On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers

Patrick Musau
Department of Electrical and Computer Engineering
Vanderbilt University
Nashville, TN, USA
patrick.musau@vanderbilt.edu

Nathaniel Hamilton
Department of Electrical and Computer Engineering
Vanderbilt University
Nashville, TN, USA
nathaniel.p.hamilton@vanderbilt.edu

Diego Manzanas Lopez
Department of Electrical and Computer Engineering
Vanderbilt University
Nashville, TN, USA
diego.manzanas.lopez@vanderbilt.edu

Preston Robinette
Department of Computer Science
Vanderbilt University
Nashville, TN, USA
preston.k.robinette@vanderbilt.edu

Taylor T. Johnson
Department of Computer Science
Vanderbilt University
Nashville, TN, USA
taylor.johnson@vanderbilt.edu

***Abstract*—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 paper, 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.**

***Index Terms*—formal verification, reachability analysis, imitation learning, deep reinforcement learning**

I. INTRODUCTION

The vision of a "driverless" future has riveted many technology enthusiasts, researchers, and corporations for decades [1]. 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 [1]. 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 [1]."

The two fundamental challenges widely regarded as limiting the arrival and widespread adoption of AVs are safety and reliability [2]. 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 [3]. 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 [3]. 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 [4] 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 [5].

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 [6]. 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* [7]. 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 [8]. 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 paper, we

extend the real-time reachability algorithm from [8], [9] 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 [10]. 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. [8], [9] 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.

In summary, the contributions of this paper are: (1) We modify the real-time reachability algorithm presented in [8] to handle static obstacles in a sound and real-time manner. (2) We implement a simplex control architecture that uses real-time reachability for online collision avoidance. (3) We show our method working with multiple machine learning controllers. (4) We demonstrate success using our method to safely navigate through obstacles the trained controllers have no prior experience with. (5) We evaluate the safety of machine learning components transferred to real-world hardware without additional training.

II. BACKGROUND: THE SIMPLEX ARCHITECTURE AND REAL-TIME REACHABILITY

A. Simplex Architecture

In the simplex architecture, the unverified component, or *complex controller*, is wrapped with a *safety controller* and a switching logic used to ensure safety [8]. 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 [9]. One must be able to clearly delineate safe states from unsafe states.

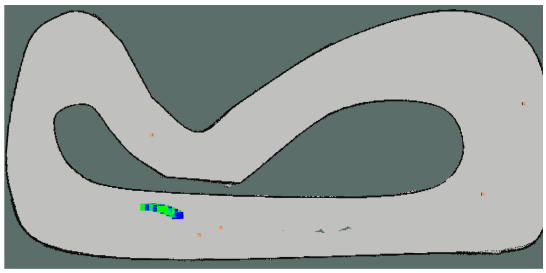

Fig. 1. Visualization of the set of reachable states using the current control action. For illustration purposes, we display only a subset of the hyper-rectangles (alternating green and blue boxes) making up the reachable set. This example corresponds to a safe scenario, as there is no intersection with obstacles or the racetrack walls (black). The orange squares represent the location of cones and their corresponding bounding box.

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) [11], or using a formal analysis hybrid-systems reachability technique [12]. In this paper, our simplex design requires computing the set of reachable states online through the use of a real-time reachability algorithm for short time horizons.

B. Real-Time Reachability

Reachability algorithms have traditionally been executed offline because they are computationally intensive endeavors [13], [14]. However, in [8], [9], Bak et al. and Johnson et al. presented a reachability algorithm, based on the seminal mixed face-lifting algorithm [15], 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 [9].

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 and a specified finite-time horizon. An example of this computation is shown in Fig. 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.

III. 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 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.

A. The F1/10 Autonomous Platform

The F1/10 platform of O'Kelly et al. [16] 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) [16]. 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 test-bed [16]. Additionally, in order to promote rapid prototyping and consider research questions around closing the simulation to reality gap, Varundev Suresh et al. designed a Gazebo-based simulation environment that includes a realistic model of the F1/10 platform and its sensor stack [17]. We utilize this simulation environment for a number of experiments and for training our controllers.

