. However, compared to the simulation experiments, our hardware experiments demonstrated a large decrease in the execution time of the approach. However, this result can be explained by the size of our racetrack and the frequency of unsafe action declarations that

[15] https://youtu.be/3jPKucx4AF4

occurred during the experiments.[16]

Within the real-time reachability algorithm, the reachable-set computations terminate whenever an unsafe state is detected. The algorithm then restarts with half the step-size in order to refine the over-approximation error and determine if the "unsafe" declaration is spurious. This strategy continues until the step size falls below a pre-specified threshold specified to guarantee numerical stability. On the hardware platform, this threshold was met consistently, which demonstrates the frequency of unsafe declarations. This observation was explored more delicately in an earlier version of this work [16], and for brevity we refer readers to the aforementioned paper for a more in-depth discussion of this phenomenon. However, in general, experiments with higher levels of safety utilize fewer iterations in constructing the reachable set and generally have higher execution times. The numerical stability termination conditions are also discussed in more detail in [106].

V.6.4 Uncertainty Analysis

In Section V.3.4, we discussed how developing exact models of dynamical systems can be challenging due to the presence of complex physical interactions that may be difficult to explicitly model [253]. While these interactions can frequently be lumped together as uncertainty, one of the challenges in system design is allowing for rigorous estimates of the bounds of uncertain parameters without prohibitively impacting the performance of the system [253]. Intuitively, accounting for more uncertainty in the models of a system and its environment inherently leads to more conservative control strategies aimed at ensuring safety. In this section, we present an analysis of the effects of uncertainty on the conservativeness of the reachability computations that form the basis of our runtime assurance framework.

Our experiments are primarily concerned with the effects of two classes of uncertainty: uncertainty with respect to our derived model of the system and uncertainty related to sensing, localization, and the environment. For context, this section summarizes over 6000, minute long experiments, that provide an enlightening analysis of the effects of uncertainty on our reachability regime.

V.6.4.1 Model Uncertainty

The difficulty of deriving models that are an accurate representation of the real world is the defining cause of uncertainty with respect to the structure and parameters of a system's models [267]. In our experiments, the types of model uncertainty that we considered were set-based disturbances with respect to the velocity and orientation variables of our model, as well as uncertainty around the acceleration, motor constant, and hysteresis constants defining our model. Since the underlying reachability approach leverages hyper-rectangles

[16]The racetrack considered in the simulation experiments is much larger than our real-world racetrack. In simulation, the narrowest point occurs when the walls have a separation of 2 meters. In contrast, our real-world track is slightly over 2 meters at its widest point. Thus, the frequency of unsafe actions declared by our approach in small spaces is significantly larger in our hardware experiments.

for the reachable set computations, we can model phenomena such as drag forces and friction components that are not explicitly captured by the kinematic bicycle model using intervals. As an example, a friction parameter could be described by a real-valued interval:

$$\left[d_1\right] = \left[\underline{d}_1, \overline{d_1}\right] = \{d_1 \in \mathbb{R} \colon \underline{d}_1 \leq d_1 \leq \overline{d_1}\} \tag{V.6}$$

which is a connected subset of $\mathbb{R}$, and $\underline{d}_1$ and $\overline{d_1}$ are the lower and upper bounds of the additive and diminutive frictional components affecting the system's velocity such that, $\underline{d}_1 \leq \overline{d_1}$ [268]. The same intuition applies to the disturbances applied to the steering of the F1/10 model. In this context, the equations of motion of the system are characterized by Equation V.4.

Similarly, in contexts where the system dynamics are a combination of a partial theoretical model and parameters obtained from data, which corresponds to a standard grey-box system identification problem, there is typically some uncertainty associated with the parameters that characterize the underlying system [269]. The space of possible model structures as well as parameters that may characterize an underlying system may be extremely rich, and there is a large body of work aimed at characterizing this uncertainty [269, 256]. Thus, to explore this form of uncertainty in our work, we allowed the parameters defining the kinematic bicycle model to lie within an interval. As an example, one can imagine that the motor or acceleration constant defining the evolution of the velocity of our vehicle could vary depending on the environmental conditions that the system is tasked with operating within.

Table V.5: Uncertainty Analysis of Reachability Computations

	Ground Truth		Particle Filter Localization	
Parameter Uncertainty (%)	Controller Usage (%)	Reachset Size (Median Area A)	Controller Usage (%)	Reachset Size (Median Area A)
0.0	79.95	2.90	77.29	10.15
5.0	78.27	4.88	72.92	12.43
10.0	80.12	7.45	73.01	16.23
15.0	76.26	10.56	65.16	20.30
20.0	63.39	14.55	57.62	23.54
25.0	56.42	19.02	52.24	28.28
30.0	45.33	24.05	41.57	34.44
35.0	38.53	30.82	32.54	40.47
40.0	36.69	39.39	17.60	591.92
45.0	37.69	47.71	1.61	2443.18

We evaluated the effects of uncertainty in our regime in two key ways. The first was an analysis of the growth of the size of the derived reachable set with respect to uncertainty. The second analysis was an investigation of the percentage of time in which the complex controller was utilized during an experimental run with increasing levels of uncertainty. To measure the growth in size of the set of reachable states for the F1/10 system with respect to increasing levels of uncertainty, we computed the median total area of the flow-pipe representing the set of trajectories that the vehicle could assume under the current control action.

Since the safety checking task in our work was concerned with a two-dimensional intersection problem, one can compute the measure of the total area of a flow-pipe by computing the area of each intermediate hyper-rectangle describing the euclidean positions x and y of the vehicle over the finite-time horizon, $t \in [0, T_{reach}]$. In this way, one can investigate the relationship between uncertainty and the conservativeness of the reachability computations.

We now formally define our notion of the total area of a flow-pipe representing the vehicle's trajectories. This notion can be extended to the concept of a volume in higher dimensions. Let $R_t \subset R_{[0,T_{reach}]}$ represent the reachable set of the system at time $t \in [0, T_{reach}]$. In our context, since we utilize hyper-rectangles as the set representation used to characterize flow-pipes, R_t can be described by a Cartesian product of intervals $[x] \times [y]$. Here, $[x]$, and $[y]$ are intervals describing the euclidean position of the vehicle. Finally, let $w([x])$ represent the width of the interval $[x]$, where $w([x]) = \overline{x} - \underline{x}$. Here $\underline{x}$, $\overline{x}$ are the left and right bounds defining $[x]$ such that $\underline{x} \leq \overline{x}$ [270]. Then the total area, A, of the flow-pipe, $R_{[0,T_{reach}]}$, can be defined as follows:

$$A = \sum_{t=0}^{T_{reach}} w([x])w([y]). \tag{V.7}$$

There are three things to note in equation (V.7). Firstly, the number of intermediate reachable sets R_t is defined by the step size, h, used in the reachability computations, as described in Section V.4.2. Secondly, since we had to bloat each intermediate hyper-rectangle representing the vehicle's trajectory to capture the true size of the F1/10 system, the area of each intermediate reachable set is necessarily non-zero. Finally, many of the intermediate reachsets overlap, which leads to an inflated estimate of the area that a flow-pipe assumes. However, our focus is on the relative growth of the flow-pipes under uncertainty. Thus, our over-approximation of the area in this context is sufficient.

V.8 displays the growth in size of the reachable sets with respect to increasing levels of parameter uncertainty, while V.9 displays the relationship between parameter uncertainty and the conservativeness of our safety regime. Table V.5 presents the data shown in the aforementioned figures. From the figures, one can see that there is an exponential relationship between the growth in the size of the reachable set and increasing levels of uncertainty. The same is true of the relationship between uncertainty and the use of the safety controller. This relationship is further exacerbated if long time horizons are utilized in the underlying reachability approach, and V.11 presents a visualization of our investigation of how T_{reach} impacts the conservativness of our reachset estimations. One way of mitigating the growth of the size of the derived reachable set computations is by utilizing short time horizons. However, this may not always be feasible in all contexts.

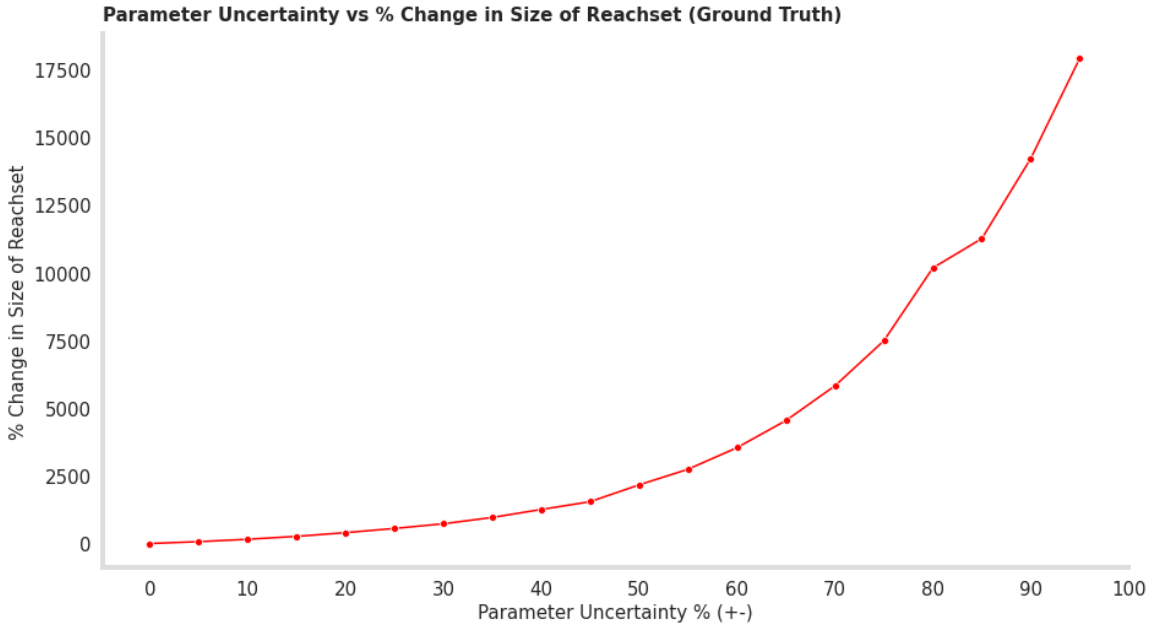

Figure V.8: Relationship between the level of parameter uncertainty in the vehicle dynamics and the size of the reachable set describing the future behavior of the vehicle

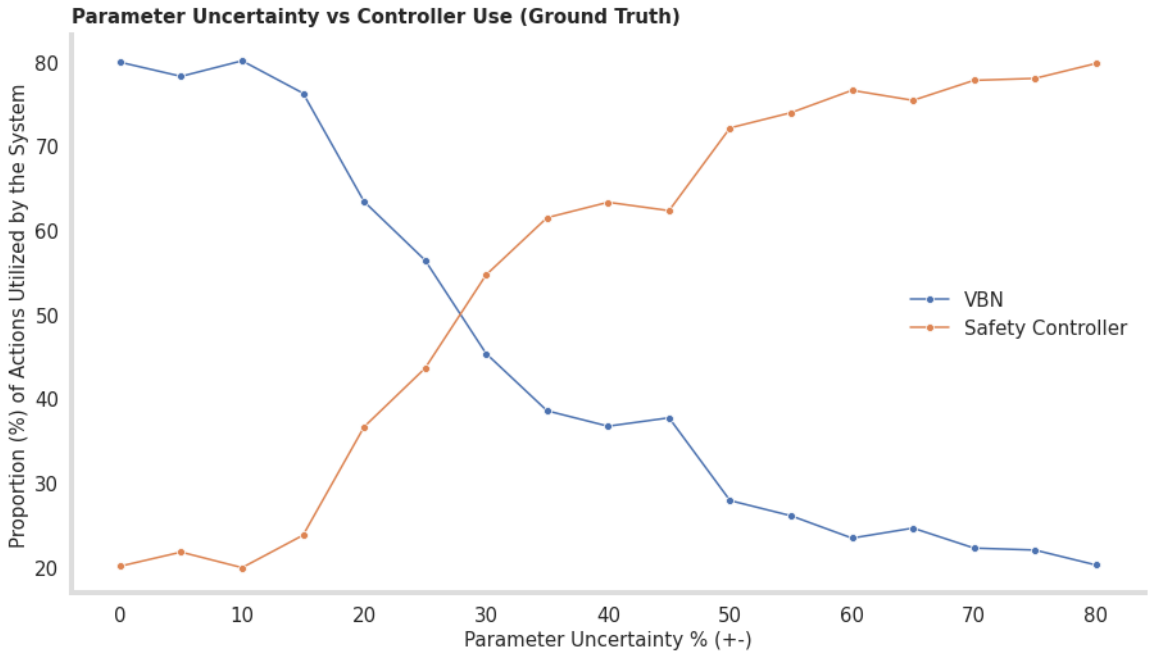

Figure V.9: Relationship between the level of parameter uncertainty in the vehicle dynamics, and percentage of the time in which the vision based machine learning controller was utilized during an experimental run (Controller Usage).

V.6.4.2 Modeling Sensor, Localization and Situational Uncertainty

The data obtained from the sensors onboard autonomous vehicles possess inaccuracies that must be accounted for in the computations aimed at building a higher level understanding of the vehicle's surroundings [271]. One such example are inaccuracies or constraints related to the resolution of a particular sensor's measurements. Typically, significant testing allows for estimations of the variance of the measurements obtained from particular sensors in different contexts [271]. Moreover, these analyses often include descriptions of how various sensors perform in the context of varying weather conditions, temperatures, and other scenarios of interest [271]. These analyses will then inform how sensor observations are used within the control stack of the vehicle.

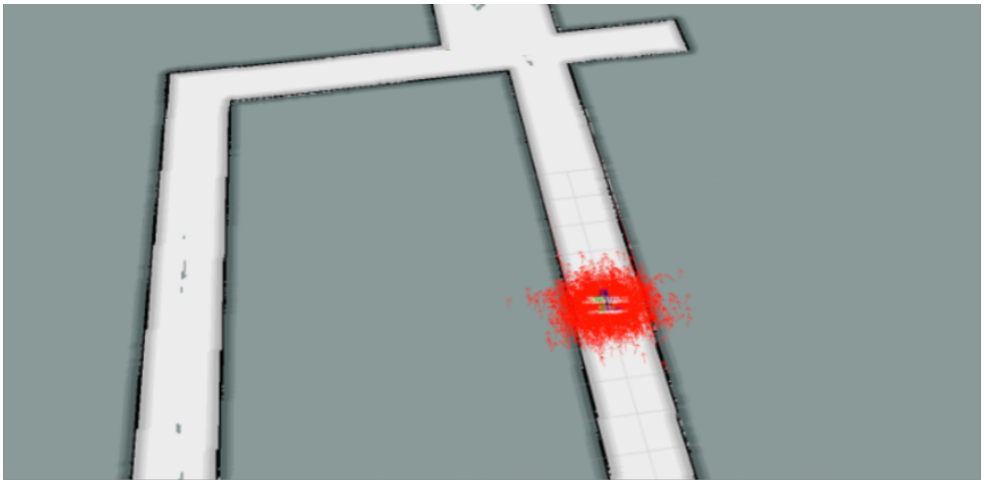

Figure V.10: GPU-based particle filtering for position and orientation estimation, developed by Walsh et al. [3]. Each arrow represents a position and orientation estimate produced by the algorithm.

Bearing the above in mind, arguably the most salient problems within this context are characterizing the propagation of sensing errors, as they relate to the system's basic measure of its position in space and its environment [271]. Localization systems for autonomous vehicles are frequently based on measurements from a variety of sensors, and in state-of-the-art systems, estimations of the vehicle's position with respect to an underlying map must be accurate to within $10cm$ or better [271]. This problem has received significant attention within the research literature, where localization subsystems often make use of algorithms such as particle filters that allow for estimations of the state of the system defined by probability densities that are a function of motion models and sensor information. Thus, they are quite adept at handling and estimating the uncertainty associated with the vehicle's understanding of its environment [3].

Finally, autonomous vehicles must be able to effectively handle interactions with other moving objects and vehicles within its environment [271]. These may include pedestrians, animals, and bicycles, whose behavior the vehicle may have limited knowledge about [271]. While effective methods for detecting, classi-

fying, and tracking objects exist [272], many of these approaches make use of deep learning and probabilistic modeling in order to characterize the behavior of moving objects. Thus, there is an inherent uncertainty in the description of the vehicle's environment.

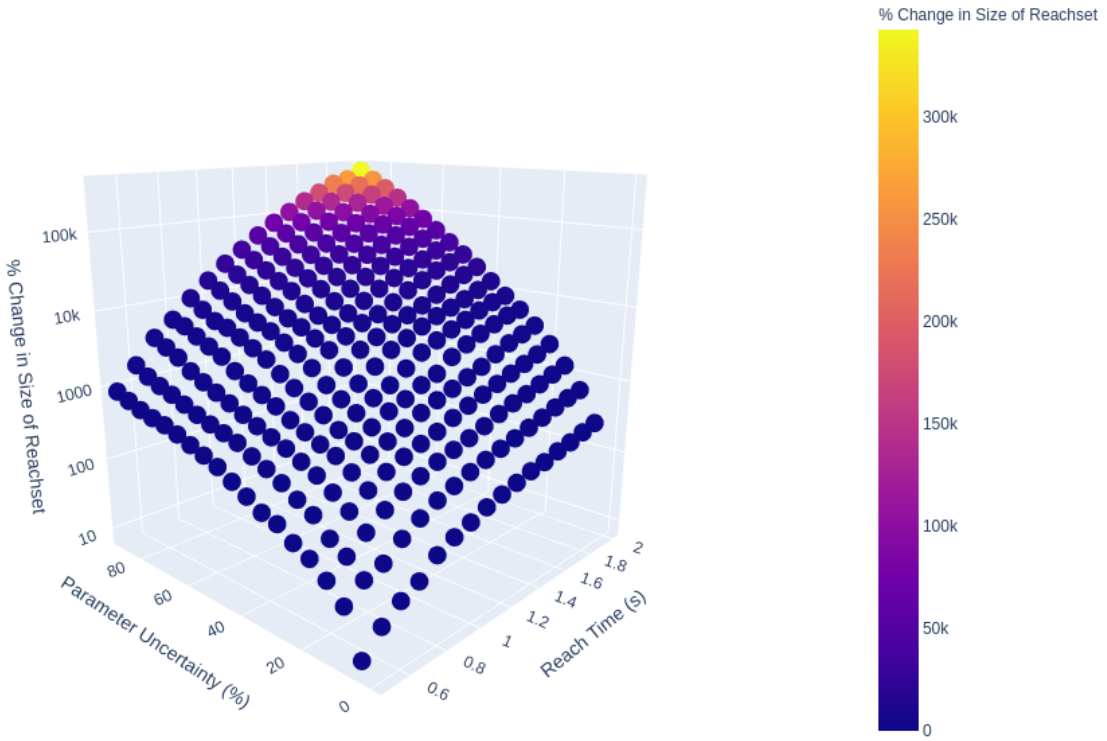

Figure V.11: Relationship between the level of parameter uncertainty in the vehicle dynamics, and the size of the reachable set describing the future behavior of the vehicle. The interested reader can interact with the above figure using the following link: tinyurl.com/8wxx2xnm.

In our experiments, we primarily considered uncertainty as it related to the position and velocity of the dynamic obstacles within our environment, as well as uncertainty with respect to the vehicle's measurement of its own position and orientation within the environment. Specifically, we allowed our estimates of the position and velocity of opponent vehicles to lie within intervals. Intuitively, there is always some uncertainty as it relates to dynamic agents within an environment. Furthermore, our localization subsystem was based on a GPU-based particle filtering localization algorithm developed by Walsh et al. [3]. In this regime, the position of the vehicle can be defined by a set of position and orientation (pose) estimates, known as particles, that are refined using sensor measurements, a motion model, and odometry data. Rather than using an aggregate measure of these particles as our estimation of the vehicle's pose, we allowed our reachability computations to investigate the set of poses defined by each particle estimate. A visualization of the particles can be seen in V.10.

Table V.5 displays our analysis of parameter uncertainty in the physical dynamics of the F1/10 model

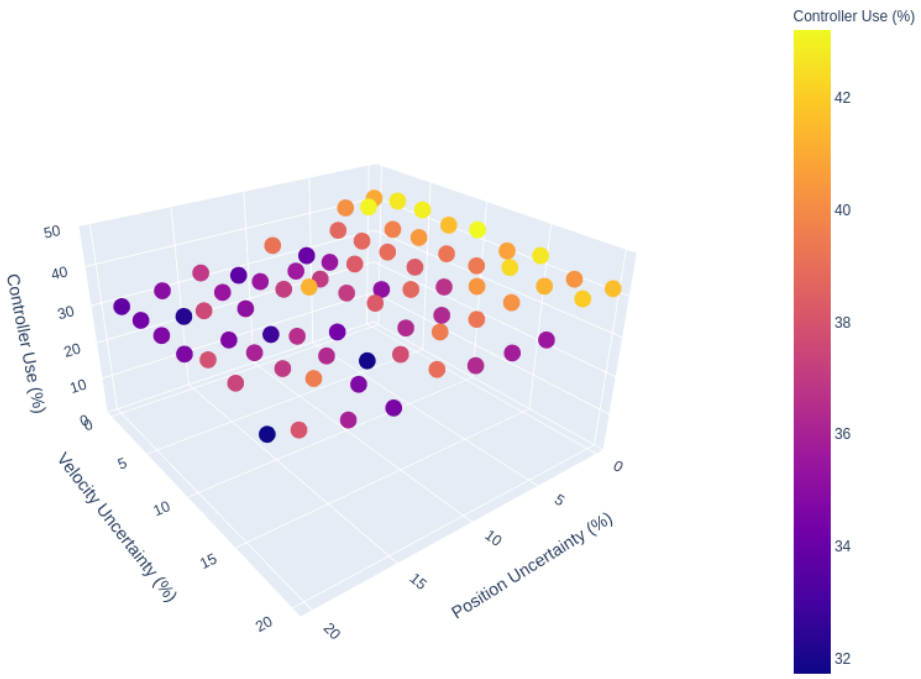

Figure V.12: Visualization of the relationship between increasing levels of uncertainty with respect to estimations of the position and velocity of dynamic obstacles, and the use of the complex controller within our simplex regime. The interested reader can interact with the above figure using the following link: https://tinyurl.com/3dcyab7n.

when the particle filter was used for localization. It also includes an analysis of the effects of parameter uncertainty using ground truth data of the vehicle's position within the simulator. While the relationship between uncertainty and the conservativeness of our reachability regime were largely the same in this context, the exponential growth in the size of the reachable sets for the particle filter experiments was more drastic. In fact, when you compare the percent change in the growth of the reachable set, utilizing the zero parameter uncertainty scenario as a baseline, by the time an uncertainty level of 45% is considered, the growth in the size of the reachsets for the particle filter localization approach are growing at a rate 15.5 times faster than the ground truth experiments.

Finally, our experiments considering uncertainty with respect to the estimation of the velocity and position of dynamic obstacles within the vehicle's environment are shown in V.12. In these contexts, the impact of uncertainty was much less straightforward. While the size of the reachsets describing the position of other agents within the environment grew significantly, this effect had no material effect unless the ego vehicle was within close-proximity of a dynamic agent. Thus, while the length of time that the complex controller was used during an experiment decreased in general, it was not as significant of a drop as the other experiments. In general, to see a similar decrease in the use of the complex controller as the other experiments, one would need to evaluate the controller within contexts where the proximity between the ego vehicle and dynamic

agents is small. However, in our work, due to our design choice of only switching back to the safe controller once a sufficient number of control actions have been determined safe, this is unlikely to occur.

In general, as shown in our experiments, one of the challenges with forward reachability schemes is that while they give strong notions of safety [241], over long time horizons, and significant uncertainty, this can lead to overly conservative behaviors, which may impede performance. An example of such a scenario are the localization uncertainty experiments. Beyond 40% uncertainty in the model parameters, the safety controller was utilized 100% of the time. While backward reachability approaches, such as Hamilton-Jacobi Reachability, are a possible alternative to these schemes, they typically incur a large computational cost. Additionally, we are not aware of any approaches that possess strong real-time guarantees. However, for low dimensional systems they are an attractive framework. We refer readers to the following paper [253] for an in-depth discussion of these methods.

V.7 Discussion and Future Work

Having evaluated the merits of our approach, both in simulation and on an embedded hardware platform, we now present some observations based on our results. In particular, we briefly focus on real-time considerations and the main limitations of our approach.

V.7.1 Real-Time Evaluation and Missed Deadlines

The basic requirement for real-time systems is that tasks operate within pre-defined and deterministic time spans. Often, this is accomplished through the use of a real-time operating system (RTOS), which allows for the specification of task priorities to ensure they are executed within established time frames. Our implementation did not make use of an RTOS, thus task management was left to the native Linux implementation. While our experimental evaluation did demonstrate deviations from the specified wall-time, the mean percentage of missed deadlines on the Jetson TX2 was fewer than 2% across all of our experiments.

V.7.2 Limitations

While the reachability algorithm presented in this work possesses provable guarantees, our architecture does not. Obtaining these guarantees requires developing a formally verified safety controller and switching logic, which was outside the scope of the work presented herein. Therefore, it is possible to enter a state in our framework in which all future trajectories will result in a collision. These states are known as *inevitable collision states* and have been well-studied within the motion planning literature [232]. In future work, we hope to address this limitation by leveraging approaches such as viability kernels and dynamic safety envelopes that allow for the synthesis of provable safe control regimes [233].

One of the challenges that emerged in our experiments was that uncertainty sharply increased the overall conservativeness of the reachable sets derived by our reachability regime. Thus, while the approach was quite successful in being used as part of a safety assurance architecture, sufficient care must be taken in order to minimize the quantity of spurious unsafe determinations that result from the over-approximation of reachable sets. While allocating more wall-time to the reachability regime is a possible solution, it is also worth considering other set representations while maintaining real-time guarantees. Moreover, one may also consider alternate reachability formulations, such as backward reachability regimes. However, these approaches come at the cost of significant computational overhead.

The second challenge is that the real-time reachability regime was designed to reason about relatively short time horizons, and there is an assumption that the control decision remains fixed throughout the reach-set construction. This assumption causes the verification results to be conservative in nature, and in future work we hope to expand this work to consider real-time closed-loop reachability analysis. The difficulty in performing closed-loop reach-set generation lies in developing accurate sensor models. As an example, for end-to-end control based on camera images, it is not clear how to generate camera images based on the state of the system to provide a meaningful and useful reachable set.

Lastly, in recent years, there has been a growth in approaches that perform online parameter estimation for dynamic obstacles within a robot's environment. In this work, we considered a simple two-dimensional kinematic model for the opponent vehicles within the racetrack environment. At low speeds, this model performs quite well; however, at higher speeds, these models would need to incorporate more sophisticated dynamics. In future work, we hope to evaluate online system identification within our framework.

V.8 Conclusion

In this manuscript, we presented a runtime verification framework leveraging real-time reachability and the simplex architecture for the safety assurance of a 1/10 scale autonomous vehicle called the F1/10 platform. The central idea behind our approach lies in computing the set of reachable states of the F1/10 system and ensuring that it never collides with both static and dynamic obstacles within its environment. Rather than analyzing the correctness of the controllers commanding the behavior of the F1/10, the reachability regime is leveraged to focus on the effects of a controller's decisions on the system's future states. In the event of a potential safety violation, a safety controller can be engaged in order to maintain safety.

One of the key benefits of utilizing reachability regimes for the design of safety assurance frameworks is that they are quite adept in handling uncertainty, and in this work we presented a rigorous analysis of the effects of several classes of uncertainty in reasoning about the correctness of the system. Specifically, we allowed for the consideration of uncertainty with respect to the model of the underlying system, as well as

uncertainty with respect to measuring the state of the vehicle's environment. Our experiments, conducted both in simulation and on an embedded hardware platform, validate the real-time aspects of our approach. Moreover, they demonstrate the efficacy of the simplex architecture in ensuring safety in different scenarios.

Improving the over-conservativeness of the reachability framework, considering closed-loop reach-set generation, evaluating backward-reachability frameworks, making use of real-time operating systems, and incorporating dynamic obstacles into our regime are left for future work. Additionally, we wish to consider online learning applications, as our regime can be applied to such schemes with minimal modifications. Finally, our future work will consider the development of a verified safety controller and switching logic in order to maximize the benefits of the provable guarantees of our reachability framework.

CHAPTER VI

Integrating Online Reachability Analysis with Model-Predictive Control for Dynamic Obstacle Avoidance

This chapter[1] presents an optimisation based approach for a static and dynamical obstacle avoidance problem within an autonomous vehicle racing context. Our control regime leverages online reachability analysis and sensor data to compute the maximal safe traversable region that an agent can traverse within the environment. The idea is to first compute a non-convex safe region, which then can be convexified via a novel coupled separating hyperplane algorithm. This derived safe area is then used to formulate a nonlinear model-predictive control problem that seeks to find an optimal and safe driving trajectory. We evaluate the proposed approach through a series of diverse experiments and assess its runtime requirements through an analysis of the effects of a set of diverse optimisation objectives for generating these coupled-hyperplanes.

VI.1 Introduction

Over the last several years, autonomous racing has actively been pursued as a strategy to explore edge-case scenarios in autonomous driving [273]. Racing scenarios present unique challenges with respect to navigating high speeds and multi-agent interactions. In these contexts, vehicles must be able to operate at the edge of their operating envelopes in close-proximity to static and dynamic obstacles. Several competitions have emerged over the last couple of years, such as the Indy Autonomous Challenge (IAC) [273], and the F1Tenth International Autonomous racing competition [2].

Although numerous racing strategies have been proposed over the last several years, head-to-head racing at high speeds remains a challenge. Unlike the time trials that are frequently used as qualification rounds in these competitions [174], head-to-head racing requires designing a regime that is able to anticipate the actions of the opponent vehicles and navigate through the track as fast as possible while avoiding collisions.

Within the autonomous racing space, one of the most popular frameworks for tackling the racing problem has been formulating and solving an optimisation problem that balances obstacle avoidance and travelling at high velocities [274, 275]. In this context, the model predictive control framework (MPC), which finds optimal control commands based on a model of the underlying system, while satisfying a set of constraints is the most widely used approach [276].

Although MPC approaches have enjoyed wide success in these settings [274], one of the main limitations exhibited by many approaches is obtaining robust online estimations of risk when operating in dynamic and

[1] This chapter has been submitted to a conference for review.

uncertain environments. Particularly around vehicle-to-vehicle interactions. While a lot of progress has been made in this area, collisions still occur due to misplaced predictions of the set of all actions that a vehicle and environmental participants could pursue [275]. Furthermore, as Katrakazas et al. note "exhaustively calculating and predicting the trajectories of other traffic participants at each epoch incurs a huge computational cost." Currently, many existing approaches treat the vehicle as an isolated entity, and the behavioural models of the other participants within the environment have not yet been widely incorporated into MPC regimes [275].

One of the ways that this challenge has been addressed has been through the use of reachability analysis approaches [48]. The idea is to compute the set of states that the other racing agents could occupy in the future, for a fixed time horizon, and plan trajectories for the ego vehicle that avoids this set [241, 114, 277]. These sets allow for modelling the inherent uncertainty in the behaviour of other agents and for the synthesis of safe racing trajectories [48]. There are two main challenges that arise in these contexts. The first is that over long time horizons, reachability approaches will result in overly conservative behaviours as the set of avoidable states grows. The second is that reachability approaches are typically computationally challenging endeavours, thus leveraging them online is quite challenging. In light of these challenges, the following chapter presents a model predictive control framework leveraging real-time reachability for a 1/10 scale autonomous vehicle test-bed in a multi-agent racing setting modeled after the F1/10 International Autonomous Racing Competition.

Finally, obtaining a solution to the MPC problem generally entails solving a convex optimisation problem, which guarantees convergence to a globally optimum solution. However, due to the presence of static and dynamic obstacles, formulating the optimal control problem as an obstacle avoidance results in solving a non-convex problem. Therefore, to solve this problem efficiently, many approaches leverage state-space convexification. In the past, several state-space convexification approaches have been proposed, including: region partitioning [278], computing separating hyperplanes [279, 280], and constructing approximations using stored data points [281] (further discussed in Section VI.2). In our framework, we propose a novel optimisation-based approach for convexifying non-convex state-spaces by computing coupled separating hyperplanes. The coupling of separating hyperplanes makes it possible to compute optimal safe and convex regions. However, it comes at the cost of increased computation time. Therefore, in this chapter, we investigate the feasibility (e.g. timing constraints) of computing coupled separating hyperplanes in a real-time autonomous racing scenario.

In summary, the contributions of this chapter are:

1. We propose a novel closed-loop model predictive obstacle avoidance controller that integrates online

reachability analysis and an optimisation based state space convexification approach.

2. We evaluate this approach across a diverse set of simulation experiments using the F1/10 simulation platform. These experiments include varying the number of dynamic agents, the number of static and dynamic obstacles, and the racing environment.

3. We present a timing analysis of the state space convexification approach.

4. Finally, we evaluate our approach against the well-known model predictive contouring control approach, which has shown great success in obstacle avoidance tasks.

VI.2 Related Work

Researchers have approached the obstacle avoidance problem from two major perspectives. The first strategy has involved formulating and solving an optimisation problem. While the second regime has typically involved a hierarchical decomposition of path planning and reference tracking. A variety of algorithms such as artificial potential fields [282], genetic algorithms [283], rapidly-exploring random trees (RRT) [284], fuzzy logic algorithms [285], elastic band theory [286], and rolling window methods [287] have demonstrated success in numerous arenas. A key limitation of many path planning approaches is that they are incapable of respecting kinodynamic constraints, such as bounds on the acceleration, and often the trajectories must be passed to a low-level controller that utilizes a higher fidelity dynamics model and respects control constraints [274]. Furthermore, in highly dynamic and uncertain environments, to maintain safety, planners must be able to replan sufficiently fast to react appropriately to split-second environmental threats [284]. However, the majority planners typically do not replan sufficiently rapidly to ensure split-second reactivity to threats [241].

As mentioned previously, MPC approaches have demonstrated great success in generating optimal trajectories that respect kinodynamic constraints and recently researchers have combined these approaches with reachability analysis to generate provably free paths [48, 96, 241, 288, 241]. Within this regime, [48, 288, 114] utilise forward reachability methods to eliminate areas of the state space that would result in collisions. While these methods are extremely effective, these approaches must be implemented carefully in order to ensure that the resulting trajectories do not result in overly conservative behaviours [241]. The alternative to these approaches are backward reachability approaches [96, 241] which utilise a target set representing a set of undesirable states, in order to design controllers that can guarantee dynamic obstacle and static dynamic obstacle in a minimally interventionist approach. However, these approaches are computationally demanding and typically the safety-ensuring control constraints, derived from these methods, are computed and cached offline before being incorporated into an MPC problem [241].

Beyond reachability methods, over the last several years, several space convexication approaches for the obstacle avoidance problem have been proposed. In [289] a feasible convex set for the model predictive control is obtained by computing two parallel time-varying hyperplanes on racetrack borders (visualised in Fig. 4 [289]). However, the resulting hyperplanes do not consider static obstacles nor dynamic agents. The works of Mercy et al. [280, 290] and Scholte et al. [291] utilise the concept of separating hyperplanes to compute a single (or multiple) hyperplane which separate autonomous systems from convex obstacles. Finally, in [278] two (polar and convex) different types of convexication methods based on region partitioning for obstacle avoidance were proposed. Their convex partitioning regime utilises a convex partitioning algorithm [292] to compute the minimum number of convex shapes needed to capture non-convex obstacles, whereas the polar partitioning approach computes derives a safe set using a minimum number of triangles.

In our approach, we express the problem of computing separating hyperplanes as an optimisation problem, where we are not only interested in computing the correct set of separating hyperplanes but also the optimal set of separating hyper-planes. As an example, one of our approaches seeks to obtain a set of hyper-planes that maximises the area of the traversable region for the autonomous vehicle. Furthermore, our proposed approach for obtaining a set of coupled separating hyperplanes can also handle non-convex obstacles.

VI.3 Preliminaries

VI.3.1 F1/10 Platform

The F1/10 platform of O'Kelly et al. [174] was originally designed to emulate the hardware and software capabilities of full scale autonomous vehicles. The platform is equipped with a standard suite of sensors such as stereo cameras, LiDAR (light detection and ranging), and inertial measurement units (IMU). The platform uses is built on the *Robot Operating System* (ROS) and supported by a Gazebo-based simulation environment [174]. The result is a platform that allows researchers to conduct real-world experiments that investigate planning, networking, and intelligent control [174]. We utilise this simulation environment for a number of experiments and for training our controllers.

VI.3.2 Model Predictive Control

Model Predictive Control (MPC), which is also known as Receding Horizon Control, is a widely used method for dealing with control problems with multivariate constraints [293]. It has been used extensively in process control industries due to its demonstrated ability to yield high performance for long periods of time without human intervention [150]. While its early inspirations were in petro-chemical processes [150], MPC has also been utilised within autonomous applications for path planning, obstacle avoidance, and in stability and

reference tracking for aircraft systems [294].

Let's suppose we have the following (VI.1) discrete-time system where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$ and $t \in \mathbb{N}$.

$$\mathbf{x_{t+1}} = f(\mathbf{x_t}, \mathbf{u_t}) \tag{VI.1}$$

The MPC problem can then be expressed as a finite horizon optimisation problem (VI.2) where a cost function J is being minimised over a finite time horizon N subject to constraints (2.1 - 2.4).

$$J_{t\to t+N}(\mathbf{x}_t) = \min_{\mathbf{u_0},\mathbf{u_1},..\mathbf{u_{N-1}}} p(\mathbf{x}_{t+N}) + \sum_{k=t}^{t+N-1} q(\mathbf{x}_k, \mathbf{u}_k) \text{ s.t. :} \tag{VI.2}$$

$$\mathbf{x_{k+1}} = f(\mathbf{x_k}, \mathbf{u_k}),\ \forall k \in \{t, ..., t+N-1\} \tag{2.1}$$

$$\mathbf{x_0} = \mathbf{x}_s \tag{2.2}$$

$$\mathbf{x_k} \in \mathcal{X},\ \forall k \in \{t, ..., t+N-1\} \tag{2.3}$$

$$\mathbf{u_k} \in \mathcal{U},\ \forall k \in \{t, ..., t+N-1\} \tag{2.4}$$

The cost function J is made up of a stage cost function q and a terminal cost function p which determine the cost of being at the interim state $\mathbf{x}_k$ after applying an input $\mathbf{u}_k$, and the cost of being at the final state $\mathbf{x}_{t+N}$. The constraints (2.1 - 2.4) assert that the optimisation problem, given by equation (VI.2), begins from an initial state $\mathbf{x_s}$ and that the interim state and control inputs must respect the constraint sets $\mathcal{X}$ and $\mathcal{U}$.

If the dynamics and constraints can be formulated as linear expressions, then the MPC problem can be solved efficiently using standard convex optimisation techniques. However, if the dynamics or constraints are nonlinear, then the problem becomes a non-linear optimisation problem that is much more computationally challenging to solve. However, allowing for nonlinear dynamics and constraints may permit one to track complex systems with a higher level of fidelity than using linear expressions. Thus, the computational cost must be evaluated against overall system performance [295].

VI.3.3 Reachability Analysis

Reachability analysis is a technique for computing the set of all reachable states for a dynamical system, beginning from a set of initial states. The reachable set of $\mathcal{R}_t$ at time t can be defined formally as:

$$\mathcal{R}_t(\mathcal{X}_0) = \{\varsigma(t, x_0, u) \,|\, x_0 \in \mathcal{X}_0, u(s) \in \mathcal{U}, \forall s \in [0, t]\} \tag{VI.3}$$

where $\mathcal{X}_0 \subseteq \mathbb{R}^n$ represents the set of initial states, $\mathcal{U} \subseteq \mathbb{R}^m$ represents the input set and $\varsigma(t, x_0, u)$[2] is the unique solution of the ODE describing the system's dynamics, $\dot{\mathcal{X}}(t) = f(x(t), u(t))$. More generally, reachability analysis methods aim to construct a *conservative* flowpipe (VI.4) which encompasses all the possible reachable sets of a dynamical system over a time-horizon $[0, T]$. This can be formalized as follows:

$$\mathcal{R}_{[0,T]}(\mathcal{X}_0) = \bigcup_{t \in [0,T]} \mathcal{R}_t(\mathcal{X}_0). \tag{VI.4}$$

Reachability analysis has been widely used in applications which range from the formal verification of systems to problems relating to the safe synthesis of complex systems [296]. The majority of reachability analysis approaches leverage a combination of numerical analysis techniques, graph algorithms, and computational geometry [51, 39], and while in some cases it is possible to derive the exact reachable set of states, for many classes of systems computing the exact reachable set is infeasible. Thus, deriving the reachable set for these classes of systems involves obtaining a sound approximation of this set using a variety of set representations. Consequently, there is an inherent tradeoff between the accuracy of the approximation and the time it takes to construct this set. However, the last decade has witnessed the development of a number of reachability tools and set representations proposed towards solving this problem including SpaceEx [297], Checkmate [298] or Flow* [299] to name a few. We refer interested readers to the following papers for an in-depth discussion of these techniques: [51, 39].

VI.4 Problem Formulation and Space Convexication

VI.4.1 Problem Formulation

In this chapter, we consider the general autonomous racing problem (VI.5), where a model predictive controller, ((VI.2)), is tasked with generating a sequence of control inputs $u_{0\ldots K}$, $u \in \mathbb{R}^m$, that steer a vehicle modeled by dynamics, $f(x_t, u_t)$ (VI.1), such that it reaches the terminal state $\mathbf{x}_f \in \mathcal{X}_f$ starting from an initial state $\mathbf{x}_s \in \mathcal{X}$, where $\mathcal{X}$ and $\mathcal{X}_f$ are the initial and terminal sets respectively. The goal is to steer the vehicle into the terminal set in the least amount of time T.

[2] Our assumption is that f is globally Lipschitz continuous. This property guarantees the existence and uniqueness of a solution for every initial condition in χ_0.

$$\min_{\mathbf{T},\mathbf{u_0},\mathbf{u_1},..\mathbf{u_{N-1}}} p(\mathbf{x}_T) + \sum_{k=0}^{T} q(\mathbf{x}_k, \mathbf{u}_k) \quad s.t.$$
$$x_{t+1} = f(x_t, u_t),\, x_0 = x_s$$
$$x_t \in \mathcal{X},\, u_t \in \mathcal{U} \tag{VI.5}$$
$$x_T = \mathcal{X}_F$$

In our formulation, the autonomous vehicle operates within a two-dimensional environment $\mathcal{W} \subset \mathbb{R}^2$ enclosed by boundaries $\{\delta\mathcal{W}_0, \delta\mathcal{W}_1.., \delta\mathcal{W}_i\}$ as $\delta\mathcal{W} \subset \mathcal{W}$, among a set of dynamic agents $\zeta = \{\zeta_0, ..., \zeta_i\}$ and static obstacles $\{\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2..., \mathcal{O}_i\}$ with $\mathcal{O}_i \subset \mathcal{W}$. The region of space occupied a dynamic agent ζ_i in the environment over a time interval $[0, T]$ from its current position x_0 is given by its reachable set $\mathcal{R}_{\zeta_i,[0,T]}(x_0) \subset \mathcal{W}$. Our assumption is that the static and dynamic obstacles are contained within the two-dimensional environment and that their position does not overlap with its boundaries. Furthermore, we refer to opponent vehicles within the racing environment as dynamic agents, and refer to all other dynamic entities as dynamic obstacles.

To obtain a globally optimal solution to problem (VI.5), as opposed to a locally optimal solution, the model predictive control problem requires the $\mathcal{X}$ to be convex. However, because of environment borders, static obstacles and dynamic agents, $\mathcal{X}$ is generally a non-convex entity. Therefore, the main sub-problem we are addressing in this chapter is the computation of the safe, convex and *optimal* state-space $\mathcal{X}_{safe}$ (VI.6) in which a safe trajectory starting from x_0 to a target location $l_\zeta \in \mathcal{X}_{safe}$ could be generated using model-predictive control for the autonomous system (see Fig. VI.1). The safe portion of the state space $\mathcal{X}_{safe}$ can be defined as follows:

$$\mathcal{X}_{safe} = \{x \,|\, x \notin (\delta\mathcal{W} \cup \mathcal{O} \cup \bigcup_{i=0}^{|\zeta|-1} \mathcal{R}_{\zeta_i,[0,T]}(x_0))\}. \tag{VI.6}$$

The computation of $\mathcal{X}_{safe}$ requires only considering the *observable* obstacles, agents and borders. To define the set of observable points, we first introduce our notion of the LiDAR sensor, which is mounted on our autonomous system and makes it possible to determine the distance to obstacles. The sensor sends N light pulses in an anti-clockwise direction around the autonomous system in $\delta\theta$ increments and returns the set of observational points $\{r_0(x_t), ..., r_N(x_t)\}$ where each LiDAR observation $r_i(x_t) \in \mathbb{R}$ in the direction θ_i can be formally defined in the following way (VI.7):

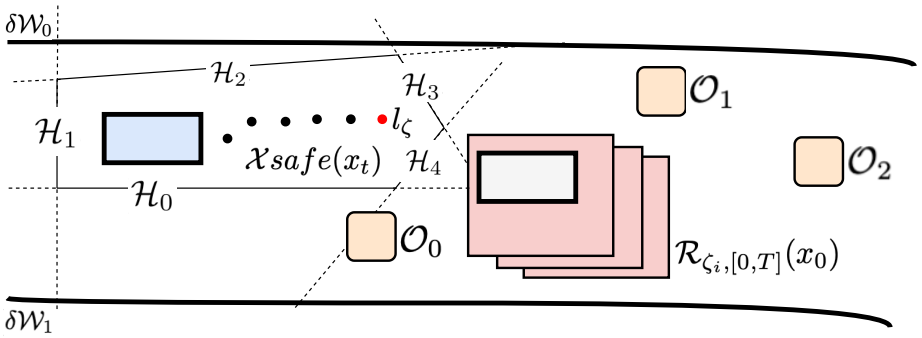

Figure VI.1: Visualisation of the autonomous racing problem with track boundaries, $\{\delta\mathcal{W}_0, \delta\mathcal{W}_1\}$, a dynamic opponent described its reachset $\mathcal{R}_{\zeta_i,[0,T]}(x_0)$ and static obstacles $\{\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2\}$. In this figure, the blue rectangle corresponds to the ego vehicle, the white rectangle corresponds to a dynamic opponent. The main sub-problem is computing an n-number of separating hyperplanes $(\mathcal{H}_0 ... \mathcal{H}_4)$ which jointly create a polyhedron $\mathcal{X}_{safe}$. The computed $\mathcal{X}_{safe}$ must contain an ego vehicle and its target location l_ζ as well as exclude observable obstacles.

$$\begin{aligned} r_i(x_t) = \min_{\mathcal{O}_i \in \mathcal{O}} \min_{z \in \mathcal{O}_i} ||z - \zeta(x_t)||_2 \\ s.t. \quad \text{atan2}(z - \zeta(x_t)) = \theta_i \end{aligned} \tag{VI.7}$$

The observable LiDAR signals $r_i(x_t)$ can be converted into a two-dimensional point cloud of the $\mathcal{W}$ where a single point $p_i(x_t)$ of an agent $\zeta(x_t)$ can be defined as a tuple (VI.8):

$$p_i(x_t) = (\zeta(x_t) + r_i(x_t)\cos\theta_i,\ \zeta(x_t) + r_i(x_t)\sin\theta_i) \tag{VI.8}$$

Therefore, we can define the observable static obstacles of $\zeta(x_t)$ as a set $\mathcal{Q}_{ob}$ of LiDAR points.

$$\begin{aligned} \mathcal{Q}_{ob} = \{q \mid q \in \{p_0, ..., p_{N-1}\} \wedge ||q - \zeta(x_t)||_2 \leq d\} \\ d \in \mathbb{R},\ 0 < d \leq \max(r_0(x_t), ..., r_{N-1}(x_t)) \end{aligned} \tag{VI.9}$$

Furthermore, we want the observable unsafe space $\mathcal{Q}_{ob}$ to include reachable sets of other dynamic agents. However, we only interested in the other agents which are *close enough* to the ego vehicle, so we update our definition $\mathcal{Q}^+_{ob}$ to include reachable regions dynamic agents that are *close* to the ego vehicle.

$$\mathcal{Q}^+_{ob} = \mathcal{Q}_{ob} \cup \{q | q \in \mathcal{R}_\zeta(x_0, \Delta t) \wedge ||q - \zeta(x_t)||_2 \leq d\} \tag{VI.10}$$

VI.4.2 Space Convexication via Separating Coupled-Hyperplanes

In this chapter, we propose a solution for the computation of $\mathcal{X}_{safe}$ which is based on the convexication of non-convex state space via separating coupled-hyperplanes. A hyperplane $\mathcal{H}$ is a set which splits set $\mathbb{R}^n$ into two halfspaces and is formally defined as follows (VI.11):

$$\mathcal{H} = \{x \mid a^T x = b\} \quad where \quad a \in \mathbb{R}^n, b \in \mathbb{R}, a \neq 0 \tag{VI.11}$$

Let's also denote $\mathcal{H}^*$ (VI.12) as one of the halfspaces of the hyperplane $\mathcal{H}$. Then, a separating hyperplane $\mathcal{H}$ is said to separate two disjoint convex sets A, B such that $A \subseteq \mathcal{H}^+$ and $B \subseteq \mathcal{H}^-$ [300].

$$\begin{aligned} &\mathcal{H}^* \in \{\mathcal{H}^+, \mathcal{H}^-\} \quad \mathcal{H}^+ \cap \mathcal{H}^- = \mathcal{H} \\ &\mathcal{H}^+ = \{x \mid a^T x \geq b\} \quad \mathcal{H}^- = \{x \mid a^T x \leq b\} \end{aligned} \tag{VI.12}$$

While an intersection of finite halfspaces is a polyhedron $\mathcal{P}$ (VI.13):

$$\mathcal{P}_{\mathcal{H}} = \{x \mid x \in \bigcap_{i=0}^{N-1} \mathcal{H}_i^*\} \tag{VI.13}$$

The idea behind non-convex space convexication via separating coupled-hyperplanes is to compute a set of hyperplanes $\mathcal{HS} = \{\mathcal{H}_0, ..., \mathcal{H}_n\}$ such that there exists a polyhedron $\mathcal{P}$ which does not intersect with the set of observable obstacles $\mathcal{Q}_{ob}^+$, and includes the autonomous system $\zeta(x_t)$ and its target location l_ζ at time t. Indeed, that polyhedron is $\mathcal{X}_{safe}$:

$$\begin{aligned} \mathcal{X}_{safe} = \{x | x \in &\mathcal{P}_i \wedge \mathcal{P}_i \cap \mathcal{Q}_{ob}^+ = \emptyset \wedge \zeta(x_t) \in \mathcal{P}_i \wedge \\ &\wedge l_{\zeta(x_t)} \in \mathcal{P}_i\} \end{aligned} \tag{VI.14}$$

The problem of generating a set of separating coupled-hyperplanes $\mathcal{HS}$ can be defined as a satisfiability (or optimisation) problem (VI.15) in which a n number of hyperplanes are computed such that: 1) each hyperplane separates a part of observable obstacles from the ego car and its target location and 2) all the observable obstacles are separated by separating coupled-hyperplanes.

$$
\begin{aligned}
find \quad & \mathcal{HS} = \{\mathcal{H}_0, ..., \mathcal{H}_i\} \quad s.t. \\
& \forall \mathcal{H}_i \in \mathcal{HS} \Rightarrow \exists q_i^{ob+} \subseteq \mathcal{Q}_{ob}^{+} \wedge q_i^{ob+} \subseteq \mathcal{H}_i^{*} \wedge \\
& \wedge \zeta(x_t), l_{\zeta(x_t)} \in \mathbb{R}^n \setminus \mathcal{H}_i^{*} \\
& \bigcup_{i=0}^{N-1} q_i^{ob+} = \mathcal{Q}_{ob}^{+}
\end{aligned}
\tag{VI.15}
$$

From the obtained set of separating coupled-hyperplanes $\mathcal{HS}$, the convex and safe polyhedron $\mathcal{X}_{safe}$ is the intersection of halfspaces $\mathcal{H}_i^*$ of each hyperplane $\mathcal{H}_i \in \mathcal{HS}$ for which $\zeta(x_t) \in \mathcal{H}_i^*$ holds (ego vehicle belongs to the halfspace).

As mentioned, a satisfiability problem (VI.15) can also be expressed as an optimisation problem on the set of hyperplanes $\mathcal{HS}$ or polyhedron $\mathcal{P}_{\mathcal{HS}}$. One possible *performance* metric could be finding the largest $\mathcal{P}_{\mathcal{HS}}$, in turn, giving model predictive control more state-space for exploration. In the literature, the problem of finding the largest convex polygon contained in a non-convex polygon or with the smallest Hausdorff distance is sometimes referred to as the Hausdorff core problem [66] or the potato-peeling problem [301]. Nonetheless, there is a clear trade-off between the computation time need for the optimisation problem and overall *performance*. Generally, the computation of separating coupled-hyperplanes must meet control system timing requirements. Furthermore, simply obtaining the largest polyhedron might not always be the most optimal solution for the autonomous racing problem.

VI.5 Autonomous Vehicle Control System

VI.5.1 Overview of the Closed-Loop Control System

The closed-loop control system for obstacle avoidance which we are proposing in this chapter combines online reachability analysis and non-linear model-predictive control (visualised in Fig. VI.2). The control cycle can be divided into four main procedures: sensing, environment data processing and local planning, state-space convexification and solving the optimal control problem.

In the following paragraphs, we briefly overview each of the control phases and then in the following sections we provide more detail on our main technical contributions: state-space convexication, online reachability analysis and their integration into model predictive controller.

The control system relies on the LiDAR sensor to obtain and identify the set of observable obstacles and safe regions. we then leverage the Ramer-Douglas-Peucker algorithm [302] to simplify the observed LiDAR data and reduce the noisiness of its measurements. Doing so allows us to reduce the computation time needed to produce a set of coupled separating hyperplanes. The other sensors, namely, odometry measurements and

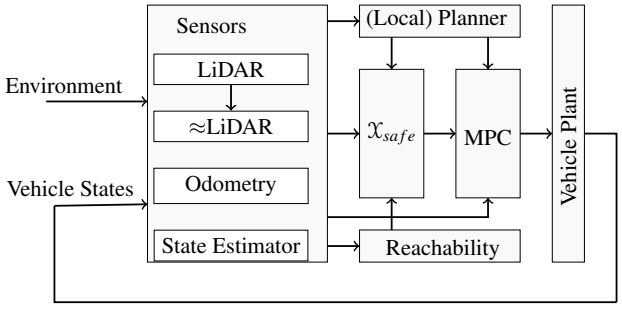

Figure VI.2: The architecture of the closed-loop control system for obstacle avoidance

the results of state-estimators, are used to determine the state of the ego vehicle and other agents respectively. In this work, we assume that the state of the ego vehicle and opponent agents are estimated perfectly. Therefore, we use the ground truth data provided by the simulator. This data is then passed to a (local) planner (e.g. Follow-the-Gap [303]) to select a target position. We then use reachability analysis to compute the set of reachable states for all agents within the environment.