B. Vehicle Dynamics Model and System Identification

The physical dynamics of the F1/10 vehicle are modeled using a kinematic bicycle model [18], 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.[2] The model has four states: Euclidean positions x and y, linear velocity v, and heading θ. The dynamics are given by the following ODEs:

$$\dot{x} = v\cos(\theta), \qquad \dot{y} = v\sin(\theta),$$
$$\dot{\theta} = \frac{v}{l_f + l_r}\tan(\delta), \qquad \dot{v} = -c_a v + c_a c_m (u - c_h),$$

where v is the car's linear velocity, θ is the car's orientation, 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 [18]. 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 experiments with an average *Mean Squared Error* (MSE) of 0.003. 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. CONTROLLER CONSTRUCTION

Modern data-driven and machine learning methods have become increasingly scalable and efficient at dealing with complex problems in numerous contexts. In this section, we provide a high-level introduction to imitation learning and deep reinforcement learning and describe the construction of the controllers used within our simplex architecture.[3]

A. Imitation Learning

Imitation learning (IL) seeks to reproduce the behavior of a human or domain expert on a given task [19]. 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 [19]. 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 [19]. In this work, we utilize imitation learning to train two neural network controllers to produce steering angles from different sensor inputs.

Vision-Based Navigation (VBN)

Since the seminal work of Krizhevsky et al. [20] in the ImageNet Large Scale Recognition Challenge, *Convolutional Neural Networks* (CNNs) have revolutionized the field of computer vision. Within the context of autonomous vehicles,

[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 $20Hz$ control period.

[2] The kinematic bicycle model typically tracks well under $5m/s$ [18]

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

CNNs have demonstrated efficacy for driving tasks such as lane following, path planning, and control, simultaneously, by computing steering commands directly from images [10].

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 Fig. 1.

LiDAR Behavior Cloning (LBC)

The second network considered for imitation learning was 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. 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 VBN regime.

B. Reinforcement Learning

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 [21]. 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 [21].

In this paper, we used two well-known state-of-the-art reinforcement learning algorithms, an off-policy DRL algorithm known as *Soft-Actor-Critic* (SAC), [22], and an on-policy RL algorithm, known as *Augmented Random Search* (ARS), [23]. In line with the imitation learning experiments, the agents were trained on the racetrack shown in, Fig. 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.

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 [22]. In this work, the SAC controller was trained using the same architecture as the LBC controller described in Section IV-A.

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 [23]. Their simple method was able to match state-of-the-art sample efficiency on the benchmark MuJoCo locomotion tasks,[4] 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 $\pm 34°$. However, the input, s, consists of 271 LiDAR range values, clipped between $[0m, 10m]$, collected from between $\pm 90°$ from forward.

V. 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* [9] where snapshots of the set of reachable states are enumerated at successive points in time. To formalize this concept, we define the reachable set below.

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 state, $\boldsymbol{x}_0 = \boldsymbol{x}(0)$, and input, $\boldsymbol{u}_0 = \boldsymbol{u}(0)$, 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 $[0, T_{reach}]$ is:*

$$R_{[0,T_{reach}]} = \left\{ \psi(\boldsymbol{x}_0, \boldsymbol{u}_0, t) \middle| \boldsymbol{x}_0 \in \chi_0, \boldsymbol{u}_0 \in U, t \in [0, T_{reach}] \right\}, \tag{1}$$

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

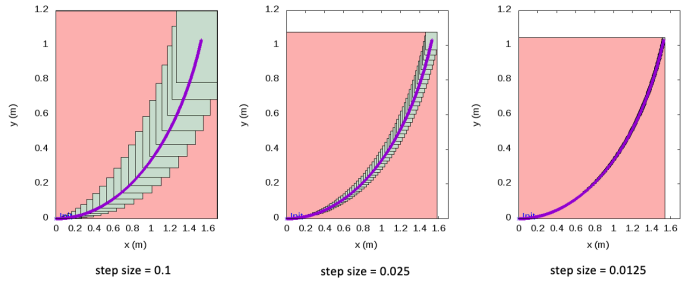

Fig. 2. The real-time reachability algorithm always returns an over-approximation of the reachable set of states. The over-approximation error decreases with successive iterations, provided that there is enough runtime for re-computations. The above images demonstrate this aspect by simulating a left-hand 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.