The computation of separating coupled hyperplanes, which produces a safe and convex $\mathcal{X}_{safe}$, involves using sensor information, the target location obtained from the local planner, and the set of reachable states of the dynamic agents within the environment. The hyperplanes are then passed to the model predictive controller, together with the target location and odometry data, which then solves an online optimal control problem to determine the optimal inputs for the vehicle.

VI.5.2 Computing Separating Coupled-Hyperplanes

The problem of computing separating coupled hyperplanes, which establishes a convex and safe $\mathcal{X}_{safe}$, can be formulated as an optimisation (or satisfiability) problem. Thus, we present an optimisation-based method for solving (VI.15) in order to separate the observable obstacle set $\mathcal{Q}_{ob}^{+}$ from the autonomous system $\zeta(x_t)$ and its target location $l_{\zeta(x_t)}$ at the state x_t.

In Algorithm 2, we describe the computation of our separating coupled-hyperplanes $\mathcal{H}_{0..n}$. First, the set of unsafe states is included in the set $\mathcal{Q}_{ob}$. The set of unsafe states consists of the set of observable obstacles from the LiDAR sensor and the reachable states of the dynamic agents. Using this set, we then make use of the state of the ego vehicle, the target location obtained from the local planner, and a predefined number of hyper-planes to formulate a constrained optimization problem. Furthermore, only obstacles within a distance d (Alg. 2 ln. 6) and reachable sets which overlap with the safety envelope (Alg. 2 ln. 7-8) of the $\zeta(x_t)$ system are considered in the hyperplane computation.

Our approach uses a derivative-free constrained optimisation formulation which utilizes a linear approxi-

mation of the objective function and optimisation constraints to solve the aforementioned optimization problem [304]. In the optimisation problem, an individual separating hyperplane $\mathcal{H}_n \in \{\mathcal{H}_0, .., \mathcal{H}_n\}$ is only *responsible* for separating a subset of $\mathcal{Q}_{ob}$ from $\zeta(x_t)$ and $l(x_t)$, while the set of all hyperplanes considered should separate the vehicle from $\mathcal{Q}_{ob}$ as a whole.

Algorithm 2: Algorithm for computing separating coupled-hyperplanes

Input: $d \in \mathbb{R}^+, n \in \mathbb{N}$
Input: $\zeta(x_t) \leftarrow \{e_x, e_y, \psi\}$
Input: $\{r_0(x_t), ..., r_N(x_t)\} \leftarrow \text{LiDAR}(\zeta(x_t))$
Input: $l(x_t) \leftarrow \text{Local-Planner}(\zeta(x_t), \{r_0(x_t), ..., r_N(x_t)\})$
  $\mathcal{R}_{0,..,i} \leftarrow \text{State-Estimator}(\zeta_0, ..., \zeta_i)$
  $\mathcal{Q}_{ob}(x_t) \leftarrow \{r \,|\, ||r_n - \zeta(x_t)||_2 \leq d\}$
  for any $\mathcal{R}_n \in \mathcal{R}_{0,..,i}$ **that** $\mathcal{R}_n \cap \{x \,|\, ||x - \zeta(x_t)||_2 \leq d\} \neq \emptyset$
    $\mathcal{Q}_{ob}(x_t)' \leftarrow \mathcal{Q}_{ob}(x_t) \cup \mathcal{R}_n$
  if feasible
    $\mathcal{H}_{0..n} \leftarrow \mathsf{COP}(\mathcal{Q}_{ob}, \zeta(x_t), l(x_t), n)$ (**C**onstrained **O**ptimisation **P**roblem [304])

For each $q_{ob} \in \mathcal{Q}_{ob}$, a separate constraint in the optimisation problem can be defined which checks if q_{ob} is on the right-hand-side of its associated hyperplanes $\mathcal{H}_n$. Similarly, each hyperplane can be classified to see if the target location and autonomous system $\zeta(x_t)$ lie on the left hand-side of the hyper-plane. As previously discussed, different objective functions characterizing how the set of hyperplanes are derived can be used, for example, minimizing the distance between each $\mathcal{H}_n$ and its associated set of q_{ob}. We present an analysis of different optimisation objective functions for this purpose in section VI.6.3.

VI.5.3 Model Predictive Control

We now define the objective function and MPC Formulation utilized in this work [305]. Let $\mathbf{x}_g = (x_t, y_t, \psi_t)$ be the target position and orientation that we are we seeking to achieve through the use of mpc, and let u_v denote the throttle input provided to the vehicle.

$$\min_{\mathbf{u_0}, \mathbf{u_1}, .. \mathbf{u_{N-1}}} p(\mathbf{x}_{t+N}) + \sum_{k=t}^{t+N-1} q(\mathbf{x}_k, \mathbf{u}_k) + \Delta u_k^T R u_k \quad \text{s.t.:} \tag{VI.16}$$

$$\mathbf{x_{k+1}} = f(\mathbf{x_k}, \mathbf{u_k}),\ \forall k \in \{t, ..., t+N-1\} \tag{VI.17}$$

$$\mathbf{x_0} = \mathbf{x}_s \tag{VI.18}$$

$$\mathbf{x_k} \in \mathcal{X},\ \forall k \in \{t, ..., t+N-1\} \tag{VI.19}$$

$$\mathbf{u_k} \in \mathcal{U},\ \forall k \in \{t, ..., t+N-1\} \tag{VI.20}$$

$$A\mathbf{x_k} < b \tag{VI.21}$$

where:

$$\Delta u_k = u_k - u_{k-1} \tag{VI.22}$$

$$q(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}_k - \mathbf{x}_g)^2 - u_v \tag{VI.23}$$

$$p(\mathbf{x}_{t+N}) = (\mathbf{x}_k - \mathbf{x}_g)^2 \tag{VI.24}$$

and R in the above equation is a diagonal matrix with the values 0.5 and 0.5 which represent the penalty for changes in the throttle and steering commands provided to the vehicle. This penalty is often used to smoothen the obtained optimal solution [305]. In the above equations, (21) corresponds to the half-space constraints describing the traversable region of the vehicle. In our formulation, we designed our objective function to minimize the distance to the target position as well as maximize the speed input provided to the vehicle throughout the trajectory.

The dynamics of the vehicle, $f(x_t, u_t)$, (17), can be modeled using a simplified kinematic bicycle model where we neglect modeling of aerodynamics and tire forces. The equations of motion are given below:

$$\begin{aligned}
\dot{x} &= v \cdot \cos(\psi + \beta) \\
\dot{y} &= v \cdot \sin(\psi + \beta) \\
\dot{\psi} &= \frac{v}{l_r} \cdot \sin(\beta) \\
\dot{v} &= a \\
\beta &= \arctan(\frac{l_r}{l_r + l_f} * \tan(\psi))
\end{aligned} \tag{VI.25}$$

where x and y are the car's position, a is acceleration, v is the vehicle's velocity, ψ is the steering input, β is the slip angle, and l_f and l_r are the distances from the car's center of mass to the front and rear respectively [306]. The kinematic bicycle model is characterized by relatively few parameters and has been show to track reasonably well at low speeds [227].

Since the dynamics described by Equation (VI.25) are nonlinear, our MPC formulation becomes a non-linear MPC problem. As a result, in general, this results in a non-convex optimization problem [307]. While it is possible to linearize Equation (VI.25) and convert the problem into a linear MPC formulation, we elected not to do so. Primarily because of the recent development of tools that can solve NMPC problems efficiently, as well a desire to try and capture the nonlinear dynamics of the F1/10 vehicle with a higher degree of fidelity. We used the software-tool *do-mpc* [308] to solve the nonlinear model predictive problem, due to its modular

and transparent implementation as well as its efficient handling of nonlinear optimization problems. The tool utilizes an orthogonal collocation of finite-elements method to discretize the NMPC problem into a form that can be comfortably handled by state-of-the-art opmtization tools. We refer interested readers to the following chapter for an in depth discussion of do-mpc [308].

VI.5.4 Reachability Analysis of Dynamic Obstacles

To perform reachability analysis, we first identify a dynamical model of the vehicle, and assume models for the dynamic obstacles within its environment.

VI.5.4.1 Dynamic Obstacle Model

The obstacle tracking problem is a well studied and challenging topic within the autonomous vehicle, computer vision, and robotics literature [262]. Typically, some assumptions are required in order to constrain the tracking problem to suit the context of the application. In our framework, we assume that the obstacles are described by a two-dimensional kinematic model and a corresponding bounding box. The equations describing the ODE are given as follows:

$$\dot{x} = v_x, \dot{y} = v_y$$

where v_x and v_y are the velocities in the x and y direction, respectively. Additionally, we make the assumption that we have access to the position and velocity of the other race participants.

While it is possible to use more sophisticated models to describe the behavior of the dynamic obstacles within the vehicles environment, for simplicity we selected a two-dimensional kinematic model. However it is worth noting that there has been a growth in approaches that perform online parameter estimation for dynamic obstacles within a robots environment through online system identification [309].

VI.5.4.2 Online Reachability Computation

Using the dynamics models obtained in the previous sections, the crux of the real-time reachability algorithm is computing the set of reachable states $\mathcal{R}_{[0,T]}(\mathcal{X}_0)$ over a finite time horizon. The algorithm utilized within this work is based on mixed face-lifting, which is part of a class of methods that deal with *flow-pipe construction* or *reachtube computation* [106]. This is done using snapshots of the set of reachable states that are enumerated at successive points in time, as outlined in Equation VI.3.

In general, it is not possible to obtain the exact reachable set $\mathcal{R}_{[0,T]}(\mathcal{X}_0)$, so we compute an over-approximation such that the actual system behavior is contained within the over-approximation [235]. The algorithm utilized in this work utilizes n-dimensional hyper-rectangles ("boxes") as the set representation to generate

reachtubes [106]. An example of these reachtubes can be seen in Figure VI.4. Over long reach-times, the over-approximation error resulting from the use of this representation can be problematic. However, for short reach-times it is ideal in terms of its simplicity and speed [108].

Traditionally, reachability approaches have been executed offline because they are computationally intensive endeavors. However, in [108, 106], Bak et al. and Johnson et al. presented a reachability algorithm, based on the seminal mixed face-lifting algorithm [221], capable of running in real-time on embedded processors. The algorithm is implemented as a standalone C-package that does not rely on sophisticated (non-portable) libraries, recursion, or dynamic data structures and is amenable to the anytime computation model in the real-time scheduling literature. In this regime, each task produces a partial result that is improved upon as more computation time is added [106]. We refer readers to the following papers for an in depth treatment of these procedures [221, 108, 106].

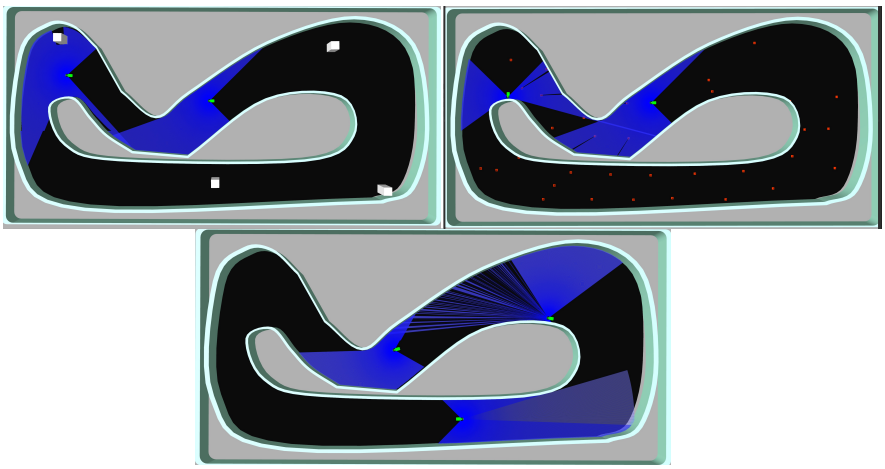

Figure VI.3: Our experiments included two and three vehicle races, as well as an evaluation in the presence of static and dynamic obstacles. The top left image in the above figure corresponds to a scenario in which the vehicle must navigate around a set of moving boxes (in white). The top right image corresponds to the static obstacle evaluation, and the bottom image is a three vehicle race.

VI.6 Evaluation

VI.6.1 Experimental Setup

In this section, we present an analysis of our approach using the F1/10 simulation platform. Our evaluation includes a diverse set of experiments that include changing the number of racing agents present within the

racetrack, including additional dynamic obstacles within the racetrack, adding static obstacles onto the racetrack, and changing the racing environment. We compare the performance of our approach against a set of controllers typically utilized within the F1/10 racing competitions with respect to two metrics that we refer to as *efficiency*, and *safety*. *Efficiency* is the total distance that the F1/10 vehicle traverses around the track divided by the amount of time it took to do so.[3] *Safety* corresponds to the controller's ability to avoid collisions over a set of experimental runs (i.e. 10 collisions in 20 experiments corresponds to a safety score of 50%).

VI.6.1.1 Controllers

The following controllers were utilised as a local planning mechanism for selecting the target point used in our MPC regime. Additionally, we utilised them as a baseline comparison for our approach.

VI.6.1.2 Pure Pursuit

The Pure Pursuit algorithm is a widely used path-tracking algorithm that was originally designed to calculate the arc needed to get a robot back onto a path [263]. It has shown great success in being used in numerous contexts, and in this work we utilize it to design a controller that allows the F1/10 vehicle to follow a path along the center of the racetrack.

VI.6.1.3 Gap Following

Obstacle avoidance is an essential component of a successful autonomous racing strategy. Gap following approaches have shown great promise in dealing with dynamic and static obstacles. They are based on the construction of a gap array around the vehicle used for calculating the best heading angle needed to move the vehicle into the center of the maximum gap [2]. In this work, we utilize a gap following controller called the "disparity extender" by Otterness et al. that won the F1/10 competition in April of 2019 [230].

VI.6.2 Computing Border Constraints without Optimization

In [279] Liniger et al. tackled the autonomous racing problem via a nonlinear MPC problem that encoded the obstacle avoidance problem by means of a high-level corridor planner based on dynamic programming. The corridor that their framework utilized was constructed by projecting the points along the center line of the track onto the racetrack borders (one for the left border, and one for the right border), as shown in Figure VI.4. Their regime demonstrated success in controlling 1/43 scale race cars, driven at speeds of more than 3 m/s using controllers executing at 50 Hz sampling rate on embedded computing platforms [279]. While their evaluation was limited to environments with static obstacles, we experimented with using such a scheme to

[3]This equivalent to the average speed attained during the experiment.

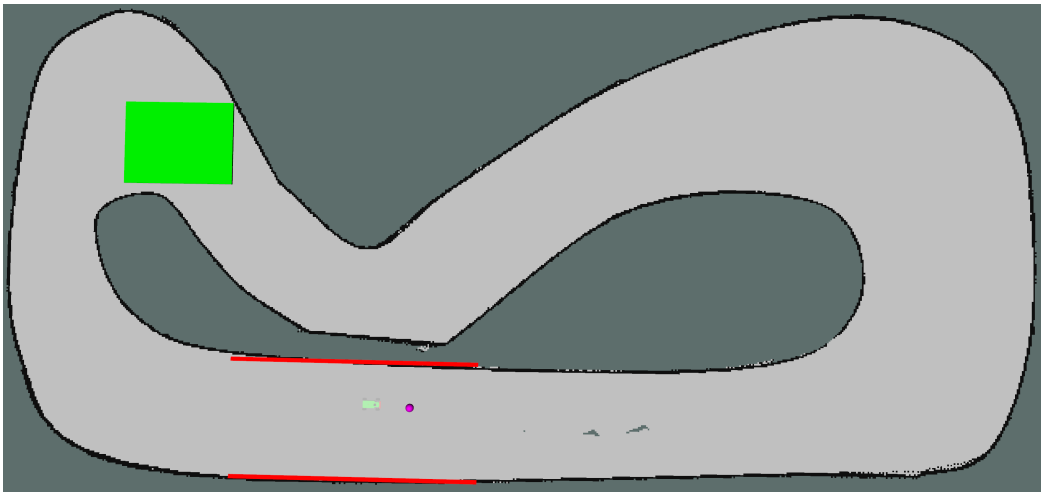

Figure VI.4: An example of a two-agent racing scenario. The bright green rectangle, represents the reachable set (convex hull) of the opponent vehicle over a $t = 0.5$ second time horizon, while the faded green vehicle represents the ego vehicle. The purple dot corresponds to the target location obtained from the local planner. The red lines are the two parallel half spaces that approximate the traversable region within the racetrack.

obtain the half-spaces framing our MPC problem. We refer readers to the following paper for an in-depth discussion of their approach [279].

VI.6.3 Runtime Analysis of Deriving Coupled Hyperplanes

Deploying optimisation-based methods into the real-time autonomous control systems requires careful considerations of timing constraints issued by the optimisation method. As we discussed previously the computation time of separating coupled hyperplanes can depend: number of obstacles points which are being considered, optimisation function and number of hyperplanes to be produced. In the following paragraphs, we describe an offline evaluation of the separating coupled hyperplane computation.

In the evaluation we considered three types of objectives functions: Hausdorff and Euclidean distances, which were computed between a hyperplane $\mathcal{H}_n$ and its associated subset of $\mathcal{Q}_{ob}$, and a simple object function (satisfiability problem) which only requires satisfying optimisation constraints. For each objective function the number of obstacle points were varied from 608 to 62, and either 2 or 4 hyperplanes were generated. The evaluation results are visually summarised in Figure VI.5.

VI.6.4 Experimental Results

Our evaluation included a sizeable diversity of experiments with respect to the number of vehicles present in the racing environment, the presence of static and dynamic obstacles, the racetrack used for the autonomous race, the local planner chosen to select goal points, and the method selected to obtain the separating hyper-

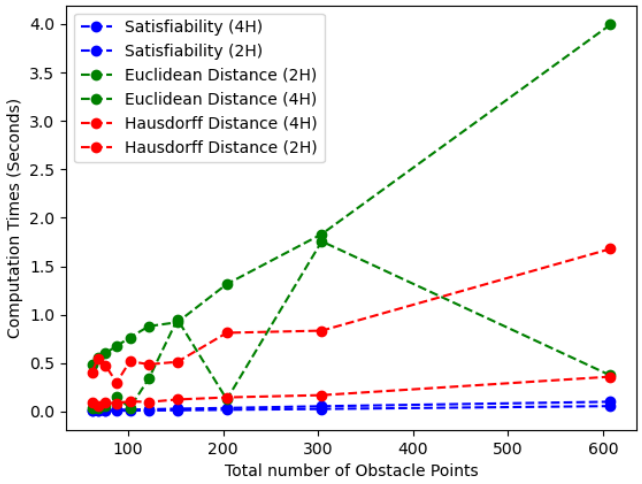

Figure VI.5: Offline evaluation of separating coupled hyperplane computation time against different numbers of obstacle points (optimisation constraints), different objective functions and number of hyperplanes. $\mathcal{X}_{safe}$ area between hyperplanes (only when two hyperplanes were computed) for each objective function: Satisfiability - 2.44428, Hausdorff - 10.4077 and Euclidean - 10.384.

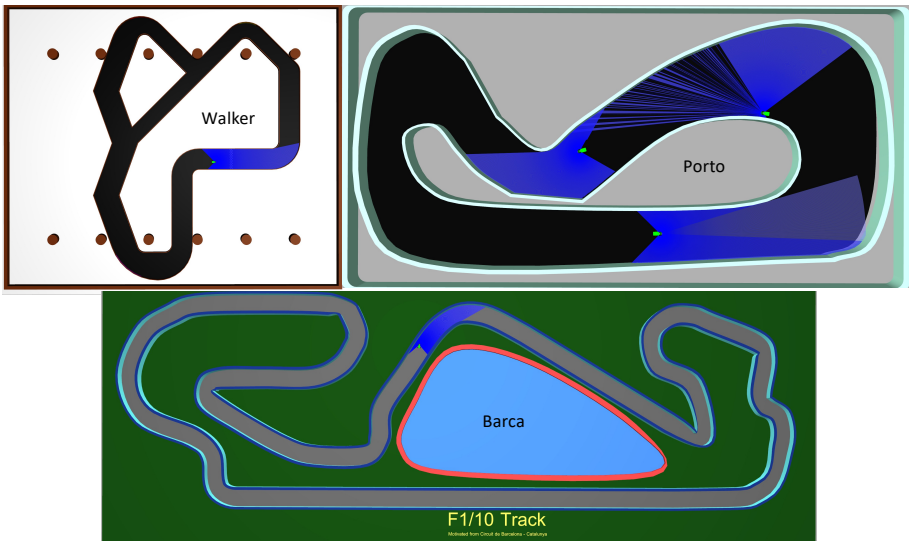

Figure VI.6: The different tracks we used in evaluating our approach. In the above figure: the bright green rectangle represents the simulated vehicles, and the blue region around each vehicle represents the full set of range values collected by the LiDAR sensor.

planes.Each configuration was evaluated over 30 experimental runs of 60 seconds. Table VI.1 and Table VI.2, Table VI.3 display the results of these experiments. In the tables that follow, DE corresponds to the disparity extender, PP corresponds to pure pursuit, MPCC corresponds to the approach presented by Liniger et al. [279], and MPC Hype corresponds to the optimization based approach presented in this document. Finally, Race Duration corresponds to the amount of time the agents were able to race before a collision occurred.

Table VI.1: Performance on Two Car Experiments Without obstacles

Track	Approach	Local Planner	Ego Efficiency (m/s)	Opponent Efficiency (m/s)	Race Duration (s)	Safety (%)
Barca	DE	DE	5.14	4.48	7.66	0.00
Barca	MPC Hype	DE	0.00	5.85	5.47	0.00
Barca	MPC Hype	PP	0.06	5.74	5.49	0.00
Barca	MPCC	DE	3.18	3.01	11.26	0.00
Barca	MPCC	PP	3.01	3.09	10.90	3.33
Barca	PP	PP	5.25	4.73	7.36	0.00
Porto	DE	DE	5.29	4.65	51.57	38.33
Porto	MPC Hype	DE	0.00	5.27	5.53	0.00
Porto	MPC Hype	PP	3.06	5.18	25.74	13.33
Porto	MPCC	DE	3.00	4.97	7.12	20.00
Porto	MPCC	PP	3.00	5.34	55.14	46.67
Porto	PP	PP	4.70	5.33	60.0	100.00
Walker	DE	DE	4.50	4.98	6.98	25.00
Walker	MPC Hype	DE	0.06	4.72	5.51	3.33
Walker	MPC Hype	PP	1.34	4.71	5.54	0.00
Walker	MPCC	DE	3.24	5.10	38.94	10.00
Walker	MPCC	PP	4.10	5.14	14.86	10.00
Walker	PP	PP	6.34	5.14	60.0	100.00

Table VI.2: Performance on three car experiments without obstacles

Track	Approach	Local Planner	Ego Efficiency (m/s)	Opponent Efficiency (m/s)	Race Duration (s)	Safety (%)
Barca	DE	DE	5.05	4.59	6.07	0.00
Barca	MPC Hype	DE	0.69	5.11	5.37	0.00
Barca	MPC Hype	PP	0.09	4.64	5.36	0.00
Barca	MPCC	DE	3.30	2.70	10.60	0.00
Barca	MPCC	PP	3.37	3.10	9.97	0.00
Barca	PP	PP	5.41	3.95	7.14	0.00
Porto	DE	DE	5.38	4.10	33.78	28.33
Porto	MPC Hype	DE	1.19	4.50	5.40	0.00
Porto	MPC Hype	PP	2.75	2.96	43.26	30.00
Porto	MPCC	DE	1.66	4.23	5.39	3.33
Porto	MPCC	PP	1.83	4.00	5.37	16.67
Porto	PP	PP	4.70	3.73	57.30	70.00
Walker	DE	DE	4.68	5.23	5.44	1.67
Walker	MPC Hype	DE	0.99	5.27	5.36	0.00
Walker	MPC Hype	PP	2.15	5.27	5.37	0.00
Walker	MPCC	DE	3.29	3.27	54.99	50.00
Walker	MPCC	PP	4.64	3.54	20.78	13.33
Walker	PP	PP	6.35	5.21	57.31	88.33

VI.7 Conclusions and Future Work

This chapter presented an optimisation based approach for static and dynamical obstacle avoidance problem within an autonomous vehicle racing context. Our control regime leveraged online reachability analysis and sensor data to compute the maximal safe traversable region that an agent can traverse within the environment.

Table VI.3: Performance on Dynamic Obstacle Experiments

Track	Approach	Local Planner	Ego Efficiency (m/s)	Opponent Efficiency (m/s)	Race Duration (s)	Safety (%)
Porto	DE	DE	4.96	3.93	22.93	13.33
Porto	MPC Hype	DE	1.08	4.77	5.39	0.00
Porto	MPC Hype	PP	2.16	4.42	9.05	0.00
Porto	MPCC	DE	2.78	5.20	11.32	0.00
Porto	MPCC	PP	1.32	4.61	5.40	0.00
Porto	PP	PP	4.45	4.83	15.42	1.67

Table VI.4: Performance on Static Obstacle Experiments

Track	Approach	Local Planner	Ego Efficiency (m/s)	Opponent Efficiency (m/s)	Race Duration (s)	Safety (%)
Porto	DE	DE	0.73	1.76	5.47	0.00
Porto	MPC Hype	DE	0.92	2.13	5.57	0.00
Porto	MPC Hype	PP	2.21	0.55	12.27	0.00
Porto	MPCC	DE	1.24	1.46	7.66	6.67
Porto	MPCC	PP	2.77	1.90	9.91	0.00
Porto	PP	PP	3.00	1.21	6.48	1.67

We described a technique for computing a convex safe region via a novel coupled separating hyperplane algorithm. This derived safe area was then used to formulate a nonlinear model-predictive control problem that sought to find an optimal and safe driving trajectory with varying degrees of efficacy. Our experimental evaluation demonstrated that our approach was feasible as an obstacle avoidance strategy. Finally, we assessed the runtime requirements of our proposed approach through an analysis of the effects of a set of varying optimisation objectives for generating these coupled-hyperplanes.

In future work, we wish to evaluate the proposed approach on the physical embedded F1/10 platform in order to further validate that our approach admits low resource requirements. Additionally, future studies would include an analysis against hierarchical control architectures that decompose the obstacle avoidance problem into planning and trajectory tracking. Finally, while the following study did not consider disturbances and uncertainty in optimisation problem, our future work will seek to assess these challenges rigorously.

CHAPTER VII

Challenges and Limitations

The following chapter presents a discussion of the challenges in designing learning-enabled autonomous systems that possess rigorous assurances of correctness with respect to formal mathematical specifications. As explored in the earlier chapters of this work, a natural solution to tackling this problem is the use of formal methods, where the major stakeholders involved in creating learned-enabled CPS utilize formal techniques for the design, specification, and verification of these systems [254]. While these approaches have demonstrated significant success in numerous contexts, it is imperative to explore the limitations and assumptions of these approaches from a practical as well as philosophical standpoint. Our discussion is divided into three sections. The first section presents a brief survey of open challenges in designing formal techniques for learning-enabled CPS. The second portion presents a brief discussion of challenges within the safety of these systems that are not addressed in this work specifically, but are important for the safety assurance problem at large. Finally, the last section discusses limitations around the assumptions and design choices made by the work presented in this dissertation. Additionally, we propose suggestions of how to overcome these challenges. It is our hope that this discussion will offer insights for those who wish to apply these techniques to other platforms and help promote further research aimed at achieving assured autonomy.

VII.1 Challenges Using Formal Methods

As previously discussed, the discipline of formal methods is at its core concerned with *proof*, that is with formulating specifications that shape proof obligations, creating systems that satisfy those obligations, and demonstrating via algorithmic search that these systems comply with their specifications [254]. Typically, this process is defined by three components. The first component is a model of the system to be verified, the second component is a model of the environment, and the third component is the specification/property being verified. Once these components have been defined, the output of the verification task is classically a yes or no report indicating whether the system satisfies the property in question. Typically, a failure case is also accompanied by a counterexample demonstrating the execution that causes the property to be violated [33]. Section II discusses several methods for solving these types of problems, and formal techniques have enjoyed wide success in numerous contexts.

The quality of the verification result (typically yes or no answer) is highly dependent on the quality of the underlying system model, environmental model, and specifications. The reality, however, is that deriving good representations of each of these elements is quite challenging, and these elements greatly impact the

difficulty and value of the verification task [254]. Therefore, it is imperative to evaluate the accuracy of the assumptions and accuracy of the major aspects of the verification problem [310]. Thus, while formal methods have demonstrated significant utility in numerous contexts, they are no ”magic bullet”, and come with their own set of limitations [311]. One only has to recognize that formal methods and tools are created by humans, in order to conclude that these approaches must also undergo rigorous inspection [312]. That is, the ”verifier must also be verified” [313]. In this section, we present a discussion of several challenges within the formal methods literature, in an effort to serve as a guide to this rapidly growing field, and promote research allowing formal methods to be utilized in more contexts.

VII.1.1 Generating Meaningful Formal Specifications of Correct Behavior.

Verification is only meaningful when high-quality formal specifications are used to describe the system's desired behaviour. As Seshia et al. note, amidst the growing interest in the use of verification methods, there has been surprisingly little work written about formal specifications, particularly within the context of learning-enabled systems [254]. One of the challenges within this realm is that it is often very hard, if not impossible, to write down a formal specification for many tasks. If we consider the case of a neural network trained to mimic a perception task, it is prohibitively difficult to formally specify the nuances of human perception. However, if the network is being used as a component in a larger system targeting a particular use case, then one can avoid having to describe the perception task, by issuing system-level specifications. For example, ”the autonomous vehicle maintains a safe distance from obstacles within its environment while it is in motion” [254]. However, this complicates the verification process in that one is now required to analyze a larger system and loses the benefits that come from decomposing a large verification task into a problem of reasoning about the correctness of smaller subsystems [254].

In a similar vein, traditionally formal verification approaches have treated the verification problem as a boolean problem where the satisfaction of a desired property is true or false [254]. However, for many learning-enabled autonomous CPS, it is common for the performance of a system to be evaluated using an objective function. Consider, for example, an unmanned underwater vehicle (UUV) tasked with carrying out a pipe inspection task. The UUV system must carry out the inspection while avoiding collisions with the pipeline, the seafloor, and all other potential obstacles. To successfully carry out this task, the UUV system must be able to explore the trade-offs between safety and its mission objectives. In this context, there may be an associated reward associated with progress in the pipe inspection task, and a cost associated with navigating close to obstacles within its environment. In this case, a boolean specification is inadequate, requiring the development of new approaches that can reason about correctness from both a quantitative and boolean perspective [254]. Seshia et al. discuss these challenges and potential opportunities for further

research in more detail in their survey of the formal specification literature for deep neural networks [33].

VII.1.2 Challenges in System and Environmental Modelling

One of the chief challenges in designing autonomous CPS is rigorously modelling the underlying system and its environment [314]. These systems are defined by interactions between a large set of heterogeneous components that bridge the physical, networking, and cyber realms [315]. Thus, modelling these systems requires the convergence of several disciplines of computer science that each possess a diverse set of modelling languages and frameworks. Consequently, significant challenges lie at the intersection of these realms.

Beyond system modelling, autonomous CPS operate in very complex environments with considerable uncertainty. One must only imagine an urban traffic setting to conceptualize the nature of complexity that an autonomous system must be able to deal with [315]. Still, the quality of the verification result is highly dependent on the fidelity of the environmental models in approximating reality. Therefore, sufficient care must be paid in capturing the important aspects of the environment while neglecting unnecessary details.

VII.1.2.1 Environmental Modelling

Autonomous systems often operate in loosely structured environments. In these contexts their understanding of the world is mediated by the particular sensor configuration chosen by the system designers, and a set of assumptions with respect to the accuracy of its sensors, its motor performance, and the major variables that define its interactions with the world [316]. The reality is that even in restricted scenarios, it may be impossible to define all the variables in the environment a priori [254]. As an example, modelling the interference between sensors is a deeply challenging task that is just one factor adding complexity to the overall modelling process. Furthermore, the safety of a system may depend on capturing the behaviour of a system in scenarios that are statistically unlikely to occur. Seshia et al. note that much of the modelling work for autonomous systems rely on distributional assumptions about the nature of the environment [254]. That is, the model of the environment is probabilistic in nature. Thus, in this case, the result of verification routines results in probabilistic guarantees that are sensitive to the parameters of the environment model. This is particularly true for environments that must model the behaviour of humans or other complex agents. Thus, environmental modelling demands rigorous explorations of uncertainty without rendering the verification process useless in practice [254]. There has been a large effort towards addressing these challenges in recent years, and we refer interested readers to the following surveys for an in-depth discussion of these approaches [316, 254].

VII.1.2.2 System Modelling

The close combination of heterogeneous physical, networking, and software interfaces in autonomous systems, yields systems with numerous interfaces and interrelations that require careful attention when reasoning about the behavior of the overall system [317]. Consider, for example, the design of an embedded controller that monitors the airflow over a wing's surface in an unmanned aerial vehicle (UAV) and modulates it through actuators to ensure that the vehicle is capable of extreme maneuvers [318]. Karsai et al. present this example, as an illustration of a design task that requires integrated design [318]. Specifically, they note that design decisions made in one aspect of the system's design, such as selecting a scheduling technique used in the embedded software, have acute consequences in the dynamic properties of the UAV [318]. Thus, the design of such systems can only be accomplished by taking a unified view and co-designing the physical aspects of the system with the computational components [318].

Modeling these complex systems as hybrid systems, which are a class of models that involve the interaction of a heterogeneous dynamics, has been an active area of research for decades and these frameworks have displayed great efficacy in modeling a wide range of systems [319]. While this area of research has developed powerful mathematical formalisms, the development of modeling languages that can be leveraged in the design of control systems that are close to the desired application domain still lag behind [318]. Most software tools leveraged by control engineers, such as Simulink/Stateflow [320], make use of modeling languages and formalisms that can be used to generate executable code that implements the desired system functionality. Hybrid modeling tools, however, are not readily available for many applications [318].

In general, hybrid modeling languages that seek to address this challenge, produce models that are often either unreasonable mathematically, or are too simplistic physically [319]. As an example, it is not uncommon for these modelling frameworks to neglect considerations such as the real-time scheduling behavior of an embedded processor, models of computation such as publish-subscribe, properties of the underlying operating system, properties of utilized middlewares, or even the performance of processing units [321, 318]. Thus, in recent years, several efforts have been devoted toward designing modeling frameworks that are rich enough to capture the most central aspects of the system design, without excessive mathematical formalisms or simplifications. An in-depth discussion of these efforts can be found in [319].

Finally, uncertainty is an inherent property of any CPS that must be considered throughout the design cycle [322]. However, it is a relatively unexplored research area. One of the central questions within this realm is how formal methods can be used to represent how the uncertainty imposed by the stochastic processes of the system and environment impact the overall system's behavior [323]. The resulting theories must be able to handle uncertainty at several layers of abstraction, and describe the interaction of multiple uncertainties in

a rigorous manner [324]. This assumes, however, that probability distributions for the relevant parameters of the system can be reliably specified or estimated [254]. Currently, the majority of approaches are too rigid to handle these types of problems, or are overly conservative, resulting in inconclusive results [325]. Within this context, Seshia et al. advocate for the extension of probabilistic verification approaches such as Markov Decision Processes and probabilistic programming in order to generate more robust probabilistic formal modeling frameworks that can adequately address these challenges [33].

VII.1.2.3 System and Model Integration

Apart from the aforementioned challenges, the integration problem for autonomous CPS is a deeply difficult problem [318]. The disparate processes used to build these systems as well as their inherent heterogeneity can often lead to unforeseen interactions. To prevent these unanticipated interactions from occurring requires a robust understanding of integration. In [318], Karsai et al. note that the integration problem can be viewed from two perspectives: model integration and system integration.

Model integration challenges typically occur when an individual seeks to perform an analysis of the entire CPS. To do so, the system designer must be able to combine several models of the various parts of the system that possibly utilize different layers of abstraction. They must also resolve any discrepancies in the execution semantics of the components (i.e. discrete-time vs continuous time models), and any discrepancies between the models utilized for design and those actually deployed on the real system [318]. Often it is the case that design modeling languages and analysis languages are different. Thus, sufficient care must be paid to ensure the agreement of models in order to trust the results of any assurance exercise [318].

Finally, system integration is where each of the essential design concerns (i.e environmental modelling, system modelling, design of the computational platform etc.) come together [326]. This is the most challenging aspect of autonomous CPS design, as any conflicts across design domains typically flare up at this stage. Sztipanovits et al. note that in many cases this problem is approached from a system management standpoint rather than from a well-designed science of integration [326]. These challenges are not unique to autonomous CPS, and Sztipanovits et al. note that the aerospace community as well as the automotive community expressed the need for such science in the early 2000s due to the rapidly growing complexity of managing the differences of components purchased from original equipment manufacturers [326]. In their work, Sztipanovits et al. advocated for a novel theory of composition to address the challenges of CPS integration, and we refer interested readers to their work for an in-depth discussion of these methods [326]. The paper additionally discussed several open problems in the area to promote further research.

VII.1.3 Scalability of Approaches

One of the major challenges plaguing many formal verification tools and techniques is scalability. While some methods have been able to handle large scale analyses of industrial applications [327], most of the existing tools are geared towards considering one or more relatively small subsystems within a bigger system [323]. The major difficulty in handling large systems, is that algorithmic decision procedures, such as model checking techniques, which often operate by performing an exhaustive exploration of the set of states that a system can attain over the course of its execution, suffer from the well known *state-explosion problem* [328].

The reality is that in modern systems, the space of possible behaviors that a system can enact is far too large for complete analysis. For instance, in some automotive vehicles (non-autonomous), it is now not unusual to have more than 100 million lines of code that govern the interaction of the power-train control, chassis control, active and passive safety systems, and other functionality [329]. Moreover, the exploration space for machine-learning based components such as image classifiers is prohibitively large. A single RGB image, of width W and height H, results in a space that contains $256^{W \times H \times 3}$ elements and in practice these images are typically fed to machine learning components as a stream [33]. Our current discussion neglects consideration of non-determinism, disturbances, and uncertainty, which only exacerbate the aforementioned challenges [330]. In recent years, however, significant progress has been made with respect to the size of systems that can be handled by formal techniques, and we refer interested readers to the following surveys for a discussion of these advances [114, 314, 331].

VII.1.4 High Learning Curve for Practitioners

Due to the sheer complexity of rigorous system design and the effective specification of system properties that need to be modeled and analyzed for a typical application, formal methods exercises typically require specialized engineers [323]. To apply many approaches effectively, one has to make the right assumptions and map component based analyses to system level properties that can effectively be reasoned about [325]. Thus, there is a steep learning curve for practitioners, who wish to use formal techniques and the process has generally proved to be a time-consuming process requiring considerable human-resources [325]. As an example, in recent years, the number of hybrid systems model checkers that have emerged has greatly increased. Regrettably, although many of these tools support roughly the same model semantics, their model description languages can differ dramatically. Thus, it is difficult to quickly evaluate using a particular tool for a desired task. Correspondingly, this makes using multiple tools prohibitively difficult [332]. Although tools like the Hybrid Systems Source Translation tool proposed by Bak et al. seek to address these source-to-source translation challenges, this limitation is also present for model checkers within the AI-verification literature as well [332, 9].

In a 2020 Survey of more than 130 high profile formal methods experts, as part of the 25th anniversary of the Formal Methods for Industrial Critical Systems Conference (FMICS), 63.8% of the experts stated that the steep learning curve of formal methods was a limiting factor of wider adoption of these techniques in industrial applications [333]. The majority of formal methods applications in industry are typically applied to problems where there is a clear return on investment (i.e. domains where requirements are well understood), or the cost of errant behavior is quite large [333]. One standard argument addressing this challenge is that while the cost of the use of formal methods may be high, this cost is amortized by the saving gained from better system quality [333]. However, the experts note that an investment in methods, tools, and practices aimed at lowering the barrier of entry for formal methods techniques and tools is urgently needed [333].

VII.2 Technical Challenges in Learning-Enabled Systems

Having discussed several challenges associated with the application of formal methods at large, in this section, we present a survey of several open challenges within the safety assurance literature of learning enabled systems that we deemed to be outside the scope of the work of this dissertation. These challenges, however, are deeply important to realizing safe autonomous systems. Since, the safety assurance problem for learning-enabled autonomous cyber-physical systems requires multi-layered and multidimensional approaches that can address the safety and functional correctness problem both at design time and operation time, the emerging literature within this realm contains a great diversity of techniques aimed at accomplishing this task.

VII.2.1 Dealing With Online-Learning Systems

Many assurance approaches are currently predicated on the assumption that once a system has been sufficiently analyzed and assured, the artifacts deployed within the underlying system will not evolve. However, for certain applications, such as systems that make use of reinforcement learning techniques, the system may evolve as it encounters new data and situations [254]. In these situations, Seshia et al. note, either verification approaches must be able to account for future changes in the operation of the overall system, or the verification must be performed online [254]. The former approach is inherently messy, however, because machine learning systems frequently mix input signals together, thereby entangling them and making isolated improvements to particular aspects of the model impossible [218]. Thus, it is extremely difficult to anticipate changes in the operation of a model based on online updates. This is a well known problem within the context of deep learning, and intuitively what this means is that slight changes in the distribution of the features provided to a particular neural network model may change all the relevant weights describing the importance of these features. This occurs whether the model is re-trained offline or modified online. Thus, machine learning models operate on a model known as the CACE principle: Changing Anything Changes

Everything, as described in [218]. While runtime assurance approaches are an effective tool in handling these types of scenarios, they are highly dependent on the assumptions made in terms of the types of monitors that are relevant for the safety task. Thus, there is still room for a more comprehensive theory of dealing with online adaptation in autonomous systems [254].

VII.2.2 Verifiable Training and Neural Network Repair

Numerous verification approaches address the problem of demonstrating that a neural network satisfies a particular property, and in the case of failure, generating a counter-example that is a manifestation of errant behavior [334]. However, a natural question arises of *How does one repair a network that has been proven to be faulty*? In recent years, several techniques have been proposed towards solving this problem [335, 334]. The main challenges within this realm are managing the computational complexity of the combined verification and training regimes, as well as maintaining the integrity of the original network [334]. Re-training inherently changes the structure of the underlying network, and sufficient care must be taken to prevent unwanted behaviors from arising [334]. To mitigate the computational cost of these approaches, researchers have proposed pruning techniques that aim to replace large neural networks with smaller versions more amenable to verification [336]. Furthermore, these methods seek to derive networks that admit comparable performance and robustness. However, in practice this a difficult problem. While significant progress has been made in recent years within the verifiable training and network repair literature, dealing with state-of-the-art networks still remains out of reach [334, 337]. Thus, there is an urgent need for verifiable training and network-repair algorithms that can be applied to real systems. We refer interested readers to the following papers for an in-depth discussion of these techniques [334, 338].

VII.2.3 Data Generation

In the same spirit as verified training and network repair regimes, several approaches have emerged over the last several years aimed at improving the quality of data that machine learning applications are synthesized from. This stems from a recognition that data is the fundamental starting point for all machine learning tasks [254]. The idea behind these approaches is to develop techniques that allow for the systematic generation of counter-examples and synthetic test data that satisfy key properties relevant to system designers [339]. The key challenge in this realm is generating meaningful data that can adequately describe complex environments in a semantically realistic manner. More specifically, the aim is to allow the system designer to construct distributions over scenarios of interest, and sample from these distributions to obtain concrete inputs for training and testing [339].

As an example, within an autonomous vehicle context, one may desire to sample from scenarios such as

high-traffic situations, or adverse weather conditions [339]. In recent years, there has been a large amount of work on generating synthetic data for specific applications [339, 340, 341, 342]. One such example is the SCENIC probabilistic programming language that can be used to generate data aimed at improving the performance of perception models in autonomous cars and robots [339]. While approaches such as these have displayed great promise, one of the challenges is addressing discrepancies between synthetically generated data and the real world [340]. Thus, the development of more comprehensive data generation techniques addressing challenges around the synthetic-reality gap, remains an open challenge [254].

VII.2.4 Handling Distributional Shifts

As mentioned previously, designing learning-enabled systems requires one to make assumptions about the environment that these systems will operate in. Bearing these assumptions in mind, the relevant data needed to train and evaluate models can be carefully curated [343]. The reality, however, is that the data sets used to construct these models are necessarily incomplete, and occasionally the system will find itself in a situation that it has not been adequately been prepared to deal with [344]. That is, the distributional assumptions made by the underlying data sets used to training learning-based components are no longer valid. As an example, one can image a perception neural network that has only been trained in clear weather conditions, having to deal with adverse weather conditions. These distribution shifts can lead to errant and unexpected behavior in many machine learning systems, and many of these models will perform poorly while reporting a high level of confidence in their predictions [344].

To address this challenge, several approaches for detecting and mitigating these distributional shifts have been proposed [345, 346]. Out-of-distribution detection (OOD) has emerged as one promising method of improving the overall safety of such systems. However, the challenge in this domain is designing methods that are robust but limit the number of false alarms produced by the distribution monitor [345]. Beyond OOD methods, several other approaches such as transfer learning, statistical hypothesis testing, conformal prediction, and self-aware learning have also been proposed [344]. One key question that must be answered within this realm is what to do when a distributional shift has been observed. In many contexts, this a deeply challenging question, that must be dealt with rigorously. We refer readers to the following paper for an in-depth consideration of these challenges [344].

VII.2.5 Negative Side Effects and Reward Hacking

One of the major challenges in designing autonomous systems is avoiding negative side effects from design decisions intended to accomplish a particular objective or task [347, 344]. This is an issue that plagues many engineering disciplines, but is particularly challenging for learning-enabled systems due to the difficulty of

specifying correct behavior in the environments that these systems typically operate in [344]. As an example, Amodei et al. provide the example of designing a robot that must move a box from one side of the room to the other. In this case, some of the most efficient ways that this can be done involves selecting a plan that might be destructive to the environment [344]. While it is possible to penalize these types of actions, it may not be possible to penalize every possible negative action that a system can select. This is particularly relevant within reinforcement learning approaches [344]. Many systems operate in large, multifaceted environments, thus it is crucial to develop techniques that can reason about the negative side effects of poorly specified objective functions.

An analogous problem within this space is reward hacking. Reward hacking refers to the phenomenon of agents gaming formal reward and objective functions to obtain solutions that may be valid in a real sense, but do not capture the designer's original intent [344]. As an example, one can imagine an autonomous robot designed to clean up oil spills that is rewarded on the basis of the total amount of oil it collects. In this scenario, the robot may intentionally create additional spills in order to earn a higher reward [344]. These problems are classically considered within the reinforcement literature, and there has been significant research proposed towards addressing this problem [344, 347]. However, they also appear in systems that make use of genetic algorithms, and it has been shown that genetic algorithms are capable of creating undesirable but formally correct solutions in numerous contexts [344]. In general, reward hacking, has been shown to be a challenging and general problem, and we refer interested readers to the following papers [344, 347] for a detailed discussion of this interesting problem.

VII.2.6 Compositional Design

Compositional design approaches have displayed great efficacy in enabling scalable verification of learning enabled autonomous systems [348]. Typically, the verification of an entire system is intractable, and the basis of compositional based approaches is deconstructing the system into a set of interacting components whose operation is defined by formally defined contracts [348]. These contracts define properties that are guaranteed to be held by each component, as well as the assumptions under which they are valid [348]. In doing so, the verification task is reduced to a more manageable analysis of individual components under a set of defined assume-guarantee contracts [348].

While theories of compositional design have been successfully developed for embedded systems, traditional software systems, and digital circuits, a comprehensive theory of compositional design for learning enabled systems has not sufficiently been developed [254]. As an example, Seshia et al. note that we do not have effective methods to answer questions such as "if two machine learning models are used for perception on two different types of sensor data (e.g. LiDAR and visual images), and individually satisfy their

specifications under certain assumptions, under what conditions can they used together to improve the overall reliability of the overall system," [254]. Thus, a systematic understanding of how this can be achieved is urgently needed. In recent years, several approaches for compositional verification of learning-enabled systems have been proposed, and we refer readers to the following papers for a survey of these techniques [348, 349, 350].

VII.2.7 Additional Considerations

To conclude this section, we also discuss several other considerations that are important when reasoning about the correctness of autonomous learning-enabled cyber-physical systems. Addressing security considerations around autonomous CPS is an active area of research. Due to the heterogeneous nature of CPS components, they are especially vulnerable to attacks, which can result in serious harm in safety critical contexts [351]. Thus, theories of CPS security from a multitude of perspectives are necessary for maintaining trust in the operation of these systems. Furthermore, methods aimed at improving the explainability of machine learning models within autonomous systems are also urgently needed in order to improve the transparency of system designs. These methods would also assist in addressing errant behavior that will necessarily occur when these systems are deployed in their operational design domains [344]. Finally, ethical issues such as fairness, and preventing abuse of these systems will also be critical in ensuring that the long term benefits of these systems can effectively be ensured. Moreover, we must ensure that these systems are not used to exacerbate inequality and contribute to the further marginalization of marginalized communities. We recognize that there are numerous other considerations that are also deeply critical within this arena, and we refer readers to the work of Amodei et al. which covers these considerations in more detail [344].

VII.3 Limits of Design Choices and Assumptions

We conclude this chapter, with an analysis of the key assumptions enabling much of the work presented in this dissertation, and provide a discussion of the limitations around the various design choices made herein.

VII.3.1 Forward vs Backward Reachability

Much of the work in this dissertation makes use of reachability analysis in order to compute the set of states that a system will assume over a finite time horizon. One can then use this set to obtain trajectories that are free from collisions. The specific form of reachability analysis that we utilized is commonly referred to as *forward reachability analysis*, and in this regime the reachable set is computed forward in time, starting from a particular initial set [241]. One of the main criticisms of forward reachability, as outlined in Chapter V is that it is only practical for short time horizons. For longer time horizons, the risk of limiting the performance of

the underlying system to overly conservative behaviors increases significantly [241]. Additionally, consider the case where an autonomous vehicle aims to navigate a scenario where it must avoid collisions with another dynamic agent. In this context, the autonomous vehicle computes the reachable set of the dynamic agent, and seeks to avoid this set in to ensure safety. Leung et al. demonstrate that this is an inherently open-loop mentality, since in considering which actions to take the autonomous vehicle does not incorporate how its future observations of where the dynamic agent may go, may affect the type of maneuvers it actually needs to take. More intuitively, forward reachability schemes do not include closed-loop feedback of the behavior of the opposing system [241].

The alternative to forward reachability approaches are backward reachability schemes. One popular formulation of backward reachability techniques are Hamilton-Jacobi (HJ) reachability methods. In backward reachability frameworks, we begin with a target set of undesirable states, and compute the set of initial states from which the system can reach the set of undesirable states [352]. The backward reachable set, therefore, represents the set of initial states from which there does not exist a controller that can prevent the system from entering the undesirable states [241] over a finite time horizon. In this case, the backward reachable set is therefore treated as the set of initial states to be avoided [241]. There are two distinguishing features of these approaches. The first is that the backward reachable set is computed backwards in time, and the second feature is that it is often computed assuming closed-loop reactions of the joint dynamics of the system and another agent within its environment. Thus, in practice, one can create regimes where a system can react to the behaviors of another dynamic agent for any time and state configuration within the finite time horizon used in the reachability analysis. While this often leads to less conservative behaviors for the overall system, techniques such as HJ reachability are quite computationally challenging [241].

In many cases, the results of backward reachable set computations are cached via a look-up table, so that they can be utilized at runtime in a near-instant lookup time fashion to obtain the optimal control policy that ensures safety [241]. To the best of our knowledge, we are not aware of any backward reachability schemes that possess rigorous real-time guarantees. Still, many approaches have achieved impressive performance in numerous contexts [241, 95, 96]. To limit the conservative nature of several of the safety architectures presented in this work, backward reachability techniques could have been used. One of the limitations of this work, is we did not evaluate such an approach in our experiments. In future work, we hope to perform such an analysis, and present an in-depth analysis of the tradeoffs between each reachability scheme.

VII.3.2 Use of Robotic Middlewares

The safety architectures presented in this work are constructed using the standard robotic paradigm of a set of communicating components and nodes. Our work made use of the Robotics Operating System (ROS),

but several other middlewares such as OPRos [353], OpenRTM [354], and Orcos [355] are also frequently utilized within this arena [316]. One of the core assumptions in many systems is that the middleware is sound. However, as Luckcuck et al. appropriately point out, given the diversity of systems that can be produced using these middlewares, and the complexity of their parametrization, demonstrating that these systems are engineering correctly is a challenging task [316]. One of our assumptions, in this work, is that the middleware is sound. However, in future work, it may be instructive to explicitly model and analyze the behavior of the middleware leveraged within the system architecture.

VII.3.3 Architectural Considerations

Machine learning components within autonomous applications are typically used in perception tasks to enable the system to perceive its surroundings. These tasks include the detection and classification of objects, handling sensor data (i.e. sensor fusion), scene understanding, behavior prediction, and localization [356]. While, there has been significant research in using machine learning for control tasks [357], most state-of-the-art systems utilize standard control systems for operation, and decompose the system into a set of sub-components where each of the machine learning sub-systems provide information to a manager that handles the behavior of the overall system [356]. In our work, to ease the verification task, we primarily considered scenarios, where the machine learning component was utilized for control tasks. In future work, our aim is to analyze more complex systems that are closer to real-world systems, and consider compositional based verification regimes.

VII.4 Dealing With Uncertainty

Autonomous CPS must be able to deal with uncertainty in almost all facets of their interaction with their environments [358]. The sources of uncertainty include sensor errors such as inaccurate measurements, actuator effects such as the degree of wheel slip in different terrains, and other environmental effects such as the strength of undersea currents or wind disturbances [358]. Models of autonomous CPS must be able to explicitly capture these types of uncertainties, when reasoning about the functional correctness of systems. Furthermore, they must be able to handle non-deterministic models of uncertainty [358].