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 $\boldsymbol{u}_0$.[5]

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 [13]–[15]. The algorithm utilized in this work utilizes n-dimensional hyper-rectangles ("boxes") as the set representation to generate reachtubes [9]. 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 [8].

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 [8], 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 Fig. 2. We refer readers to the following papers for an in depth treatment of these procedures [8], [9], [15].

VI. 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 $R_{[0,T_{reach}]} \cap \Lambda = \emptyset$.*

[5]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.

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. These representations are then converted into their hyper-rectangle formulations that make up Λ. Fig. 1 provides a visualization of the obstacles, and Fig. 4 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 deadlines, 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: $k = T_{reach}/h$.[6] 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 $O(2^k)$.[7] Therefore, we can

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

[7]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.

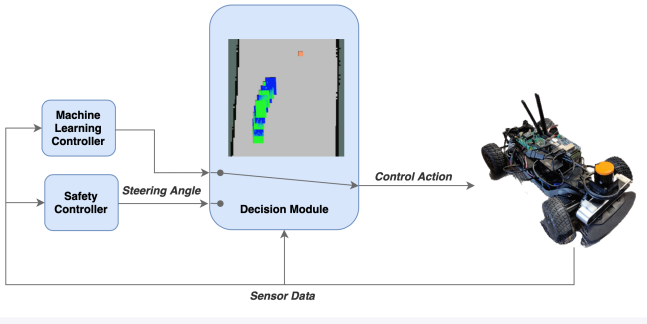

Fig. 3. Overview of the Simplex architecture deployed on the F1/10 system described in Section VII. The switching logic consists of monitoring the intersection between the F1/10 reachable set and the positions of static obstacles within the environment.

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 or not the system has entered an unsafe scenario, the safety checking is done during the computation with intermediate hyper-rectangles as outlined in [8]. 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.

VII. ROS SIMPLEX ARCHITECTURE

Our simplex architecture for the F1/10 is designed using ROS [24], and an overview of the design is shown in Fig. 3.

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.[8]

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.

[8]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**

In the traditional simplex architecture, both the decision module and the safety controller must be verified for the system to be verified as correct [8]. 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 [18]. 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 [25]. It was primarily selected due to its robust collision avoidance ability and simplicity.

VIII. 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 [26]. 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) [26] in Table II. 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).

A. 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 Fig. 1 provides a snapshot of this setup. Utilizing the control architecture discussed in Section VII, 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 Table I, 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.[9]

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.[10] 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.

B. 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 [21]. 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 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 Fig. 4.[11] (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.

TABLE I
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

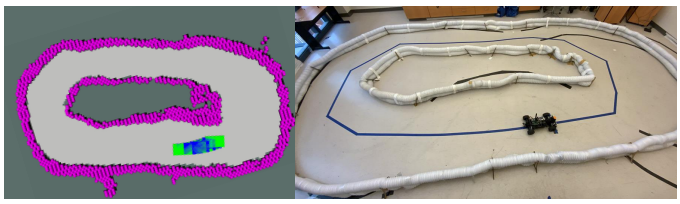

Fig. 4. Visualization of the hardware experiments on the F1/10 Platform, the magenta points are the point-discretization of the wall boundaries (not all points are visualized). Videos of the experiments can be found at **https://github.com/pmusau17/F1TenthHardware**.

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

[10]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 $T_{reach} = 1.0s$.

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

TABLE II
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

We ran a total of 80 experiments, 5 for each controller, with the same structure as the simulation episodes. A video of the experiments is available online.[12] 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 Table II. 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 II. 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 [9].

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

IX. 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.

A. 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.

B. Challenges Moving from Simulation to the Real World

Moving from simulation to the real-world hardware platform saw a significant increase in safety controller usage. This increased usage is a direct result of *sim2real* challenges in ML. Because the real world is inherently more noisy and more complex than the simulation environments that ML 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 [21]. However, the main challenge that we faced in the transition from simulation to the real world was the size of the track in our laboratory setup.

The track shown in Fig. 1 is much larger than the real world track we tested on, shown in Fig. 4. At the narrowest point, the simulated track is $2m$ wide. In contrast, the widest part of our real-world track is slightly over $2m$. 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 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. 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 controller, it is not clear how to generate camera images based on the state of the system to provide a meaningful and useful reachable set.

C. 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 [27]. 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 [28].

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 III, show that our model is reasonably accurate, in practice the implementation 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 [29]. 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.

X. RELATED WORK

The simplex architecture has been used widely in the research literature to provide guarantees for systems with unverified components. Examples include aerospace systems, fleets of remote controlled cars, industrial embedded infrastructure, and distributed mobile robotics applications [30]. Most similar to this work in [30], 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. However, 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.

Closely related to simplex techniques are other runtime assurance (RTA) methods and runtime verification (RV) methods. RTA techniques are tasked with ensuring the safe operation of systems with untrusted components, while runtime verification techniques monitor a system against presupposed formal properties at runtime [31], [32]. The distinction here is that while runtime assurance techniques may often utilize verification results, they may also employ statistical techniques such as anomaly detection or simulation based strategies [33].

Finally, in recent years, researchers have begun to integrate traditionally non-real-time 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 [34], as well as Hamilton-Jacobi reachability (HJR) techniques that can deal with dynamical systems with general nonlinear dynamics in uncertain environments [35].

XI. CONCLUSIONS AND FUTURE WORK

This paper presents a simplex architecture for the safety assurance of a 1/10 scale open source autonomous vehicle platform. The approach relies on a real-time reachability regime that is used to provide safety guarantees and detect potential unsafe scenarios during operation. This approach 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 reachset generation, 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.

ACKNOWLEDGEMENT

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.

REFERENCES

[1] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, pp. 90–96, 2017.

[2] 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: Springer International Publishing, 2020, pp. 75–84.

[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[4] 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," ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 2503–2511.

[5] 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.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. San Diego, CA: OpenReview, 2015.

[7] 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, 1998, pp. 3504–3508 vol.6.

[8] 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: IEEE, 2014, pp. 138–148.

[9] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," vol. 15, no. 2, Feb. 2016.

[10] 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: IEEE, 2018, pp. 4701 – 4708.

[11] 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)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-99-TR-020, 2000.

[12] 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: IEEE, 2009, pp. 99–107.

[13] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *2012 IEEE 33rd Real-Time Systems Symposium*, 2012, pp. 183–192.

[14] M. Althoff, "An introduction to cora 2015," in *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 34. Berlin, Germany: EasyChair, 2015, pp. 120–151.

[15] T. X. T. Dang, "Verification and Synthesis of Hybrid Systems," Theses, Institut National Polytechnique de Grenoble - INPG, Oct. 2000.

[16] 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.

[17] V. S. Babu and M. Behl, "Ros f1/10 autonomous racecar simulator," October 2019.

[18] 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," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '20. New York, NY, USA: Association for Computing Machinery, 2020.

[19] A. Hussein, M. Gaber, E. Elyan, and C. Jayne, "Imitation learning," *ACM Computing Surveys (CSUR)*, vol. 50, pp. 1 – 35, 2017.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017.

[21] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[22] 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.

[23] 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*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 1805–1814.

[24] 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)*, 2012, pp. 99–104.

[25] N. Otterness, "The "disparity extender" algorithm, and f1/tenth," Apr 2019.

[26] 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, no. 3, May 2008.

[27] 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, 2003, pp. 388–393 vol.1.

[28] J. Nilsson, J. Fredriksson, and A. C. Ödblom, "Verification of collision avoidance systems using reachability analysis," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 10 676–10 681, 2014, 19th IFAC World Congress.

[29] 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)*, 2020, pp. 3393–3398.

[30] 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: Springer International Publishing, 2019, pp. 261–277.

[31] 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: Springer International Publishing, 2018, pp. 333–348.

[32] 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: IEEE, 2014, pp. 1424 – 1431.

[33] H. D. Tran, L. V. Nguyen, N. Hamilton, W. Xiang, and T. T. Johnson, "Reachability analysis for high-index linear differential algebraic equations," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11750 LNCS, 2019.

[34] 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.

[35] 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.