In this work, we modeled the uncertainty around the location of our vehicle models utilizing intervals around the real position, velocity and orientation of the vehicle. Additionally, we captured the uncertainty around the specific parameters of the mathematical models of our system by using intervals around the constants defining our models. Thus, in this context, the ordinary differential equations describing the evolution of our vehicle models, were transformed into differential inclusions. Regrettably, these intervals were constructed without any assumptions about the underlying distributions governing these intervals. However,

to be meaningful, distributional assumptions about uncertainty must be obtained. Several formalisms exist within the CPS literature exist that support estimating these distributions, such as the paradigm of probabilistic programming [254]. It is worth noting that these distributional estimates are not precise measurements, and obtaining these estimates is difficult in practice. While we judged these analyses, to be out of the scope of this dissertation, comprehensive regimes for dealing with uncertainty are crucial in reasoning about the correctness of autonomous CPS [358].

CHAPTER VIII

Conclusions

There are few technologies that hold as much promise as autonomous CPS in re-orienting the way we move around, explore new environments, distribute resources, and conduct complex missions. To bring forth these benefits, we must ensure that CPS meet rigorous standards of correctness both at design time and during operation. This mandates the development of modeling tools and algorithms that can deal with the complexity exhibited by CPS and their environments. Since CPS operate in unstructured and dynamic environments, their design often incorporates opaque learning-enabled or machine learning components. Thus, assurance tools must be able to provide guarantees for these methods as well. While numerous works have been proposed in the past few years for the analysis of CPS, the vast majority of these efforts have not been able to scale to the complexity found in real world applications. Thus, designing solutions that are both practical and rigorous is extremely challenging.

Bearing the above in mind, we began this work with a case study investigating the use of two leading machine learning methods, Reinforcement Learning and Imitation Learning, in synthesizing neural network controllers for an autonomous racing task. In this study, we focused on the performance and safety of the derived controllers when they were deployed in novel contexts. The results of this study provided an enlightening analysis of the tradeoffs of the two machine learning paradigms and provided a persuasive motivation for the need to monitor learning enabled components at runtime.

Some of the most plausibly effective methods of demonstrating that autonomous CPS satisfy relevant safety specifications is through the use of safety architectures, runtime monitoring, runtime verification and runtime assurance [9]. Motivated by these techniques, and the need to provide correctness guarantees for machine learning methods at runtime, we formulated a runtime assurance regime leveraging reachability analysis and the simplex architecture that can handle components, such as machine learning models, that may too large or too complex to formally assure. The switching logic characterizing our simplex architecture was based on a real-time reachability regime for dynamical systems that allowed us to abstract away the underlying nature of the controllers governing the behavior of the system and rather analyze the effects of the controller's decisions on the system's future states. We showed how this architecture could be used to ensure the safety of a 1/10 scale autonomous vehicle that made use of a diverse set of machine learning controllers as it navigated an environment with static obstacles. Our experiments were performed in simulation as well as an embedded hardware platform. The latter configuration validates our claims that our safety regime admits low resource requirements.

Building on the aforementioned work, we extended our runtime assurance framework to handle dynamic obstacles and account for various types of uncertainty in our safety analyses. In this work, we saw the trade-off between the performance of the system and the effects of uncertainty on the conservativness of our safety regime. Additionally, we demonstrated how reachability analysis can be leveraged to efficiently describe the evolution of a system for a large set of bounded initial conditions, disturbances, and variations in the parameters values governing the underlying system model. Reachability analysis allows for these analyses to be done with a completeness or coverage that a finite number of simulations cannot deliver.

Having demonstrated how reachability analysis can be used as a robust analysis tool for ensuring safety, we then utilized this framework to develop an optimization based control framework for static and dynamic obstacle avoidance within an autonomous vehicle racing context. Our framework is based on the well-known model predictive control framework, and the central idea of this work is to compute the maximal safe traversable region that an agent can traverse within the environment via reachability analysis. We then solve an online optimization problem to synthesize control inputs that will allow the vehicle to traverse the racing environment as fast as possible. We demonstrated the efficacy of this approach through a large set of experiments with the F1/10 model.

Finally, we concluded this work with a discussion of the challenges in designing learning-enabled autonomous systems that possess rigorous assurances of correct behavior with respect to formal mathematical specifications. As explored in the earlier chapters of this work, a natural solution to tackling this problem is the use of formal methods, and while these approaches have demonstrated significant success in numerous contexts, there are limits to their effectiveness. We divided our discussion into three sections. The first section presented a brief survey of open challenges in designing formal techniques for learning-enabled CPS. The second portion presented a brief discussion of challenges in the design of learning-enabled systems that are not specifically addressed in this document. Finally, the last section discussed limitations around the assumptions and design choices made by this dissertation. It is our hope that this discussion will aid those who wish to apply these techniques to other platforms and help promote further research aimed at realizing assured autonomous systems.

In summary, we proposed a set of runtime verification techniques for the safety assurance of autonomous learning-enabled CPS. The technologies enabling our work are extensions of real-time reachability methods and the simplex architecture. In this document, we demonstrated that this framework can be used in a diverse set of dynamic and uncertain environments to ensure safety, and our work was validated using a vast set of experiments both in simulation and on an embedded hardware platform. This demonstrates the cross-platform nature of our techniques, as well as validating our claim that they admit minimal resource requirements. We recognize that there is still a great deal of work to be done in order to realize, safe, secure, and reliable

autonomous systems, and in an effort to promote further research we provide a description of all the artifacts enabling this work. Detailed instructions on how to reproduce this work can be found in the appendix of this document.

— “The truth is, most of us discover where we are headed when we arrive. At that time, we turn around and say, yes, this is obviously where I was going all along. It’s a good idea to try to enjoy the scenery on the detours, because you’ll probably take a few,” Bill Watterson

CHAPTER IX

List of Publications

1. Patrick Musau, Nathinel Hamilton, Diego Manzanas Lopez, Preston Robinette, and Taylor T. Johnson, "An Empirical Analysis of the Use of Real-Time Reachability for the Safety Assurance of Autonomous Vehicles," arXiv, 2022 May.

2. Nathaniel Hamilton, Patrick Musau, Diego Manzanas Lopez, Taylor Johnson, "Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning," In IEEE International Conference on Assured Autonomy (ICAA), 2022, March.

3. Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston Robinette, Taylor Johnson, "On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers," In IEEE International Conference on Assured Autonomy (ICAA), 2022, March.

4. Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T Johnson, "Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter," In Formal Aspects of Computing, Volume 33, Issue 4-5, pp. 519-545, 2021 August.

5. Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, Taylor T. Johnson, "Robustness Verification of Semantic Segmentation Neural Networks Using Relaxed Reachability," In 33rd International Conference (CAV), 2021 July.

6. Taylor T. Johnson, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, Chao Huang, "ARCH-COMP20 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants," In ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 74, pp. 107–139, 2020, September.

7. Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson, "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems," In 32nd International Conference on Computer-Aided Verification (CAV), 2020, July.

8. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, Hoang-Dung Tran, Taylor T. Johnson, "Case Study: Safety Verification of an Unmanned Underwater Vehicle," In 2020 IEEE Security and Privacy Workshops (SPW) 2020, May.

9. Hoang-Dung Tran, Diego Manzanas Lopez, Xiaodong Yang, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson, "Demo: The Neural Network Verification (NNV) Tool," In 2020 IEEE Workshop on Design Automation for CPS and IoT (DESTION) 2020, April.

10. Weiming Xiang, Diego Manzanas Lopez, Patrick Musau, Taylor T. Johnson, "Reachable Set Estimation and Verification for Neural Network Models of Nonlinear Dynamic Systems," In Safe, Autonomous and Intelligent Vehicles. Unmanned System Technologies (Huafeng Yu, Xin Li, Richard M. Murray, S. Ramesh, Claire J. Tomlin, eds.), Springer International Publishing, pp. 123–144, 2019.

11. Hoang-Dung Tran, Feiyang Cai, Diego Manzanas Lopez, Patrick Musau, Taylor T. Johnson and Xenofon Koutsoukos"Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control," In ACM Transactions on Embedded Computing Systems, Volume 18, Issue No. 5s, Article 105, 2019, October.

12. Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Taylor T. Johnson "Star-Based Reachability Analysis for Deep Neural Networks," In 23rd International Symposium on Formal Methods (FM'19) (, ed.), Springer International Publishing, 2019, October.

13. Hoang-Dung Tran, Luan Viet Nguyen, Patrick Musau, Weiming Xiang, Taylor T. Johnson, "Decentralized Real-Time Safety Verification for Distributed Cyber-Physical Systems," In Formal Techniques for Distributed Objects, Components, and Systems (FORTE'19) (Jorge A. Pérez, Nobuko Yoshida, eds.), Springer International Publishing, Cham, pp. 261–277, 2019, June.

14. Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Taylor T. Johnson, "Parallelizable Reachability Analysis Algorithms for Feed-forward Neural Networks," In Proceedings of the 7th International Workshop on Formal Methods in Software Engineering (FormaliSE'19), IEEE Press, Piscataway, NJ, USA, pp. 31–40, 2019, May.

15. Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, Taylor T. Johnson, "ARCH-COMP19 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants," In ARCH19.

6th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 61, pp. 103–119, 2019, April.

16. Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Taylor T. Johnson, "Verification of Closed-loop Systems with Neural Network Controllers," In ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (Goran Frehse, Matthias Althoff, eds.), EasyChair, vol. 61, pp. 201–210, 2019, April.

17. Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, Taylor T. Johnson "Verification for Machine Learning, Autonomy, and Neural Networks Survey," ArXiV, 2018, October.

18. Patrick Musau, Diego Manzanas Lopez, Hoang-Dung Tran, Taylor T. Johnson, "Linear Differential-Algebraic Equations (Benchmark Proposal)", In 5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Oxford, UK, 2018, July.

19. Patrick Musau, Taylor T. Johnson, "Continuous-Time Recurrent Neural Networks (CTRNNs) (Benchmark Proposal)", In 5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Oxford, UK, 2018, July.

Appendix A

Experimental Design and Source Code Repositiories

A large part of this work utilized an open-source robotics test-bed, known as the F1/10 platform, as an evaluation, experimentation, and demonstration platform for the safety assurance methods presented in this dissertation. This platform allowed for the testing of design-time and runtime assurance approaches within the context of robotic system software stacks. The F1/10 Autonomous Racing Platform was originally developed by the University of Pennsylvania[1] and has now become the centerpiece of bi-annual racing competitions, and numerous research projects. In an effort to reduce the barrier of entry for researchers who seek to reproduce this work, as well as promote further research, the following chapter outlines the details of setting up this platform for research in assured autonomous systems. Additionally, the source code used within our experiments is provided through Zenodo[2] and GitHub[3] [359].

A.1 F1/10 Autonomous Racing Platform

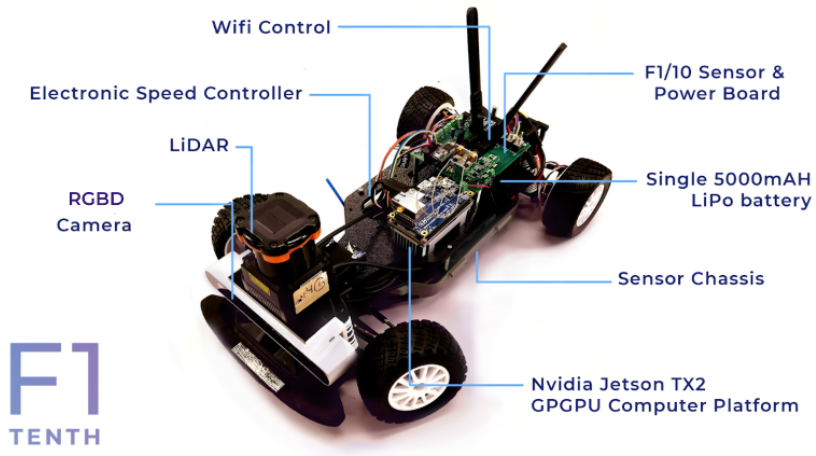

Figure A.1: F1/10 Hardware Platform. (Mangaraham 2020)

As previously outlined in this document, The F1/10 platform [174] is an open-source 1/10 scale remote controlled (RC) racecar platform that is a model of a Ford Fiesta ST rally car. It boasts a four-wheel drive

[1] http://f1tenth.org/

[2] Zenodo is an open-source platform that allows researchers to upload software and other artefacts that can be identified by a persistent digital object identifier.

[3] All the artifacts used to set up the experiments utilized within this dissertation can be found at the following repository: https://zenodo.org/record/5879646.

brushless powertrain capable of speeds of up to 44 MPH.[4]. In the F1/10 community, the RC car is often modified to include a standard suite of sensors such as stereo cameras, LiDAR (light detection and ranging), and inertial measurement units (IMU). These components are often controlled using either an NVIDIA Jetson TX2 or Jetson Xavier as the compute platform, and the software stack is built on the *Robot Operating System* (ROS). A build manual describing how to assemble the F1/10 platform is available online at **f1tenth.org**.

The F1/10 platform also possesses a Gazebo-based simulation environment [225] that includes a realistic model of the F1/10 platform and its sensor stack[5] [226]. Figures A.3 and A.4 display some simulation environments available within the simulation environment. The software architecture utilized within the hardware and simulation platforms are nearly identical, and we will outline any differences in the discussion that follows. In the following subsections, we outline the software packages used within our work. The majority of these packages are available in ROS, and a link to the source code for each utility is provided alongside each description.

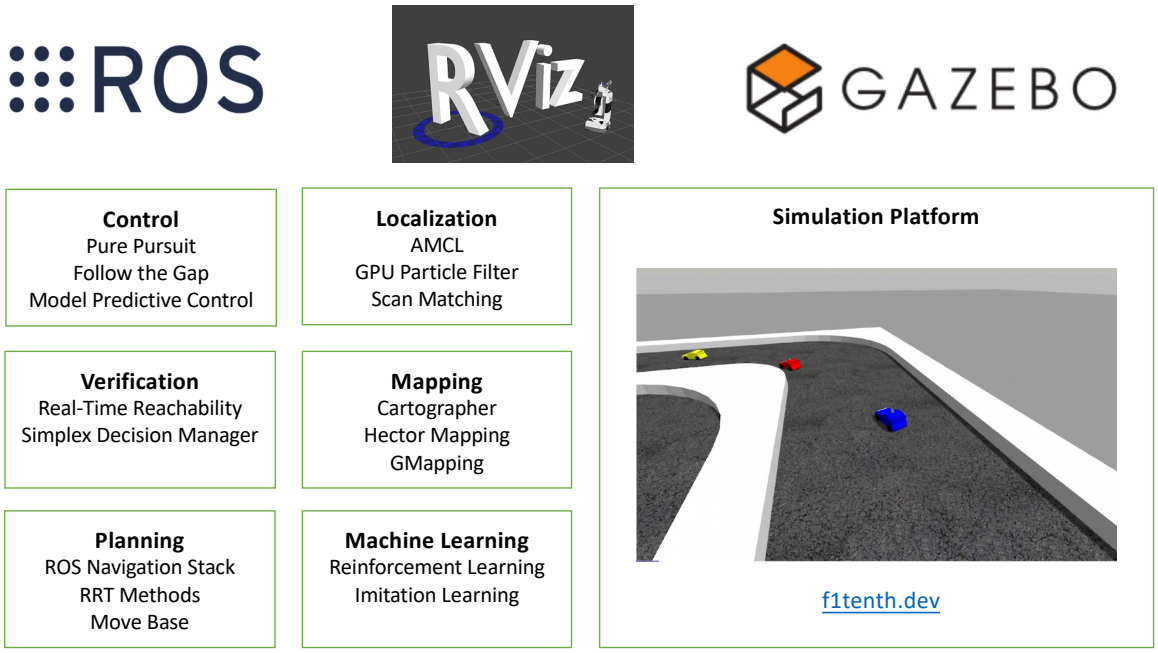

Figure A.2: Summary of packages used within the F1/10 Software Architecture.

There are four fundamental tasks in developing a successful autonomous platform: *perception*, *localization*, *planning*, and *control*. The vehicle must be able to interpret its sensor information, it must be able to determine its position within the environment; it must decide how to act to achieve its goals; and it must be able to synthesize motor outputs to carry out actions that satisfy its goal [360]. A summary of the packages

[4]https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=details

[5]Instructions on setting up the simulation platform can be found at the following link https://github.com/pmusau17/Platooning-F1Tenth.

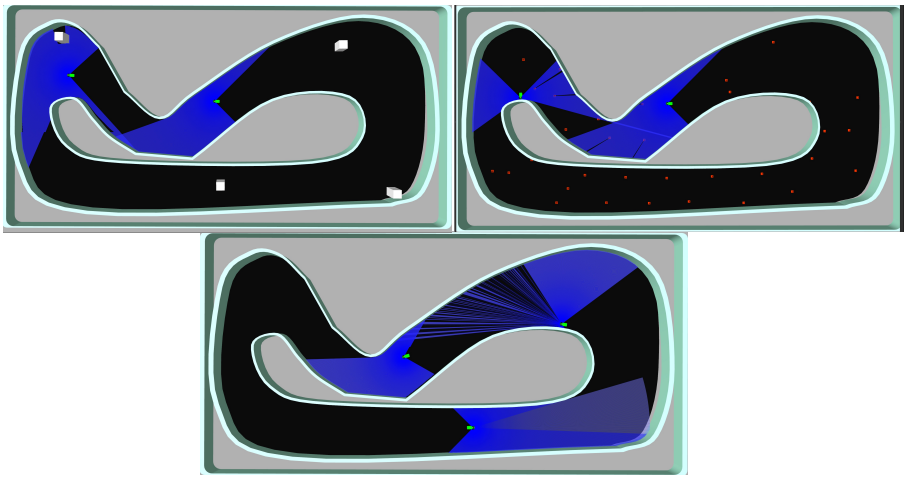

Figure A.3: Example of two and three vehicle racing environments. These environments may also include, the presence of static and dynamic obstacles. The top left image in the above figure corresponds to a scenario in which the vehicle must navigate around a set of moving boxes (in white). The top right image corresponds to a scenario with static obstacles, and the bottom image is a three vehicle race.

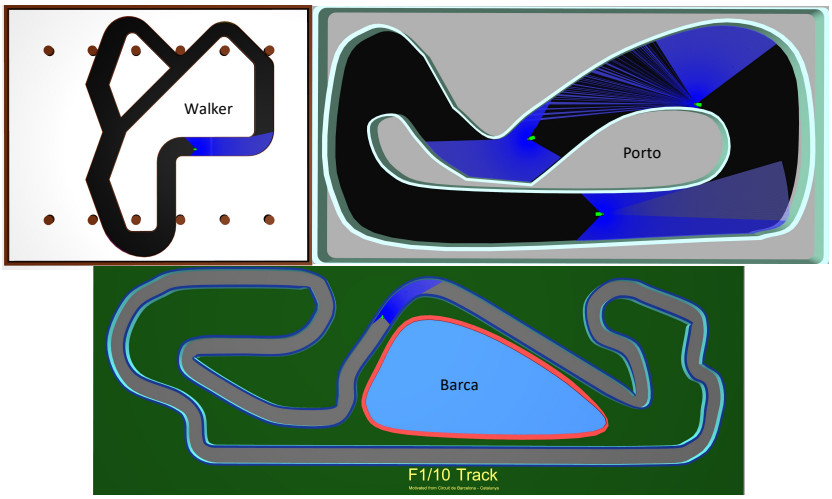

Figure A.4: Visualization of the various racing environments available within the F1/10 Simulation Platform. In the above figure: the bright green rectangle represents the simulated vehicle, and the blue region around each vehicle represents the full set of range values collected by the LiDAR sensor.

used to accomplish these goals within our experiments can be found in Figure A.2

A.1.1 Mapping

Building maps is one of the fundamental problems in creating intelligent mobile robots. These maps allow a robot to determine its position and orientation in a particular environment, as well as perform path planning routines. Frequently, these maps are represented as an *occupancy grid*, where the environment is discretized using a grid with a fixed resolution (i.e 1x1 meter cells). The value in each of these cells contains the probability that the cell is occupied. Thus, the task of many mapping packages is to use sensor data to construct such a grid. An example of such a grid is shown in Figure A.5. The following ROS packages are often used to construct maps of unknown environments with varying sensor requirements.

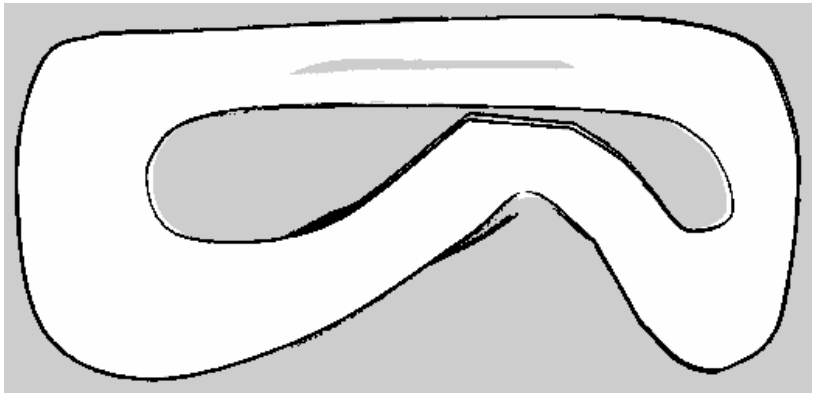

Figure A.5: Example of an occupancy grid produced by a mapping utility that can be leveraged online for planning and obstacle avoidance tasks. This particular occupancy grid was generated using the Cartographer package.

A.1.1.1 Hector Mapping

The Hector Mapping approach proposed by Kohlbrecher et al. utilizes a robust scan matching method to efficiently create an occupancy grid representing a robot's environment.[6] The scan matching approach can be augmented with odometry data to improve the accuracy of the occupancy grid produced by the approach. The approach admits low resource requirements and has been utilized on Unmanned Ground Robots, Handheld Mapping Devices, as well as Unmanned Aerial Vehicles [361].

A.1.1.2 GMapping

GMapping is a highly efficient mapping approach that utilizes laser range data to construct occupancy grids. The underlying technique makes use of Rao-Blackwellized particle filters [362]. Their approach uses an adaptive technique to limit the number of particles needed to construct a map as well as fuse information

[6] http://wiki.ros.org/hector_mapping

from a laser based sensor, odometry information, and a scan matching process. This method allows for smaller degree of uncertainty with respect to the robot's position and orientation and a more accurate map. There are a fair number of parameters that must be defined in utilizing this approach, and we refer readers to the following link for further detail.[7]

A.1.1.3 Cartographer

Cartographer[8] is a real-time mapping and localization package that has been used on a wide range of robotic platforms with varying sensor modalities [363]. It was designed to be able to operate in low-resource environments and has shown success in generating maps with resolutions as low as 5 cm [363]. The approach is based on a laser scan-matching approach that leverages sub-maps, loop closure detection, and graph optimization techniques. The majority of maps utilized within this dissertation were constructed using this package due to its extensive documentation and ease of use.

A.1.2 Localization

Localization is the process of determining the state (position and orientation) of a robot with respect to its environment. Typically, this involves utilizing a map of the environment as well as sensor information to observe the environment and monitor the motion of the robot. The robot localization task is a well-studied problem, and there is a wealth of literature and software tools aimed at tackling this problem [360]. The following surveys provide an excellent summary of these approaches [364, 365]. In many cases, if a robot possesses an appropriate sensor configuration, it can autonomously explore an environment, gain knowledge about it, and construct an appropriate map while simultaneously localizing itself relative to this map [360]. This latter problem is referred to as the *Simultaneous Localization and Mapping* (SLAM) problem, and has been lauded as arguably one of the most difficult problems in mobile robotics. In the sections, that follow, several ROS packages for robot localization are described, many of these approaches have the capability of being run as SLAM approaches, or solely localization packages that assume the presence of a pre-computed map.

A.1.2.1 Scan Matching

Scan matching is the task of registering successive laser scans to determine the relative positions from which the scans were obtained. It is one of the most popular localization tools for mobile robotics [366]. The laser scan matcher package[9], is a scan matching approach, based on Andrea Censi's Canonical Scan Matching

[7] http://wiki.ros.org/slam_gmapping
[8] https://google-cartographer.readthedocs.io/
[9] http://wiki.ros.org/laser_scan_matcher

approach, that leverages a variant of the iterative closest corresponding point surface matching problem to perform an incremental laser scan registration task [367]. Intuitively, it allows for the matching of consecutive laser scans to estimate the position of a laser sensor in a given reference frame. This can be done with or without additional odometry estimation. However, the accuracy of the approach improves when fused with other odometry inputs, such as the use of an IMU.

A.1.2.2 Adaptive Monte Carlo Localization (AMCL)

One of the most successful methods for state estimation tasks have been particle filter based approaches. Particle Filters, or sequential Monte Carlo Methods, are a class of numerical methods for the solution of optimal estimation problems for non-linear and non-Gaussian scenarios [368]. They differ from other standard approximations methods, such as the Extended Kalman Filter, in that they do not rely on linearization techniques or crude functional approximations [368]. The key challenge in implementing these approaches is managing the computational costs of representing posterior probability densities over the state space by large sets of samples [368]. One of the ways that this computational burden can be handled is by adapting the number of samples used to estimate the position of a robot on the fly. The AMCL[10] package leverages a sampling method based on the Kullback-Leibler distance in order to select the number of samples used in the particle filtering approach [369]. The idea is to use the KLD distance to measure the approximation error of the particle filter belief state. If the particles cover a large area of the state space, then the approach utilizes a large number of particles to update the belief state. However, if the particles cover a small area of the state space, then a smaller number of particles is utilized. The approach has been used on a wide range of robot platforms and has demonstrated great efficacy in efficiently tackling the robot localization task [369].

A.1.2.3 Ray-Casting Based Particle Filter Localization

As mentioned previously, particle filters are a popular class of Monte Carlo algorithms used for a variety of state estimation problems [3]. They are often used to track the pose of mobile robots by iteratively refining a set of pose hypotheses called particles. These particles are then updated using motion models, odometry data, and sensor information. For robots that make use of range sensors such as LiDAR or Sonar, ray-casting approaches can be used to compare sensor readings against the hypothesized pose and obstacles within the map. While effective, these approaches can be computationally expensive. In [3] Walsh et al. presented a novel data structure called the Compressed Directional Distance Transform which allows efficient ray casting. The method allows for GPU acceleration and displayed an order of magnitude speed increase against comparable ray casting particle filter methods. A visualization of the approach within a racing setting

[10] http://wiki.ros.org/amcl

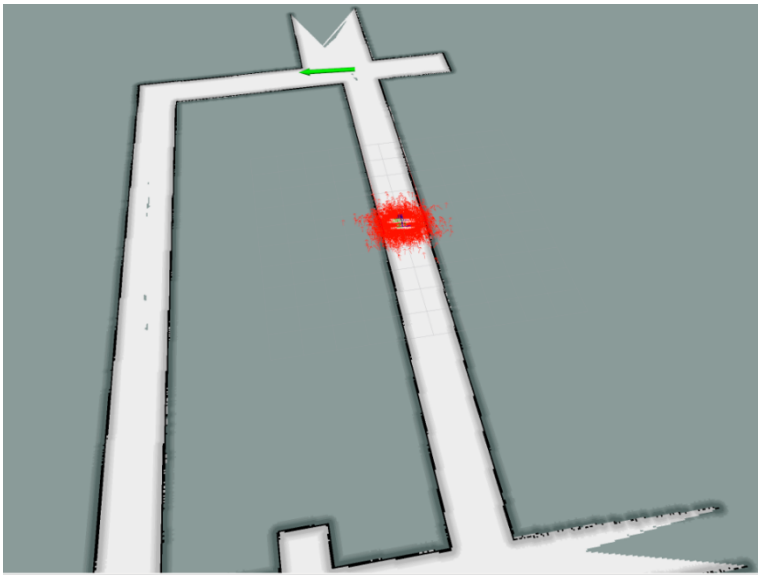

Figure A.6: GPU-based particle filtering for position and orientation estimation, developed by Walsh et al. [3]

is shown in Figure A.6.[11] We leveraged this package most frequently in our autonomous racing strategies.

A.1.2.4 Hector Slam

The Hector Slam package[12] is a SLAM package designed for unmanned surface vehicles that admits low resource requirements. The approach implemented in this package makes use of laser scans in a planar map, and measurements from inertial measurement units to localize the robot within a map. The approach can also be used to generate and update a map online. The interested reader can find more details on the approach in the following paper [361].

A.1.3 Planning

Motion planning is one of the most important problems in creating intelligent autonomous mobile robots. The central idea is to find an optimal or near-optimal path starting from an initial state to a selected target state that avoids obstacles. The optimality of this path depends on the metric used to evaluate performance (i.e. minimum energy path, the shortest path, the lowest risk), and numerous motion planning approaches have been developed over the years. A nice summary of these approaches can be found in the following survey [370] by Ferguson et al. The following subsections outline several packages that can be used to perform motion planning, as well as several specific algorithms that we leveraged in our software stack[13].

[11] https://github.com/mit-racecar/particle_filter

[12] http://wiki.ros.org/hector_slam

[13] The source code for planning utilities leveraged within our work can be found at the following linkhttps://github.com/pmusau17/Planning-and-MPC

A.1.3.1 Navigation Stack

The ROS Navigation stack is a collection of software packages aimed at solving the motion planning problem for differential drive and holonomic wheeled robots. These packages allow you to leverage sensor and odometry information and send velocity commands to a mobile robot. It has extensive documentation and set of tutorials that can be found at the following link: http://wiki.ros.org/navigation. Although the F1/10 is not a differential drive robot, several of the navigation utilities were useful in designing the F1/10 software architecture. These utilities included: sending goal commands to a behavioural planning layer, updating local costmaps based on sensor information, and performing global path planning[14].

A.1.3.2 Move Base

Many platforms leverage a hierarchical architecture to allow a robot to accomplish its goals, and classically high-level mission planning and motion planning can be seen as representing two different perspectives [371, 372]. Motion planning techniques are typically viewed from a bottom-up viewpoint, where the task is to generate a trajectory that guides the system from an initial state to a goal state. High-level mission planning, is a top-down mindset, where the focus is to generate a set of discrete actions that can be combined to achieve more complex tasks [373]. The Move Base package[15] in ROS provides an implementation for creating hierarchical planning architectures and combines local and global path planners to accomplish navigation tasks in an action based framework. It serves as an interface for configuring, running, and interacting with the navigation stack described in the previous section. The package is extensively documented and provides a large suite of tutorials for those interested in leveraging its capabilities.

A.1.3.3 The Open Motion Planning Library and Moveit

Beyond the navigation stack, The Open Motion Planning Library (OMPL),[16] and the MoveIt[17] motion planning framework are also popular packages that are frequently leveraged within the F1/10 Autonomous Racing Community. While we did not leverage these libraries within our software stack, they have displayed efficacy on numerous platforms and implement many state-of-the-art motion planning algorithms.

A.1.3.4 Reeds-Shepp Based Elastic Band Planner

In general, obtaining a collision free path for a robot within a particular environment is a computationally demanding problem and is reliant on generating useful abstractions of the real world. These abstractions are often incomplete and as a result, if a robot blindly follows an obtained path, there is a risk that collisions

[14] Our plannning source code can be found at: https://github.com/pmusau17/Planning-and-MPC
[15] http://wiki.ros.org/move_base
[16] https://ompl.kavrakilab.org/
[17] https://moveit.ros.org/

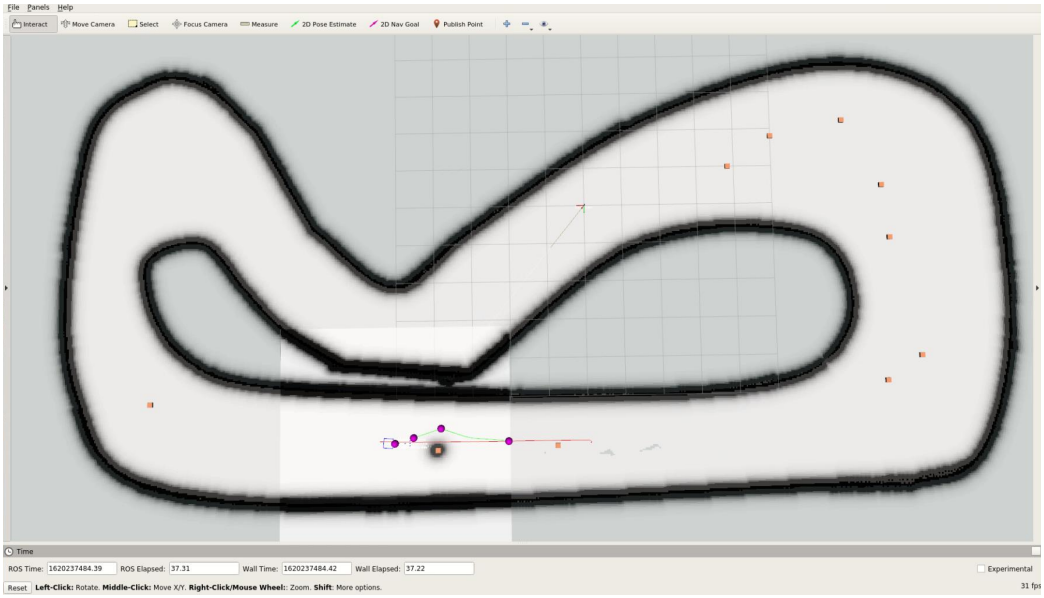

Figure A.7: Visualization of a Reeds-Shepp based Elastic Band Planning Approach. In this image, the blue rectangle corresponds to the vehicle, the red line represents the global plan obtained from the Reeds-Shepp path planner, and the green line represents the local plan obtained by deforming the global plan with artificial forces obtained from the vehicle's sensors. The orange boxes are a set of cones that the vehicle must avoid while navigating its environment.

with obstacles within the environment may still occur [374]. One way of mitigating this risk, is to use an Elastic Band Planning approach, where an initial collision free path is subjected to artificial forces that deform the path in order to maintain a safe distance from obstacles. These deformations are based on local sensor observations and allow the robot to accommodate uncertainties as they relate to the world model used in global path planning. One of the approaches we leveraged in our work was an approach that combined a Reeds-Shepp path planner with an elastic band methodology in order to achieve reactive local planning for car-like robots [375].[18]

A.1.3.5 Timed Elastic Bands (TEB)

Timed Elastic Bands are a variant of elastic band planning approaches that explicitly considers the temporal aspect of robot manoeuvres [376]. Therefore, in this setting, a planner can optimize the robot's trajectory with respect to execution time and separation from obstacles. In [376], Keller et al. presented an approach that can be used with holonomic, and non-holonomic robots. Their approach is characterized by a large set of parameters, and the interested reader may refer to their ROS documentation for further details. [19].

[18] https://github.com/gkouros/rsband_local_planner
[19] http://wiki.ros.org/teb_local_planner

A.1.3.6 Rapidly Exploring Random Trees (RRT)

Rapidly Exploring Random Trees (RRT) and its variants are a family of sampling-based path planning algorithm that has been shown to have theoretical guarantees such as probabilistic completeness [4]. What this means is that these algorithms will return a valid solution in finite time if one exists, and will return a failure otherwise. Strictly speaking, probabilistic completeness means that the probability that the planner fails to return a solution decays to zero as the number of samples used in the approach tends toward infinity [4]. These algorithms operate on an explicit representation of the environment, which is often represented as an occupancy grid. However, rather than searching the occupancy grid for candidate trajectories as in other path planning approaches, RRTs rely on a collision checking module that provides information about the validity of candidate trajectories that are formulated by connecting points sampled within the free space of the environment. These valid connections are then used to build a graph of feasible trajectories that can be used as a solution to the original motion planning problem [4].

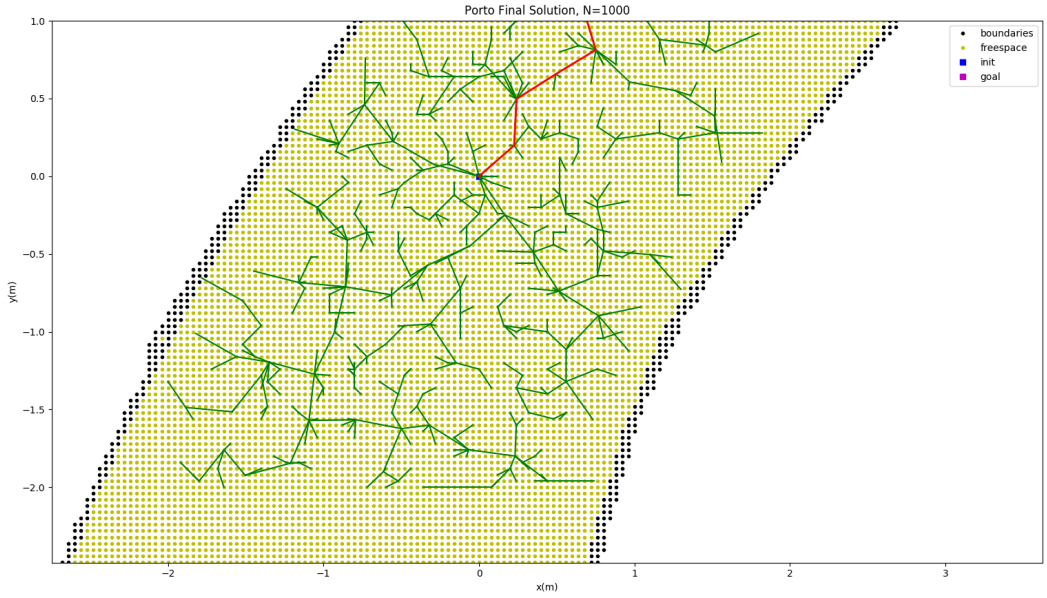

Figure A.8: Example of a path obtained using an RRT approach with 1000 random samples. [4]

RRT algorithms are extremely adept at finding paths within high dimensional spaces. However, most of these approaches are only applicable to robots with simple dynamics [377]. One of the main assumptions is that it is possible to connect two sampled points with an optimal trajectory. For holonomic robots, a straight line connecting sampled points represents the optimal trajectory.[20] However, for robots with more complicated dynamics, such trajectories are not always feasible, and there has been a large body of work proposed towards extending RRT approaches to these systems. These approaches are often referred to as

[20]We assume the robot is operating in a cartesian coordinate system here. However, for other coordinate systems, you can expand this notion to using geodesics.

Kinodynamic RRT approaches. We refer readers to the following paper for an in depth discussion of these approaches [378].

Several software tools provide implementations of RRT approaches, such as the MoveIt Library and the Open Motion Planning Library. In Figure A.8, we provide a visualization of an RRT planner leveraged within our work. It displays our own implementation based on a reeds-shepp motion planning model for car-like robots. Our implementation is available online, where interested readers can interact with the implementation more closely.[21]

A.1.4 Control Algorithms

Once the vehicle has extracted meaningful data from its sensors that allow it to interpret its environment, localized itself using a map of its environment, and generated a plan to accomplish its goals, the final phase requires generating motor outputs that allow the robot to achieve its goals [360]. Specifically, what this entails is generating the appropriate throttle, brake, and steering commands that allow a plan to be followed [379]. In this section, we describe the controllers utilized within our experiments.

A.1.4.1 Pure Pursuit

The Pure Pursuit algorithm is a widely used path-tracking algorithm that was originally designed to calculate the arc needed to get a robot back onto a path [263]. It has shown great success in being used in numerous contexts, and in this work we utilize it to design a controller that allows the F1/10 vehicle to follow a path along the center of the racetrack.

A.1.4.2 Gap Following

Obstacle avoidance is an essential component of a successful autonomous racing strategy. Gap following approaches have shown great promise in dealing with dynamic and static obstacles. They are based on the construction of a gap array around the vehicle used for calculating the best heading angle needed to move the vehicle into the center of the maximum gap [2]. In this work, we utilize a gap following controller called the "disparity extender" by Otterness et al. that won the F1/10 competition in April 2019 [230].

A.1.4.3 Model Predictive Control

Model Predictive Control (MPC), which is also known as Receding Horizon Control, is a widely used method for dealing with control problems with multivariate constraints [293]. It has demonstrated great efficacy in autonomous applications for path planning, obstacle avoidance, and in stability and reference tracking problems [294]. The basic structure of MPC is defined by utilizing an explicit model of the system under

[21] https://github.com/pmusau17/Planning-and-MPC

consideration (typically referred to as the plant), which is then used to predict the future output behaviour of the model [380]. The predictions allow for the solution of an optimal control problem online, where the difference between the predicted output and a desired reference output is minimized over a finite time horizon [293, 381]. This minimization is often defined by a set of constraints which represent the structural or physical limitations of the plant and its environment.

Within the autonomous racing space [273, 2], MPC has been frequently used as a method for realizing high performance racing strategies. In this regime, the optimization problem is formulated by balancing the need for high velocities and obstacle avoidance. In the work presented in this dissertation, we designed an MPC framework for generating a sequence of control inputs that steer an autonomous vehicle from a starting point to a goal location while avoiding obstacles. Our framework used a nonlinear kinematic bicycle model to model the F1/10 platform, making the MPC problem a nonlinear model predictive control problem. As a result, in general, this results in a non-convex optimization problem [307]. While it is possible to linearize the dynamics used in our framework and convert the problem into a linear MPC formulation, we elected not to do so. Primarily because of the recent development of tools that can solve NMPC problems efficiently, as well a desire to try and capture the nonlinear dynamics of the F1/10 vehicle with a higher degree of fidelity.

We used the software-tool *do-mpc* [308] to solve the nonlinear model predictive problem, due to its modular and transparent implementation as well as its efficient handling of nonlinear optimization problems.[22] The tool utilizes an orthogonal collocation of finite-elements method to discretize the NMPC problem into a form that can be comfortably handled by state-of-the-art optimization tools. We refer interested readers to the following paper for an in depth discussion of do-mpc [308]. Details with respect to the objective function utilized within our work, can be found in the source code and accompanying documentation.[23]

A.2 Machine Learning Approaches

A.2.1 Reinforcement Learning Approaches

Reinforcement Learning (RL) is a branch of machine learning that focuses on software agents learning to maximize rewards in an environment through experience. In recent years, it has been used to solve many complex problems in many application domains. In the following section, we present the RL approaches that we leveraged within our work. These methods were implementing using the open source machine learning framework Pytorch[24] in order to train and deploy our RL agents. All the training and evaluation scripts are available in our simulation packages, in addition to instructions on how to run training campaigns and get started.

[22] https://www.do-mpc.com/en/latest/
[23] https://github.com/pmusau17/Platooning-F1Tenth/tree/noetic-port/src/mpc
[24] https://pytorch.org/

A.2.1.1 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is explained in [149] by Lillicrap et al. The method uses model-free, off-policy learning to determine an optimal deterministic policy for the agent to follow. This means, during training, the agent executes random actions not determined using the learned policy. This string of randomly executed actions is stored in a replay buffer, which is sampled from after each step to learn an estimation of the state-action value (a.k.a. Q-value) function. This Q-function is used to train the policy to take actions that result in the largest Q-value.

A.2.1.2 Soft Actor Critic

This algorithm was first introduced in 2018 as an improvement to *Deep Deterministic Policy Gradient* (DDPG) that tackled RL's major challenges: high sample complexity and brittle convergence properties, i.e. a heavy dependence of hyperparameters being "just right" in order to effectively learn [188]. They address these issues using a maximum entropy reinforcement learning framework. Instead of allowing the agent to explore the environment randomly while training, the actions are selected in a way that maximizes the number of unique training samples that are collected. Thus, the sample complexity is reduced because the process of collecting unique training samples is more efficient.

A.2.1.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is explained in [382] by Schulman et al. The method was proposed as a simplification of the well-known Trust Region Policy Optimization (TRPO) technique and uses a model-free, on-policy learning technique in order to determine the optimal stochastic policy for the agent to follow. This means, during training, the agent executes actions randomly chosen from the output policy distribution. The policy is followed over the course of a horizon. After the horizon is completed, the advantages are computed and used to determine the effectiveness of the policy. The advantage values are used along with the probability of the action being taken to compute a loss function, which is then clipped to prevent large changes in the policy. The clipped loss is used to update the policy and improve future advantage estimation.

A.2.2 Imitation Learning

Imitation Learning (IL) is a branch of machine learning that seeks to reproduce the behaviour of a human or domain expert on a given task [185].These methods fall under the branch of *Expert Systems* in AI which has seen a surge in interest in recent years. The increase in demand for these approaches is spurred on by two main motivations. (1) In many settings, the number of possible actions needed to execute a complex task is too large to cover using explicit programming. (2) Demonstrations show that having prior knowledge

provided by an expert is more efficient than learning from scratch [185].

One of the most common imitation learning methods is *Behaviour Cloning*, whereby a controller is constructed by learning a mapping from sensor-action pairs collected either from a baseline controller or through human-in-the-loop control. In this section, we describe the two behaviour cloning methods, used within our work. The first leverages images captured from a camera in order to determine the steering angle that the vehicle should use, while the second uses the LiDAR to perform the same prediction.

A.2.2.1 Vision Based Learning

Vision-based perception systems for autonomous driving can be broadly classified into two categories: mediated-perception approaches and end-to-end, or behavioral-reflex, approaches [383]. Mediated perception approaches involve multiple sub-components for recognizing objects relevant to the driving task, that are then combined into a world representation utilized by a decision manager to issue control actions. In contrast, end-to-end approaches compute a direct mapping from images to control actions [383]. While end-to-end approaches have proven to be an elegant and effective reduction of a complex system, they are ”black box” in nature which makes them difficult to debug. Within the context of this manuscript, we opted to use end-to-end approaches because of their straightforward incorporation within the simplex architecture, and they allow us to evaluate our safety regime more plainly.

Since the seminal work of Krizhevsky et al. [229] in the ImageNet Large Scale Recognition Challenge, *Convolutional Neural Networks* (CNNs) have revolutionized the field of computer vision. Within the context of autonomous vehicles, CNNs have demonstrated efficacy in being used to enable driving tasks such as lane following, path planning, and control, simultaneously, by computing steering commands directly from images [383]. In [187], Bojarski et al. utilized a CNN architecture called DAVE-2 to control a 2016 Lincoln MKZ for over 10 miles in a series of successful on-road tests. For this reason, we selected this architecture for our experiments. The architecture is show in Figure A.9.

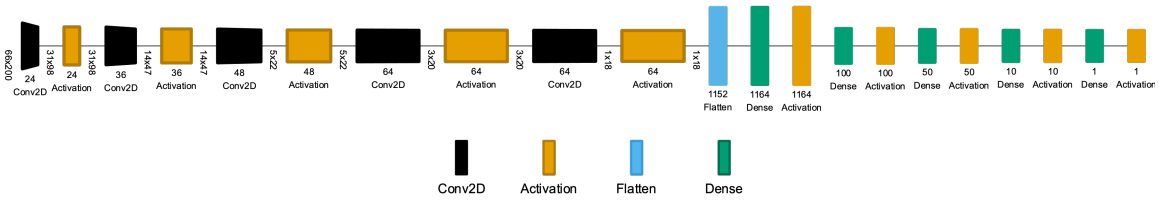

Figure A.9: Dave Architecture

The data we used to train DAVE-2 was collected from a set of simulation experiments where the sensor-action pairs were generated by a path tracking controller optimized to keep the F1/10 in the center of the track. Examples of the data used within our experiments is shown in Figure A.10.

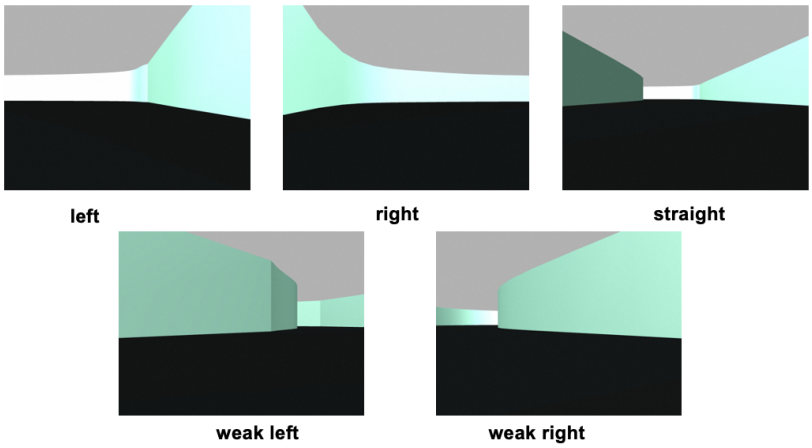

Figure A.10: Example of Classes that were used in a discrete version of an end-to-end (image to steering angle) classification task.

A.2.2.2 Lidar Behaviour Cloning (LBC)

The second method utilized for behaviour cloning made of use of the LiDAR mounted on the F1/10 Vehicle and utilized a standard multi-layer perceptron network that consists of an input layer, 2 fully connected hidden layers of 64 neurons with ReLU activation functions, and a fully connected output layer with a tanh activation function. To reduce the size of the networks utilized within our approach. he input layer accepts nine range values collected from the LiDAR at $-90°$, $-60°$, $-45°$, $-30°$, $0°$, $30°$, $45°$, $60°$, and $90°$ from forward. The range values are clipped between $[0m, 10m]$. The data used to train this controller was collected in the same fashion as the vision based network described in the previous section.

A.3 System Identification

The objective of system identification is to develop a mathematical model of a specific dynamic system using data collected from a series of experiments [384]. The experimental data is often then combined with prior knowledge about the basic mechanics and dynamics of the underlying system, as well as the control signals that are provided to the system. Depending on the level of knowledge that the system designer has about the system, different mathematical models describing the operation of the system can be obtained. These include *White-box models*, where the dynamics of the system are described using well known mathematical models, *Black-box models* where very little is known about the underlying system resulting in a completely empirical description of the system, and *Grey-box models*, which are a combination of white-box and black-box models [384].

In our work, we utilized a grey-box model in order to model the dynamics of the F1/10 system. The data used to identify the model was collected by running a set of experiments that capture the relationship between

the control inputs provided to the platform and the resulting outputs. The underlying model describing the vehicle is a kinematic bicycle model and the task is to identify the acceleration constant, motor constant, and hysteresis constant that define its behaviour. The identification was performed using MATLAB's Grey-Box System Identification toolbox[25]. All the data used to identify the model as well as the scripts used to generate it are available in the Zenodo repository containing the source code for our experiments.

[25] https://www.mathworks.com/help/ident/grey-box-model-estimation.html

BIBLIOGRAPHY

[1] X. Ge, R. Rijo, R. F. Paige, T. P. Kelly, and J. A. McDermid, "Introducing goal structuring notation to explain decisions in clinical practice," *Procedia Technology*, vol. 5, pp. 686–695, 2012. 4th Conference of Enterprise Information Systems aligning technology, organizations and people (CENTERIS 2012).

[2] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track* (H. J. Escalante and R. Hadsell, eds.), vol. 123 of *Proceedings of Machine Learning Research*, pp. 77–89, PMLR, 08–14 Dec 2020.

[3] C. H. Walsh and S. Karaman, "CDDT: fast approximate 2d ray casting for accelerated localization," *CoRR*, vol. abs/1705.01167, 2017.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[5] P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, and et al., "Artificial intelligence and life in 2030.," Sep 2016.

[6] J. M. Mueller, "The abcs of assured autonomy," in *International Symposium on Technology and Society, Proceedings*, vol. 2019-November, 2019.

[7] S. Russell, D. Dewey, and M. Tegmark, "Research Priorities for Robust and Beneficial Artificial Intelligence," *AI Magazine*, vol. 36, no. 4, p. 105, 2015.

[8] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[9] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. A. Rosenfeld, and T. T. Johnson, "Verification for machine learning, autonomy, and neural networks survey," *CoRR*, vol. abs/1810.01989, 2018.

[10] J. Rushby, "The interpretation and evaluation of assurance cases," Tech. Rep. SRI-CSL-15-01, Computer Science Laboratory, SRI International, Menlo Park, CA, July 2015. Available at http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf.

[11] J. Rushby, "The interpretation and evaluation of assurance cases," Tech. Rep. SRI-CSL-15-01, Computer Science Laboratory, SRI International, Menlo Park, CA, July 2015. Available at http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf.

[12] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," *CoRR*, vol. abs/1702.01135, 2017.

[13] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, (San Francisco, CA), pp. 99–107, IEEE, 2009.

[14] A. Rashid, U. Siddique, and S. Tahar, "Formal verification of cyber-physical systems using theorem proving," in *Communications in Computer and Information Science*, vol. 1165 CCIS, 2020.

[15] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[16] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 1–10, 2022.

[17] N. Hamilton, P. Musau, D. M. Lopez, and T. T. Johnson, "Zero-shot policy transfer in autonomous racing: Reinforcement learning vs imitation learning," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 11–20, 2022.

[18] H. Dezfuli, *NASA System Safety Handbook*. CreateSpace Independent Publishing Platform, 2012.

[19] J. Rushby, "Runtime certification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5289 LNCS, 2008.

[20] J. Sztipanovits, T. Bapty, X. Koutsoukos, Z. Lattmann, S. Neema, and E. Jackson, "Model and tool integration platforms for cyber–physical system design," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1501–1526, 2018.

[21] E. M. Clarke, J. M. Wing, R. Alur, E. Clarke, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger, G. Holzmann, C. Jones, R. Kurshan, N. Leveson, K. McMillan, J. Moore, D. Peled, A. Pnueli, J. Rushby, N. Shankar, J. Sifakis, P. Sistla, B. Steffen, P. Wolper, J. Wing, J. Woodcock, and P. Zave, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[22] R. Jhala and R. Majumdar, "Software model checking," *ACM Computing Surveys*, vol. 41, 2009.

[23] J. A. McDermid, "Formal methods: use and relevance for the development of safety-critical systems," in *Safety Aspects of Computer Control* (P. Bennett, ed.), pp. 96–153, Butterworth-Heinemann, 1993.

[24] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*, ch. 1, pp. 1–30. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[25] O. Maler, "Algorithmic verification of continuous and hybrid systems," in *Proceedings 15th International Workshop on Verification of Infinite-State Systems, INFINITY 2013, Hanoi, Vietnam, 14th October 2013* (L. Holík and L. Clemente, eds.), vol. 140 of *EPTCS*, pp. 48–69, 2013.

[26] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, 2003.

[27] L. Doyen, G. Frehse, G. J. Pappas, and A. Platzer, *Verification of Hybrid Systems*. Cham: Springer International Publishing, 2018.

[28] J. Yang, M. A. Islam, A. Murthy, S. A. Smolka, and S. D. Stoller, "A simplex architecture for hybrid systems using barrier certificates," in *Computer Safety, Reliability, and Security* (S. Tonetta, E. Schoitsch, and F. Bitsch, eds.), (Cham), pp. 117–131, Springer International Publishing, 2017.

[29] M. Clark, X. Koutsoukos, J. Porter, R. Kumar, G. J. Pappas, O. Sokolsky, I. Lee, and L. Pike, "A study on run time assurance for complex cyber physical systems," tech. rep., Aerospace Systems Directorate, Air Force Research Lab, Wright-Patterson Air Force Base, 2013.

[30] C. Kwon and I. Hwang, "Reachability analysis for safety assurance of cyber-physical systems against cyber attacks," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 2272–2279, 2018.

[31] J. Harrison, "Theorem Proving for Verification (Invited Tutorial)," in *Computer Aided Verification* (A. Gupta and S. Malik, eds.), (Berlin, Heidelberg), pp. 11–18, Springer Berlin Heidelberg, 2008.

[32] O. A. Beg, H. Abbas, T. T. Johnson, and A. Davoudi, "Model validation of pwm dc–dc converters," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7049–7059, 2017.

[33] . S. S. A. Lee, E. A., *Introduction to Embedded Systems. A Cyber-Physical Systems Approach. Second Edition*, vol. 195. MIT Press, 2017.

[34] A. Gupta, "Formal hardware verification methods: A survey," in *Computer-Aided Verification: A Special Issue of Formal Methods In System Design on Computer-Aided Verification* (R. Kurshan, ed.), (Boston, MA), pp. 5–92, Springer US, 1993.

[35] L. Zhang, "Specification and design of cyber physical systems based on system of systems engineering approach," in *2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pp. 300–303, 2018.

[36] Y. Driouich, M. Parente, and E. Tronci, "Modeling cyber-physical systems for automatic verification," in *2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 1–4, 2017.

[37] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *IEEE Transactions on Automatic Control*, vol. 48, pp. 64–75, jan 2003.

[38] H. Täubig, U. Frese, C. Hertzberg, C. Lüth, S. Mohr, E. Vorobev, and D. Walter, "Guaranteeing functional safety: Design for provability and computer-aided verification," *Autonomous Robots*, vol. 32, no. 3, pp. 303–331, 2012.

[39] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler, "Recent progress in continuous and hybrid reachability analysis," *Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, CACSD*, pp. 1582–1587, 2007.

[40] A. Aerts, B. Tong Minh, M. R. Mousavi, and M. A. Reniers, "Temporal logic falsification of cyber-physical systems: An input-signal-space optimization approach," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 214–223, 2018.

[41] M. Sirjani, E. A. Lee, and E. Khamespanah, "Verification of cyberphysical systems," *Mathematics*, vol. 8, no. 7, 2020.

[42] B. De Schutter, W. P. M. H. Heemels, J. Lunze, and C. Prieur, *Survey of modeling, analysis, and control of hybrid systems*. Cambridge University Press, 2009.

[43] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Information and Computation*, vol. 185, no. 1, pp. 105–157, 2003.

[44] S. N. Krishna and A. Trivedi, "Hybrid automata for formal modeling and verification of cyber-physical systems," *Journal of the Indian Institute of Science*, vol. 93, 2013.

[45] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design and analysis of hybrid systems, with applications to robotic aerial vehicles," in *Robotics Research* (C. Pradalier, R. Siegwart, and G. Hirzinger, eds.), (Berlin, Heidelberg), pp. 139–149, Springer Berlin Heidelberg, 2011.

[46] A. Rocca, V. Acary, and B. Brogliato, "Index-2 hybrid dae: a case study with well-posedness and numerical analysis," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1888–1893, 2020. 21st IFAC World Congress.

[47] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, 2021.

[48] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[49] C. Desoer and J. Wing, "A minimal time discrete system," *IRE Transactions on Automatic Control*, vol. 6, no. 2, pp. 111–125, 1961.

[50] S. V. Rakovic, E. C. Kerrigan, D. Q. Mayne, and J. Lygeros, "Reachability analysis of discrete-time systems with disturbances," *IEEE Transactions on Automatic Control*, vol. 51, no. 4, pp. 546–561, 2006.

[51] E. Asarin, T. Dang, and A. Girard, "Reachability analysis of nonlinear systems using conservative approximation," in *Hybrid Systems: Computation and Control* (O. Maler and A. Pnueli, eds.), (Berlin, Heidelberg), pp. 20–35, Springer Berlin Heidelberg, 2003.

[52] M. Althoff, "An introduction to cora 2015," in *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems* (G. Frehse and M. Althoff, eds.), vol. 34 of *EPiC Series in Computing*, (Berlin, Germany), pp. 120–151, EasyChair, 2015.

[53] G. Lafferriere, G. J. Pappas, and S. Yovine, "Reachability computation for linear hybrid systems," *IFAC Proceedings Volumes*, vol. 32, no. 2, pp. 2137–2142, 1999. 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July.

[54] J. V. Deshmukh and S. Sankaranarayanan, "Formal techniques for verification and testing of cyber-physical systems," in *Design Automation of Cyber-Physical Systems*, 2019.

[55] Z. Han and B. H. Krogh, "Reachability analysis of large-scale affine systems using low-dimensional polytopes," in *Hybrid Systems: Computation and Control* (J. P. Hespanha and A. Tiwari, eds.), (Berlin, Heidelberg), pp. 287–301, Springer Berlin Heidelberg, 2006.

[56] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2242–2253, 2017.

[57] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *Formal Methods – The Next 30 Years* (M. H. ter Beek, A. McIver, and J. N. Oliveira, eds.), (Cham), pp. 670–686, Springer International Publishing, 2019.

[58] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *2012 IEEE 33rd Real-Time Systems Symposium*, pp. 183–192, 2012.

[59] S. Bak and P. S. Duggirala, "Hylaa: A tool for computing simulation-equivalent reachability for linear systems," in *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, (New York, NY, USA), p. 173–178, Association for Computing Machinery, 2017.

[60] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification* (N. Sharygina and H. Veith, eds.), (Berlin, Heidelberg), pp. 258–263, Springer Berlin Heidelberg, 2013.

[61] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software," *IEEE Trans. Softw. Eng.*, vol. 19, p. 3–12, Jan. 1993.

[62] K. Das, A. Gurung, and R. Ray, "Parallel Simulation of Cyber-Physical Systems," *Innovations in Systems and Software Engineering*, no. March, 2021.

[63] J. J. Majikes, R. Pandita, and T. Xie, "Literature review of testing techniques for medical device software," *Proceedings of the 4th Medical Cyber*, 2013.

[64] S. A. Asadollah, R. Inam, and H. Hansson, "A survey on testing for cyber physical system," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9447, 2015.

[65] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2e2: A verification tool for stateflow models," in *Tools and Algorithms for the Construction and Analysis of Systems* (C. Baier and C. Tinelli, eds.), (Berlin, Heidelberg), pp. 68–82, Springer Berlin Heidelberg, 2015.

[66] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Computer Aided Verification* (G. Gopalakrishnan and S. Qadeer, eds.), (Berlin, Heidelberg), pp. 379–395, Springer Berlin Heidelberg, 2011.

[67] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi, "Enabling model testing of cyber-physical systems," in *Proceedings - 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018*, 2018.

[68] D. J. Richardson, T. O. O'Malley, and C. Tittle, "Approaches to specification-based testing," in *In Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification (TAV3*, pp. 86–96, ACM Press, 1989.

[69] A. Suzuki, K. Masutomi, I. Ono, H. Ishii, and T. Onoda, "Cps-sim: Co-simulation for cyber-physical systems with accurate time synchronization," *IFAC-PapersOnLine*, vol. 51, 2018.

[70] M. Woehrle, K. Lampka, and L. Thiele, "Conformance testing for cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, Jan. 2013.

[71] L. M. Barroca and J. A. Mcdermid, "Formal methods: Use and relevance for the development of safety-critical systems," *Computer Journal*, vol. 35, 1992.

[72] W. Torres-pomales, "Software fault tolerance: A tutorial," tech. rep., Langley Research Center, NASA, 2000.

[73] L. Pike and A. Goodloe, "Monitoring distributed real-time systems: A survey and future directions," tech. rep., The National Aeronautics and Space Administration, 2010.

[74] C. J. Dennehy and L. M. Fesq, "The development of nasa's fault management handbook," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 8, 2012.

[75] S. Sader, I. Husti, and M. Daróczi, "Enhancing failure mode and effects analysis using auto machine learning: A case study of the agricultural machinery industry," *Processes*, vol. 8, 2020.

[76] A. Aniculaesei, D. Arnsberger, F. Howar, and A. Rausch, "Towards the verification of safety-critical autonomous systems in dynamic environments," *Electronic Proceedings in Theoretical Computer Science, EPTCS*, vol. 232, pp. 79–90, 2016.

[77] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger, "Introduction to runtime verification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10457 LNCS, pp. 1–33, 2018.

[78] D. Boursinos and X. Koutsoukos, "Trusted confidence bounds for learning enabled cyber-physical systems," in *2020 IEEE Security and Privacy Workshops (SPW)*, (San Francisco, CA), pp. 228–233, IEEE, 2020.

[79] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, 2005.

[80] A. Desai, T. Dreossi, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," in *Runtime Verification* (S. Lahiri and G. Reger, eds.), (Cham), pp. 172–189, Springer International Publishing, 2017.

[81] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, 2017.

[82] L. Masson, J. Guiochet, H. Waeselynck, K. Cabrera, S. Cassel, and M. Törngren, "Tuning permissiveness of active safety monitors for autonomous systems," in *NASA Formal Methods* (A. Dutle, C. Muñoz, and A. Narkawicz, eds.), (Cham), pp. 333–348, Springer International Publishing, 2018.

[83] A. K. Akametalu, S. Kaynama, J. F. Fisac, M. N. Zeilinger, J. H. Gillula, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 2015-February, (Los Angeles, CA), pp. 1424 – 1431, IEEE, 2014.

[84] S. Mitsch and A. Platzer, "Modelplex: verified runtime validation of verified cyber-physical system models," *Formal Methods in System Design*, vol. 49, no. 1, pp. 33–74, 2016.

[85] C. Daws and S. Tripakis, "Model checking of real-time reachability properties using abstractions," in *Tools and Algorithms for the Construction and Analysis of Systems* (B. Steffen, ed.), (Berlin, Heidelberg), pp. 313–329, Springer Berlin Heidelberg, 1998.

[86] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural simplex architecture," in *NASA Formal Methods* (R. Lee, S. Jha, and A. Mavridou, eds.), (Cham), pp. 97–114, Springer International Publishing, 2020.

[87] M. Leucker and C. Schallhart, "A brief account of runtime verification," *Journal of Logic and Algebraic Programming*, vol. 78, pp. 293–303, 2009.

[88] Y. Falcone, K. Havelund, and G. Reger, "A tutorial on runtime verification," *Engineering Dependable Software Systems*, vol. 34, pp. 141–175, 2013.

[89] A. Arpaci-Dusseau and G. M. Voelker, "Introduction to the special section on osdi'18," *ACM Transactions on Storage*, vol. 15, pp. 1–5, 2019.

[90] O. Sokolsky, K. Havelund, and I. Lee, "Introduction to the special section on runtime verification," *International Journal on Software Tools for Technology Transfer*, vol. 14, 2012.

[91] K. Bae and J. Lee, "Bounded model checking of signal temporal logic properties using syntactic separation," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[92] K. Havelund and A. Goldberg, "Verify your runs," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4171 LNCS, pp. 374–383, 2008.

[93] T. Gurriet, M. Mote, A. D. Ames, and E. Feron, "An online approach to active set invariance," in *2018 IEEE Conference on Decision and Control (CDC)*, (Miami, FL), pp. 3592–3599, IEEE, 2018.

[94] S. L. Herbert, S. Bansal, S. Ghosh, and C. J. Tomlin, "Reachability-based safety guarantees using efficient initializations," in *Proceedings of the IEEE Conference on Decision and Control*, vol. 2019-December, (Nice, France), pp. 4810 – 4816, IEEE, 2019.

[95] A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, J. F. Fisac, S. Deglurkar, A. D. Dragan, and C. J. Tomlin, "A scalable framework for real-time multi-robot, multi-human collision avoidance," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, 2019.

[96] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin, "An efficient reachability-based framework for provably safe autonomous navigation in unknown environments," *Proceedings of the IEEE Conference on Decision and Control*, vol. 2019-December, pp. 1758 – 1765, 2019.

[97] S. Bansal, A. Bajcsy, E. Ratner, A. D. Dragan, and C. J. Tomlin, "A hamilton-jacobi reachability-based framework for predicting and analyzing human motion for safe planning," *CoRR*, vol. abs/1910.13369, 2019.

[98] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, pp. 2737 – 2752, 2019.

[99] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Proceedings of the Conference on Robot Learning* (L. P. Kaelbling, D. Kragic, and K. Sugiura, eds.), vol. 100 of *Proceedings of Machine Learning Research*, pp. 420–429, PMLR, 30 Oct–01 Nov 2020.

[100] A. Gattami, A. Al Alam, K. H. Johansson, and C. J. Tomlin, "Establishing safety for heavy duty vehicle platooning: A game theoretical approach," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 3818 – 3823, 2011. 18th IFAC World Congress.

[101] O. Bokanowski, N. Forcadel, and H. Zidani, "Reachability and minimal times for state constrained nonlinear problems without any controllability assumption," *SIAM J. Control Optim.*, vol. 48, p. 4292–4316, June 2010.

[102] M. Chen, Q. Hu, C. Mackin, J. F. Fisac, and C. J. Tomlin, "Safe platooning of unmanned aerial vehicles via reachability," in *2015 54th IEEE Conference on Decision and Control (CDC)*, (Osaka, Japan), pp. 4695–4701, IEEE, 2015.

[103] A. Dhinakaran, M. Chen, G. Chou, J. C. Shih, and C. J. Tomlin, "A hybrid framework for multi-vehicle collision avoidance," in *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, vol. 2018-January, (Melbourne, Australia), pp. 2979 – 2984, IEEE, 2018.

[104] H.-D. Tran, L. V. Nguyen, P. Musau, W. Xiang, and T. T. Johnson, "Decentralized real-time safety verification for distributed cyber-physical systems," in *Formal Techniques for Distributed Objects, Components, and Systems* (J. A. Pérez and N. Yoshida, eds.), (Cham), pp. 261–277, Springer International Publishing, 2019.

[105] R. E. Allen, A. A. Clark, J. A. Starek, and M. Pavone, "A machine learning approach for real-time reachability analysis," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2202–2208, 2014.

[106] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *ACM Trans. Embed. Comput. Syst.*, vol. 15, feb 2016.

[107] J. Rivera, A. Danylyszyn, C. Weinstock, L. Sha, and M. Gagliardi, "An architectural description of the simplex architecture," Tech. Rep. CMU/SEI-96-TR-006, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[108] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," in *2014 IEEE Real-Time Systems Symposium*, (Rome, Italy), pp. 138–148, IEEE, 2014.

[109] U. Mehmood, S. Bak, S. A. Smolka, and S. D. Stoller, "Safe CPS from Unsafe Controllers," *arXiv e-prints*, p. arXiv:2102.12981, Feb. 2021.

[110] D. Seto, E. Ferriera, and T. Marz, "Case study: Development of a baseline controller for automatic landing of an f-16 aircraft using linear matrix inequalities (lmis)," Tech. Rep. CMU/SEI-99-TR-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[111] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. 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)*, (Tucson, AZ), pp. 400–412, IEEE, 2007.

[112] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "Soter: A runtime assurance framework for programming safe robotics systems," in *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019*, (Portland, OR), pp. 138 – 150, IEEE, 2019.

[113] H. Tran, W. Xiang, and T. T. Johnson, "Verification approaches for learning-enabled autonomous cyber-physical systems," *IEEE Design Test*, pp. 1–1, 2020.

[114] C. Liu, T. Arnon, C. Lazarus, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019.

[115] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[116] Z. C. Lipton, "A Critical Review of Recurrent Neural Networks for Sequence Learning," *CoRR*, vol. abs/1506.00019, 2015.

[117] S. E. V.T. and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *IEEE Transactions on Neural Networks*, vol. 5, pp. 594–603, July 1994.

[118] H. Sak, A. W. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *CoRR*, vol. abs/1402.1128, 2014.

[119] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, pp. 1464–1480, Sept 1990.

[120] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1989.

[121] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), pp. 1–14, 2015.

[122] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[123] Y. Sun, X. Huang, and D. Kroening, "Testing Deep Neural Networks," *CoRR*, vol. abs/1803.04792, 2018.

[124] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A Dual Approach to Scalable Verification of Deep Networks," *CoRR*, vol. abs/1803.06567, 2018.

[125] G. Katz, ""Verification of Machine Learning Programs"," 7 2018. St Anne's College (Oxford), Oxford, England, July 4, 2018.

[126] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, (New York, NY, USA), pp. 169–178, ACM, 2019.

[127] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output Range Analysis for Deep Neural Networks," *CoRR*, vol. abs/1709.09130, 2017.

[128] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," *CoRR*, vol. abs/1706.07351, 2017.

[129] A. Rössig and M. Petkovic, "Advances in verification of relu neural networks," *Journal of Global Optimization*, 2020.

[130] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems*, vol. 2018-December, 2018.

[131] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar, "Piecewise Linear Neural Network verification: A Comparative Study," *CoRR*, vol. abs/1711.00455, 2017.

[132] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring Neural Net Robustness with Constraints," *CoRR*, vol. abs/1605.07262, 2016.

[133] V. Tjeng and R. Tedrake, "Verifying Neural Networks with Mixed Integer Programming," *CoRR*, vol. abs/1711.07356, 2017.

[134] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A Dual Approach to Scalable Verification of Deep Networks," *CoRR*, vol. abs/1803.06567, 2018.

[135] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *35th International Conference on Machine Learning, ICML 2018*, vol. 12, 2018.

[136] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *35th International Conference on Machine Learning, ICML 2018*, vol. 8, 2018.

[137] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, 2020.

[138] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety Verification of Deep Neural Networks," *CoRR*, vol. abs/1610.06940, 2016.

[139] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[140] T. W. Weng, H. Zhang, H. Chen, Z. Song, C. J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," in *35th International Conference on Machine Learning, ICML 2018*, vol. 12, 2018.

[141] H. Zhang, T. W. Weng, P. Y. Chen, C. J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems*, vol. 2018-December, 2018.

[142] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, (New York, NY, USA), pp. 157–168, ACM, 2019.

[143] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," *CoRR*, vol. abs/1810.13072, 2018.

[144] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," in *Safe, Autonomous and Intelligent Vehicles* (H. Yu, X. Li, R. M. Murray, S. Ramesh, and C. J. Tomlin, eds.), pp. 123–144, Springer International Publishing, 2019.

[145] W. Xiang, H.-D. Tran, J. Rosenfeld, and T. T. Johnson, "Reachable set estimation and verification for a class of piecewise linear systems with neural network controllers," in *American Control Conference (ACC 2018), Special Session on Formal Methods in Controller Synthesis I*, IEEE, June 2018.

[146] S. Bak, "Reducing the wrapping effect in flowpipe construction using pseudo-invariants (work in progress)," in *Proceedings of the 4th ACM Workshop on Design, Modeling and Evaluation of Cyber Physical Systems, CyPhy 2014*, 2014.

[147] D. M. Lopez, P. Musau, H.-D. Tran, S. Dutta, T. J. Carpenter, R. Ivanov, and T. T. Johnson, "Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems* (G. Frehse and M. Althoff, eds.), vol. 61 of *EPiC Series in Computing*, (Montreal, QC, Canada), pp. 103–119, EasyChair, April 2019.

[148] T. T. Johnson, D. M. Lopez, P. Musau, H.-D. Tran, E. Botoeva, F. Leofante, A. Maleki, C. Sidrane, J. Fan, and C. Huang, "Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants," in *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)* (G. Frehse and M. Althoff, eds.), vol. 74 of *EPiC Series in Computing*, pp. 107–139, EasyChair, 2020.

[149] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, OpenReview, 2016.

[150] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[151] P. A. Andersen, M. Goodwin, and O. C. Granmo, "Towards safe reinforcement-learning in industrial grid-warehousing," *Information Sciences*, vol. 537, 2020.

[152] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, 2015.

[153] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[154] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, 2007.

[155] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013*, vol. 2, 2013.

[156] S. P. Coraluppi and S. I. Marcus, "Risk-sensitive and minimax control of discrete-time, finite-state markov decision processes," *Automatica*, vol. 35, 1999.

[157] N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, A. Lerchner, R. Keramati, J. H. Whang, P. Cho, and E. Brunskill, "Strategic exploration in object-oriented reinforcement learning," *Icml 2018 Erl*, 2018.

[158] A. Baranes and P. Y. Oudeyer, "Robust intrinsically motivated exploration and active learning," in *2009 IEEE 8th International Conference on Development and Learning, ICDL 2009*, 2009.

[159] A. Basu, T. Bhattacharyya, and V. S. Borkar, "A learning algorithm for risk-sensitive cost," *Mathematics of Operations Research*, vol. 33, 2008.

[160] V. S. Borkar, "A sensitivity formula for risk-sensitive cost and the actor-critic algorithm," *Systems and Control Letters*, vol. 44, 2001.

[161] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause, "Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes," in *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, 2016.

[162] M. Pecka and T. Svoboda, "Safe exploration techniques for reinforcement learning – an overview," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8906, 2014.

[163] Y. Sui, A. Gotovos, J. W. Burdick, and A. Krause, "Safe exploration for optimization with gaussian processes," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, 2015.

[164] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, 2009.

[165] C. M. Yin, H. X. Wang, and F. Zhao, "Risk-sensitive reinforcement learning algorithms with generalized average criterion," *Applied Mathematics and Mechanics (English Edition)*, vol. 28, 2007.

[166] S. Baskiyar and N. Meghanathan, "A survey of contemporary real-time operating systems," *Informatica (Ljubljana)*, vol. 29, 2005.

[167] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," in *Foundations of Real-Time Computing: Scheduling and Resource Management* (A. M. van Tilborg and G. M. Koob, eds.), (Boston, MA), pp. 203–249, Springer US, 1991.

[168] G. P. Arya, K. Nilay, and D. Prasad, "An improved round robin cpu scheduling algorithm based on priority of process," *International Journal of Engineering and Technology(UAE)*, vol. 7, 2018.

[169] F. Jaramillo, B. Keles, and M. Erkoc, "Modeling single machine preemptive scheduling problems for computational efficiency," *Annals of Operations Research*, vol. 285, 2020.

[170] L. Epstein and A. Levin, "Robust algorithms for preemptive scheduling," *Algorithmica*, vol. 69, 2014.

[171] A. A. Bertossi and A. Fusiello, "Rate-monotonic scheduling for hard-real-time systems," *European Journal of Operational Research*, vol. 96, 1997.

[172] O. A. E. O.-O. I. F. A. J. A., "An analytical survey of real time system scheduling techniques," *International Journal of Science and Research (IJSR)*, vol. 3, 2014.

[173] F. Mueller, "Challenges for cyber-physical systems : Security , timing analysis and soft error protection," *National Workshop on High Confidence Software Platforms for Cyber-Physical Systems: Research Needs and Roadmap (HCSP-CPS)*, 2006.

[174] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna, "F1/10: an open-source autonomous cyber-physical platform," *CoRR*, vol. abs/1901.08567, 2019.

[175] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *CoRR*, vol. abs/1911.01562, 2019.

[176] D. Murphy, "Tum autonomous motorsport wins the indy autonomous challenge powered by cisco at the indianapolis motor speedway and the $1 million grand prize," Oct 2021.

[177] G. Velasco-Hernandez, D. J. Yeong, J. Barry, and J. Walsh, "Autonomous driving architectures, perception and data fusion: A review," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 315–321, 2020.

[178] M. Han, Y. Tian, L. Zhang, J. Wang, and W. Pan, "Reinforcement learning control of constrained dynamic systems with uniformly ultimate boundedness stability guarantee," *Automatica*, vol. 129, p. 109689, 2021.

[179] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[180] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[181] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[182] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. A. Malikopoulos, and A. M. Bayen, "Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS*, (Montreal, QC, Canada), pp. 291–300, IEEE, 2019.

[183] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, "Sim2real predictivity: Does evaluation in simulation predict real-world performance?," *IEEE Robotics and Automation Letters*, vol. 5, pp. 6670 – 6677, 2020.

[184] T. P. Gros, D. Höller, J. Hoffmann, and V. Wolf, "Tracking the race between deep reinforcement learning and imitation learning - extended version," *CoRR*, vol. abs/2008.00766, 2020.

[185] A. Hussein, M. Gaber, E. Elyan, and C. Jayne, "Imitation learning," *ACM Computing Surveys (CSUR)*, vol. 50, pp. 1 – 35, 2017.

[186] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural computation*, vol. 3, no. 1, pp. 88–97, 1991.

[187] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. J. Ackel, U. Muller, P. Yeres, and K. Zieba, "Visualbackprop: Efficient visualization of cnns for autonomous driving," in *Proceedings - IEEE International Conference on Robotics and Automation*, (Brisbane, Australia), pp. 4701 – 4708, IEEE, 2018.

[188] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *35th International Conference on Machine Learning, ICML 2018*, vol. 5, (Stockholm, Sweden), PMLR, 2018.

[189] J. Kerr and K. Nickels, "Robot operating systems: Bridging the gap between human and robot," in *Proceedings of the 2012 44th Southeastern Symposium on System Theory (SSST)*, pp. 99–104, 2012.

[190] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[191] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), p. 1805–1814, Curran Associates Inc., 2018.

[192] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[193] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[194] K. Judah, A. P. Fern, T. G. Dietterich, and P. Tadepalli, "Active imitation learning: Formal and practical reductions to i.i.d. learning," *Journal of Machine Learning Research*, vol. 15, no. 120, pp. 4105–4143, 2014.

[195] R. Memmesheimer, I. Kramer, V. Seib, and D. Paulus, "Simitate: A hybrid imitation learning benchmark," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5243–5249, 2019.

[196] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[197] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[198] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[199] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[200] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[201] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[202] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," 2018.

[203] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions," *arXiv preprint arXiv:1901.01753*, 2019.

[204] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[205] K. Jothimurugan, R. Alur, and O. Bastani, "A composable specification language for reinforcement learning tasks," *arXiv preprint arXiv:2008.09293*, 2020.

[206] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, 2018.

[207] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods," in *AAAI Conference on Artificial Intelligence*, 2018.

[208] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[209] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, 2019.

[210] H. Zhao, X. Zeng, T. Chen, Z. Liu, and J. Woodcock, "Learning safe neural network controllers with barrier certificates," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 177–185, Springer, 2020.

[211] A. Amini, W. Schwarting, G. Rosman, B. Araki, S. Karaman, and D. Rus, "Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing," in *IEEE International Conference on Intelligent Robots and Systems*, 2018.

[212] S. L. Herbert, S. Bansal, S. Ghosh, and C. J. Tomlin, "Reachability-based safety guarantees using efficient initializations," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4810–4816, IEEE, 2019.

[213] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. 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)*, pp. 1758–1765, IEEE, 2019.

[214] S. Bansal, A. Bajcsy, E. Ratner, A. D. Dragan, and C. J. Tomlin, "A hamilton-jacobi reachability-based framework for predicting and analyzing human motion for safe planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7149–7155, IEEE, 2020.

[215] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conference on Robot Learning*, pp. 420–429, PMLR, 2020.

[216] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, pp. 90–96, 2017.

[217] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Robotics Research* (N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, eds.), (Cham), pp. 75–84, Springer International Publishing, 2020.

[218] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, pp. 1–9, Curran Associates, Inc., 2015.

[219] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, vol. 6, pp. 3504–3508 vol.6, 1998.

[220] S. Jha, J. Rushby, and N. Shankar, "Model-centered assurance for autonomous systems," in *Computer Safety, Reliability, and Security* (A. Casimiro, F. Ortmeier, F. Bitsch, and P. Ferreira, eds.), (Cham), pp. 228–243, Springer International Publishing, 2020.

[221] T. X. T. Dang, *Verification and Synthesis of Hybrid Systems*. Theses, Institut National Polytechnique de Grenoble - INPG, Oct. 2000.

[222] X. Dai and A. Burns, "Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems," *Journal of Systems Architecture*, vol. 103, p. 101691, 2020.

[223] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu, "Rosrv: Runtime verification for robots," in *Runtime Verification* (B. Bonakdarpour and S. A. Smolka, eds.), (Cham), pp. 247–254, Springer International Publishing, 2014.

[224] F. Muratore, M. Gienger, and J. Peters, "Assessing transferability from simulation to reality for reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 4, pp. 1172–1183, 2021.

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

[226] V. S. Babu and M. Behl, "Ros f1/10 autonomous racecar simulator," October 2019.

[227] R. Rajamani, *Lateral Vehicle Dynamics*. Boston, MA: Springer US, 2012.

[228] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, *Case Study: Verifying the Safety of an Autonomous Racing Car with a Neural Network Controller*, ch. 28, pp. 1–7. New York, NY, USA: Association for Computing Machinery, 2020.

[229] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, p. 84–90, May 2017.

[230] N. Otterness, "The "disparity extender" algorithm, and f1/tenth," Apr 2019.

[231] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, May 2008.

[232] T. Fraichard and H. Asama, "Inevitable collision states. a step towards safer robots?," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 1, pp. 388–393 vol.1, 2003.

[233] J. Nilsson, J. Fredriksson, and A. C. Ödblom, "Verification of collision avoidance systems using reachability analysis," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 10676–10681, 2014. 19th IFAC World Congress.

[234] A. R. Ramapuram Matavalam, U. Vaidya, and V. Ajjarapu, "Data-driven approach for uncertainty propagation and reachability analysis in dynamical systems," in *2020 American Control Conference (ACC)*, pp. 3393–3398, 2020.

[235] Q. Lin, X. Chen, A. Khurana, and J. Dolan, "Reachflow: An online safety assurance framework for waypoint-following of self-driving cars," in *International Conference on Intelligent Robots and systems (IROS)*, IROS'2020, (Las Vegas, Nevada), pp. 6627 – 6632, IEEE, 2020.

[236] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li, "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *SIGBED Rev.*, vol. 8, p. 7–10, jun 2011.

[237] L. Bu, Q. Wang, X. Ren, S. Xing, and X. Li, "Scenario-based online reachability validation for cps fault prediction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2081–2094, 2020.

[238] T. Li, F. Tan, Q. Wang, L. Bu, J. Cao, and X. Liu, "From offline toward real time: A hybrid systems model checking and cps codesign approach for medical device plug-and-play collaborations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 642–652, 2014.

[239] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, (Melbourne,Australia), pp. 1517–1522, 2017.

[240] P. Holmes, S. Kousik, B. Zhang, D. Raz, C. Barbalata, M. Johnson-Roberson, and R. Vasudevan, "Reachable sets for safe, real-time manipulator trajectory design," 2020.

[241] K. Leung, E. Schmerling, M. Zhang, M. Chen, J. Talbot, J. C. Gerdes, and M. Pavone, "On infusing reachability-based safety assurance within probabilistic planning frameworks for human-robot vehicle interactions," *The International Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1326–1345, 2020.

[242] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia, "Drona: A framework for safe distributed mobile robotics," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, ICCPS '17, (New York, NY, USA), p. 239–248, Association for Computing Machinery, 2017.

[243] M. Bui, M. Lu, R. Hojabr, M. Chen, and A. Shriraman, "Real-Time Formal Verification of Autonomous Systems With An FPGA," *arXiv e-prints*, p. arXiv:2012.04011, Dec. 2020.

[244] P. Musau, N. Hamilton, D. Manzanas Lopez, P. Robinette, and T. T. Johnson, "An Empirical Analysis of the Use of Real-Time Reachability for the Safety Assurance of Autonomous Vehicles," *arXiv e-prints*, p. arXiv:2205.01419, May 2022.

[245] "Driverless cars are taking longer than we expected. here's why.," Jul 2019.

[246] S. Thrun, "Winning the darpa grand challenge," in *Machine Learning: ECML 2006* (J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, eds.), (Berlin, Heidelberg), pp. 4–4, Springer Berlin Heidelberg, 2006.

[247] A. Amini, W. Schwarting, G. Rosman, B. Araki, S. Karaman, and D. Rus, "Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 568–575, 2018.

[248] P. Ballester and R. M. Araujo, "On the performance of googlenet and alexnet applied to sketches," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, (Phoenix, Arizona), pp. 1124–1128, AAAI Press, 2016.

[249] K. Dunlap, M. Mote, K. Delsing, and K. L. Hobbs, "Run time assured reinforcement learning for safe satellite docking," in *2022 AIAA SciTech Forum*, pp. 1–20, 2022.

[250] C. Urban and A. Miné, "A review of formal methods applied to machine learning," *CoRR*, vol. abs/2104.02466, 2021.

[251] A. Ferrando, R. C. Cardoso, M. Fisher, D. Ancona, L. Franceschini, and V. Mascardi, "Rosmonitoring: A runtime verification framework for ros," in *Towards Autonomous Robotic Systems* (A. Mohammad, X. Dong, and M. Russo, eds.), (Cham), pp. 387–399, Springer International Publishing, 2020.

[252] H.-D. Tran, L. V. Nguyen, N. Hamilton, W. Xiang, and T. T. Johnson, "Reachability analysis for high-index linear differential algebraic equations," in *Formal Modeling and Analysis of Timed Systems*, (Berlin, Heidelberg), p. 160–177, Springer-Verlag, 2019.

[253] A. Akametalu, *A Learning-Based Approach to Safety for Uncertain Robotic Systems*. PhD thesis, EECS Department, University of California, Berkeley, May 2018.

[254] S. A. Seshia and D. Sadigh, "Towards verified artificial intelligence," *CoRR*, vol. abs/1606.08514, 2016.

[255] R. Orellana, M. Coronel, R. Carvajal, R. A. Delgado, P. Escárate, and J. C. Agüero, "On the uncertainty modelling for linear continuous-time systems utilising sampled data and gaussian mixture models∗∗this work was supported by anid/doctorado nacional/2017-21170804, anid-fondecyt grants 1211630 and 11201187, dpp at universidad técnica federico santa maría and the advanced center for electrical and electronic engineering, ac3e, basal project fb0008, anid, chile.," *IFAC-PapersOnLine*, vol. 54, no. 7, pp. 589–594, 2021. 19th IFAC Symposium on System Identification SYSID 2021.

[256] A. Doty, J. Doty, J. Camberos, and K. Yerkes, "Nonlinear uncertainty quantification, sensitivity analysis, and uncertainty propagation of a dynamic electrical circuit," in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, pp. 13110–13234, 2013.

[257] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," *Machine Learning*, vol. 110, no. 3, pp. 457–506, 2021.

[258] A. Puri and P. Varaiya, "Decidability of hybrid systems with rectangular differential inclusions," in *Computer Aided Verification* (D. L. Dill, ed.), (Berlin, Heidelberg), pp. 95–104, Springer Berlin Heidelberg, 1994.

[259] S. Z. Gonzalez, P. Collins, L. Geretti, D. Bresolin, and T. Villa, "Higher order method for differential inclusions," 2020.

[260] T. Dang and O. Maler, "Reachability analysis via face lifting," in *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control*, HSCC '98, (Berlin, Heidelberg), p. 96–109, Springer-Verlag, 1998.

[261] B. Sohlberg and E. Jacobsen, "Grey box modelling branches and experiences," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 11415–11420, 2008. 17th IFAC World Congress.

[262] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, p. 13–es, Dec. 2006.

[263] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Tech. Rep. CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[264] S. Devi, P. Malarvezhi, R. Dayana, and K. Vadivukkarasi, "A comprehensive survey on autonomous driving cars: A perspective view," *Wirel. Pers. Commun.*, vol. 114, pp. 2121–2133, 2020.

[265] H.-D. Tran, F. Cai, M. L. Diego, P. Musau, T. T. Johnson, and X. Koutsoukos, "Safety verification of cyber-physical systems with reinforcement learning control," *ACM Trans. Embed. Comput. Syst.*, vol. 18, Oct. 2019.

[266] T. Fraichard, "A short paper about motion safety," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1140–1145, 2007.

[267] F. P. Bernardo, "Model analysis and optimization under uncertainty using highly efficient integration techniques," in *26th European Symposium on Computer Aided Process Engineering* (Z. Kravanja and M. Bogataj, eds.), vol. 38 of *Computer Aided Chemical Engineering*, pp. 2151–2156, Elsevier, 2016.

[268] M. Althoff and D. Grebenyuk, "Implementation of interval arithmetic in cora 2016," in *ARCH16. 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems* (G. Frehse and M. Althoff, eds.), vol. 43 of *EPiC Series in Computing*, pp. 91–105, EasyChair, 2017.

[269] L. Ljung, Q. Zhang, P. Lindskog, and A. Juditski, "Estimation of grey box and black box models for non-linear circuit data," *IFAC Proceedings Volumes*, vol. 37, no. 13, pp. 399–404, 2004. 6th IFAC Symposium on Nonlinear Control Systems 2004 (NOLCOS 2004), Stuttgart, Germany, 1-3 September, 2004.

[270] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 1821–1830, 2021.

[271] J. Macfarlane and M. Stroila, "Addressing the uncertainties in autonomous driving," *SIGSPATIAL Special*, vol. 8, p. 35–40, dec 2016.

[272] P. Girão, A. Asvadi, P. Peixoto, and U. Nunes, "3d object tracking in driving environment: A short review and a benchmark dataset," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 7–12, 2016.

[273] G. Hartmann, Z. Shiller, and A. Azaria, "Autonomous head-to-head racing in the indy autonomous challenge simulation race," *CoRR*, vol. abs/2109.05455, 2021.

[274] T. Schoels, L. Palmieri, K. O. Arras, and M. Diehl, "An NMPC approach using convex inner approximations for online motion planning with guaranteed collision freedom," *CoRR*, vol. abs/1909.08267, 2019.

[275] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015.

[276] C. Jung, S. Lee, H. Seong, A. Finazzi, and D. H. Shim, "Game-theoretic model predictive control with data-driven identification of vehicle model for head-to-head autonomous racing," *CoRR*, vol. abs/2106.04094, 2021.

[277] J. Lorenzetti, M. Chen, B. Landry, and M. Pavone, "Reach-avoid games via mixed-integer second-order cone programming," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4409–4416, 2018.

[278] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "A nonlinear model predictive control formulation for obstacle avoidance in high-speed autonomous ground vehicles in unstructured environments," *Vehicle System Dynamics*, vol. 56, no. 6, pp. 853–882, 2018.

[279] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, p. 628–647, Jul 2014.

[280] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 6, pp. 2182–2189, 2018.

[281] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: a predictive control approach," 2019.

[282] L. Zhiyang and J. Tao, "Route planning based on improved artificial potential field method," in *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pp. 196–199, 2017.

[283] Y. Zeqing, L. Libing, T. Zhihong, and L. Weiling, "Application of adaptive genetic algorithm in flexible inspection path planning," in *2008 27th Chinese Control Conference*, pp. 75–80, 2008.

[284] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, 2011.

[285] S. H. A. Mohammad, M. A. Jeffril, and N. Sariff, "Mobile robot obstacle avoidance by using fuzzy logic technique," in *2013 IEEE 3rd International Conference on System Engineering and Technology*, pp. 331–335, 2013.

[286] H. Dong, C.-Y. Weng, C. Guo, H. Yu, and I.-M. Chen, "Real-time avoidance strategy of dynamic obstacles via half model-free detection and tracking with 2d lidar for mobile robots," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 4, pp. 2215–2225, 2021.

[287] Y. Zhang, Z. Liu, and L. Chang, "A new adaptive artificial potential field and rolling window method for mobile robot path planning," in *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 7144–7148, 2017.

[288] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[289] U. Rosolia, S. De Bruyne, and A. G. Alleyne, "Autonomous vehicle control: A nonconvex approach for obstacle avoidance," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 469–484, 2017.

[290] T. Mercy, W. Van Loock, and G. Pipeleers, "Real-time motion planning in the presence of moving obstacles," in *2016 European Control Conference (ECC)*, pp. 1586–1591, 2016.

[291] E. Scholte and M. E. Campbell, "Robust nonlinear model predictive control with partial state information," *IEEE Transactions on Control Systems Technology*, vol. 16, no. 4, pp. 636–651, 2008.

[292] M. KEIL and J. SNOEYINK, "On the time bound for convex decomposition of simple polygons," *International Journal of Computational Geometry & Applications*, vol. 12, no. 03, pp. 181–192, 2002.

[293] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control* (A. Garulli and A. Tesi, eds.), (London), pp. 207–226, Springer London, 1999.

[294] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2018.

[295] H. Seki, S. OOYAMA, and M. OGAWA, "Nonlinear model predictive control using successive linearization," *Transactions of the Society of Instrument and Control Engineers*, vol. 38, pp. 61–66, 01 2002.

[296] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.

[297] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Computer Aided Verification* (G. Gopalakrishnan and S. Qadeer, eds.), (Berlin, Heidelberg), pp. 379–395, Springer Berlin Heidelberg, 2011.

[298] A. Chutinan and B. H. Krogh, "Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations," in *Hybrid Systems: Computation and Control* (F. W. Vaandrager and J. H. van Schuppen, eds.), (Berlin, Heidelberg), pp. 76–90, Springer Berlin Heidelberg, 1999.

[299] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification* (N. Sharygina and H. Veith, eds.), (Berlin, Heidelberg), pp. 258–263, Springer Berlin Heidelberg, 2013.

[300] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[301] J. S. Chang and C. K. Yap, "A polynomial solution for the potato-peeling problem," *Discrete Comput. Geom.*, vol. 1, p. 155–182, dec 1986.

[302] D. H. DOUGLAS and T. K. PEUCKER, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[303] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm: "follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[304] M. J. D. Powell, "A view of algorithms for optimization without derivatives," tech. rep., Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, England, 2017.

[305] S. Lucia, A. Tatulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, p. 51–62, 03 2017.

[306] Q. Ge, Q. Sun, S. E. Li, S. Zheng, W. Wu, and X. Chen, "Numerically stable dynamic bicycle model for discrete-time control," in *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pp. 128–134, 2021.

[307] G. Marafioti, P. Liljebäck, and A. A. Transeth, "A study of nonlinear model predictive control (nmpc) for snake robot path following," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pp. 568–573, 2014.

[308] S. Lucia, A. Tătulea-Codrean, C. Schoppmeyer, and S. Engell, "An environment for the efficient testing and implementation of robust nmpc," in *2014 IEEE Conference on Control Applications (CCA)*, pp. 1843–1848, 2014.

[309] G. Garimella, M. Sheckells, and M. Kobilarov, "Robust obstacle avoidance for aerial platforms using adaptive model predictive control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5876–5882, 2017.

[310] S. Florman, "Approaches to verifying safety."

[311] R. Kneuper, "Limits of formal methods," *Form. Asp. Comput.*, vol. 9, p. 379–394, jul 1997.

[312] J. Bobek, *Hidden Fallacies in Formally Verified Systems*. PhD thesis, Computer Science Department, Georgia Institute of Technology, May 2020.

[313] S. Putter and A. Wijs, "Verifying a verifier: On the formal correctness of an lts transformation verification technique," in *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633*, (Berlin, Heidelberg), p. 383–400, Springer-Verlag, 2016.

[314] P. Nuzzo, *Compositional Design of Cyber-Physical Systems Using Contracts*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2015.

[315] J. Sztipanovits and G. Karsai, "Model-integrated computing," *Computer*, vol. 30, no. 4, pp. 110–111, 1997.

[316] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Comput. Surv.*, vol. 52, sep 2019.

[317] M. Törngren and U. Sellgren, *Complexity Challenges in Development of Cyber-Physical Systems*, pp. 478–503. Cham: Springer International Publishing, 2018.

[318] G. Karsai and J. Sztipanovits, "Model-integrated development of cyber-physical systems," in *Proceedings of the 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, SEUS '08, (Berlin, Heidelberg), p. 46–54, Springer-Verlag, 2008.

[319] K. H. Johansson, J. Lygeros, and S. Sastry, "Modeling of hybrid systems," in *Encyclopedia of Life Support Systems (EOLSS), Theme 6.43: Control Systems, Robotics and Automation*, UNESCO, 2004. Qc 20120228.

[320] I. The MathWorks, *Stateflow*. Natick, Massachusetts, United States, 2019.

[321] Z. Gu, S. Wang, J. C. Kim, and K. G. Shin, "Integrated modeling and analysis of automotive embedded control systems with real-time scheduling," in *SAE 2004 World Congress & Exhibition*, SAE International, mar 2004.

[322] A. Chatterjee and H. Reza, "Toward modeling and verification of uncertainty in cyber-physical systems," in *2020 IEEE International Conference on Electro Information Technology (EIT)*, pp. 568–576, 2020.

[323] J. B. Michael, G. W. Dinolt, and D. Drusinsky, "Open questions in formal methods," *Computer*, vol. 53, no. 5, pp. 81–84, 2020.

[324] C. Radojicic, C. Grimm, A. Jantsch, and M. Rathmair, "Towards verification of uncertain cyber-physical systems," in *Proceedings 3rd International Workshop on Symbolic and Numerical Methods for Reachability Analysis, SNR@ETAPS 2017, Uppsala, Sweden, 22nd April 2017* (E. Ábrahám and S. Bogomolov, eds.), vol. 247 of *EPTCS*, pp. 1–17, 2017.

[325] T. Yamaguchi, T. Kaga, A. Donzé, and S. A. Seshia, "Combining requirement mining, software model checking and simulation-based verification for industrial automotive systems," in *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 201–204, 2016.

[326] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyber–physical system integration," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, 2012.

[327] V. Paruthi, "Large-scale application of formal verification: From fiction to fact," in *Formal Methods in Computer Aided Design*, pp. 175–180, 2010.

[328] S. Ray, *Introduction*, pp. 1–5. Boston, MA: Springer US, 2010.

[329] Y. Dajsuren and M. v. den Brand, *Automotive Software Engineering: Past, Present, and Future*, pp. 3–8. Cham: Springer International Publishing, 2019.

[330] M. Chen, *High Dimensional Reachability Analysis: Addressing the Curse of Dimensionality in Formal Verification*. PhD thesis, University of California, Berkeley, USA, 2017.

[331] L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. J. Kochenderfer, "Toward Scalable Verification for Safety-Critical Deep Networks," *CoRR*, vol. abs/1801.05950, 2018.

[332] S. Bak, S. Bogomolov, and T. T. Johnson, "Hyst: A source transformation and translation tool for hybrid automaton models," in *1st International Workshop on Symbolic and Numerical Methods for Reachability Analysis (¡a href="http://snrworkshop.github.io/"¿SNR 2015¡/a¿), Co-Located with the 27th International Conference on Computer Aided Verification (¡a href="http://i-cav.org/2015/"¿CAV 2015¡/a¿)*, (San Francisco, California), July 2015.

[333] H. Garavel, M. H. t. Beek, and J. v. d. Pol, "The 2020 expert survey on formal methods," in *Formal Methods for Industrial Critical Systems* (M. H. ter Beek and D. Ničković, eds.), (Cham), pp. 3–69, Springer International Publishing, 2020.

[334] D. Guidotti, L. Pulina, and A. Tacchella, "pynever: A framework for learning and verification of neural networks," in *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings* (Z. Hou and V. Ganesh, eds.), vol. 12971 of *Lecture Notes in Computer Science*, pp. 357–363, Springer, 2021.

[335] M. Brundage, S. Avin, J. Wang, H. Belfield, G. Krueger, G. K. Hadfield, H. Khlaaf, J. Yang, H. Toner, R. Fong, T. Maharaj, P. W. Koh, S. Hooker, J. Leung, A. Trask, E. Bluemke, J. Lebensold, C. O'Keefe, M. Koren, T. Ryffel, J. B. Rubinovitz, T. Besiroglu, F. Carugati, J. Clark, P. Eckersley, S. de Haas, M. Johnson, B. Laurie, A. Ingerman, I. Krawczuk, A. Askell, R. Cammarota, A. Lohn, D. Krueger,

C. Stix, P. Henderson, L. Graham, C. Prunkl, B. Martin, E. Seger, N. Zilberman, S. Ó. hÉigeartaigh, F. Kroeger, G. Sastry, R. Kagan, A. Weller, B. Tse, E. Barnes, A. Dafoe, P. Scharre, A. Herbert-Voss, M. Rasser, S. Sodhani, C. Flynn, T. K. Gilbert, L. Dyer, S. Khan, Y. Bengio, and M. Anderljung, "Toward trustworthy AI development: Mechanisms for supporting verifiable claims," *CoRR*, vol. abs/2004.07213, 2020.

[336] O. Lahav and G. Katz, "Pruning and slicing neural networks using formal verification," in *Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021*, pp. 1–10, IEEE, 2021.

[337] X. Yang, T. Yamaguchi, H. Tran, B. Hoxha, T. T. Johnson, and D. V. Prokhorov, "Neural network repair with reachability analysis," *CoRR*, vol. abs/2108.04214, 2021.

[338] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, (New York, NY, USA), p. 588–603, Association for Computing Machinery, 2021.

[339] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. 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.

[340] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[341] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, and R. Vasudevan, "Driving in the matrix Can virtual worlds replace human-generated annotations for real world tasks?," *CoRR*, vol. abs/1610.01983, 2016.

[342] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *CoRR*, vol. abs/1406.2227, 2014.

[343] S. Ramakrishna, Z. Rahiminasab, A. Easwaran, and A. 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, 2020.

[344] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *CoRR*, vol. abs/1606.06565, 2016.

[345] F. Cai, *Out-of-Distribution Detection in Learning-Enabled Cyber-Physical Systems*. PhD thesis, ECE Department, Vanderbilt University, January 2022.

[346] D. Boursinos and X. D. Koutsoukos, "Improving prediction confidence in learning-enabled autonomous systems," in *Dynamic Data Driven Applications Systems - Third International Conference, DDDAS 2020, Boston, MA, USA, October 2-4, 2020, Proceedings* (F. Darema, E. Blasch, S. Ravela, and A. Aved, eds.), vol. 12312 of *Lecture Notes in Computer Science*, pp. 217–224, Springer, 2020.

[347] H. Karvonen, E. Heikkilä, and M. Wahlström, "Safety challenges of ai in autonomous systems design – solutions from human factors perspective emphasizing ai awareness," in *Engineering Psychology and Cognitive Ergonomics. Cognition and Design* (D. Harris and W.-C. Li, eds.), (Cham), pp. 147–160, Springer International Publishing, 2020.

[348] C. S. Pasareanu, D. Gopinath, and H. Yu, "Compositional verification for autonomous systems with deep learning components," *CoRR*, vol. abs/1810.08303, 2018.

[349] R. Ivanov, K. Jothimurugan, S. Hsu, S. Vaidya, R. Alur, and O. Bastani, "Compositional learning and verification of neural network controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 20, sep 2021.

[350] G. Brat, E. Denney, K. Farrell, D. Giannakopoulou, A. Jonsson, J. Frank, M. Boddy, T. Carpenter, T. Estlin, and M. Pivtoraiko, "A robust compositional architecture for autonomous systems," in *2006 IEEE Aerospace Conference*, pp. 8 pp.–, 2006.

[351] J.-P. A. Yaacoub, O. Salman, H. N. Noura, N. Kaaniche, A. Chehab, and M. Malli, "Cyber-physical systems security: Limitations, issues and future trends," *Microprocessors and Microsystems*, vol. 77, p. 103201, 2020.

[352] A. Kuwajerwala and C. Chen, "Backwards reachability: A tutorial."

[353] M.-s. Kim and H. S. Park, "Open platform for ubiquitous robotic services," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, (New York, NY, USA), p. 892–893, Association for Computing Machinery, 2012.

[354] I. Chen, B. MacDonald, B. Wünsche, G. Biggs, and T. Kotoku, "A simulation environment for openrtm-aist," in *2009 IEEE/SICE International Symposium on System Integration (SII)*, pp. 113–117, 2009.

[355] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, pp. 2766–2771, 2003.

[356] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: Insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, p. 100164, 2021.

[357] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 812–818, 2017.

[358] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer, "Formal verification of obstacle avoidance and navigation of ground robots," *Int. J. Rob. Res.*, vol. 36, p. 1312–1340, oct 2017.

[359] European Organization For Nuclear Research and OpenAIRE, "Zenodo," 2013.

[360] P. K. Panigrahi and S. K. Bisoy, "Localization strategies for autonomous mobile robots: A review," *Journal of King Saud University - Computer and Information Sciences*, 2021.

[361] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, November 2011.

[362] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2432–2437, 2005.

[363] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, 2016.

[364] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[365] Y. Jia, X. Yan, and Y. Xu, "A survey of simultaneous localization and mapping for robot," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1, pp. 857–861, 2019.

[366] E. B. Olson, "Real-time correlative scan matching," in *2009 IEEE International Conference on Robotics and Automation*, pp. 4387–4393, 2009.

[367] A. Censi, "An icp variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, pp. 19–25, 2008.

[368] A. Doucet and A. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, 01 2009.

[369] D. Fox, "Kld-sampling: Adaptive particle filters," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[370] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments: Part i," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1063–1069, 2008.

[371] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[372] L. Humphrey, E. Wolff, and U. Topcu, "Formal specification and synthesis of mission plans for unmanned aerial vehicles," *AAAI Spring Symposium Series*, 2014.

[373] L. Janson, T. Hu, and M. Pavone, "Safe motion planning in unknown environments: Optimality benchmarks and tractable policies," *CoRR*, vol. abs/1804.05804, 2018.

[374] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, 1993.

[375] M. Khatib, H. Jaouni, R. Chatila, and J. Laumond, "Dynamic path modification for car-like nonholonomic mobile robots," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, pp. 2920–2925 vol.4, 1997.

[376] M. Keller, F. Hoffmann, C. Hass, T. Bertram, and A. Seewald, "Planning of optimal collision avoidance trajectories with timed elastic bands," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9822–9827, 2014. 19th IFAC World Congress.

[377] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.

[378] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*, pp. 5054–5061, 2013.

[379] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano, "Control architecture design for autonomous vehicles," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 404–411, 2018.

[380] D. Simon, J. Löfberg, and T. Glad, "Reference tracking mpc using dynamic terminal set transformation," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2790–2795, 2014.

[381] J. Bravo, T. Alamo, and E. Camacho, "Robust mpc of constrained discrete-time nonlinear systems based on approximated reachable sets," *Automatica*, vol. 42, no. 10, pp. 1745–1751, 2006.

[382] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[383] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (USA), p. 2722–2730, IEEE Computer Society, 2015.

[384] K. J. Keesman, *System Identification: An Introduction*, pp. 1–13. London: Springer London, 2011.