Runtime Assurance for Autonomous Driving with Neural Reachability

Sergiy Bogomolov[1], Abdelrahman Hekal[1], Bardh Hoxha[2] and Tomoya Yamaguchi[2]

Abstract— **The development of safe autonomous vehicles remains a challenge. Modern autonomous driving controllers are able to operate safely in most operating conditions, however, rare conditions may cause undesired behaviors. To alleviate this issue, a complex controller may be paired with a simpler, yet verified, controller in a simplex architecture. To switch between the two, we take inspiration from formal methods, namely reachability analysis, to develop an online method called neural reachability that estimates the reachable sets of the system. The method predicts the possible states of the system under uncertain initial states and disturbances in control. The approach utilizes deep neural networks for conservative approximation of reachable sets in bounded time. If an intersection of the reachable set with an unsafe set is detected, a simple verified controller is utilized. We show how a deep neural network is trained using existing reachability analysis algorithms and demonstrate how neural reachability may be utilized in a simplex architecture. Also, we provide probabilistic guarantees based on statistical model checking approaches. Finally, the approach is evaluated as part of a resilient safety architecture for autonomous vehicles in a simulated environment with several maneuvers. Our evaluation demonstrates that reachability analysis can be done within a fraction of a second and outperforms traditional nonlinear reachability tools by two orders of magnitude.**

I. Introduction

The development of safe Autonomous Vehicles (AV) remains a fundamental challenge. As autonomous systems see use in a variety of safety-critical applications such as autonomous driving, there is a need for methods to ensure safety and correctness of their behavior. While sophisticated controllers have been developed to handle day-to-day operation, there are still rarely-occurring scenarios in which these systems fail to meet safety requirements [1]. Formal methods and reachability analysis [2] have shown to be useful in modeling and development of AV components at design time [3], [4]. However, the ability to completely guarantee correctness of the entire system remains a challenge. Therefore, in addition to verification at design time, run time assurance frameworks are necessary to ensure that these systems behave correctly.

Several frameworks have been proposed in the past [5], [6] that combine an advanced, unverified controller in conjunction with a simple, verified controller. A decision module maintains the safety of the system by forecasting the system behavior and switching to the safe controller once the advanced controller approaches unsafe states. In [7], the decision module is developed using a machine learning approach that considers dynamic feasibility, legality, and collision likelihood to switch between an advanced to a safe controller. In this work, we draw on formal methods, namely reachability analysis to present a complementary framework for training Deep Neural Networks (DNNs) for estimation of reachable states of the system.

The reachability problem is defined as follows: given a set of initial states, compute all possible states starting from the initial region. If there exists any intersection with unsafe states, the system is deemed unsafe [8]. For example, an autonomous robot is deemed safe if for all possible future behaviors, it does not collide with any static or dynamic obstacles. Reachability analysis typically obtains such information by iteratively computing the conservative overapproximation of the reachable region [9]–[12]. Despite advances in reachability algorithms, for complex systems with high-order dynamics, utilization in a run time setting remains a challenge. This calls for the development of faster online methods that would enable analysis in real-time.

In this work, we propose an online neural reachability algorithm based on DNNs for conservative approximation of the reachable states of the system in bounded time. The trained DNN needs to satisfy conservativeness bounds and enable real-time computation. Neural networks boast several properties that provide particular allure for their deployment for this purpose. In particular, they run in constant time and space, making them suitable for online computations. We utilize template polyhedra [13] for set representation of reachable sets where a neural network is utilized for the computation of each of its vector coefficients. The training data for the DNN is generated using offline reachability tools such as Flow* [14]. We evaluate our reachability approach by employing it in a closed-loop control setting.

In particular, we consider several automotive maneuvers and show that our neural reachability method can be used as the decision module of a simplex architecture to provide a safe fallback mechanism from a complex to a safe controller. Due to the approximate nature of neural networks, we also provide probabilistic guarantees on the validity of the computed reachset.

The paper is organized as follows. We first introduce related works in Sec. II. Then in Sec. III we define notation and definitions. In Sec. IV we present the neural network training process for computation of reachable sets and provide an approach for the development of statistical guarantees on the validity of the computed reachset. In Sec. V we demonstrate the performance of our method on an online monitoring

[1]Sergiy Bogomolov and Abdelrahman Hekal are with the School of Computing, Newcastle University, Newcastle Upon Tyne, UK {sergiy.bogomolov, a.waleed-elsayed-aly-hekal}@newcastle.ac.uk

[2]Bardh Hoxha and Tomoya Yamaguchi are with TRINA in Toyota Motor North America R&D, Ann Arbor, MI, 48187, USA {bardh.hoxha,tomoya.yamaguchi}@toyota.com

application over a car-like model, followed by conclusion and a discussion on future work in Sec. VI.

II. Related Work

Recently, researchers took to identifying scalable and robust methods for online verification and validation of autonomous mobile systems. One common approach is by solving the time-expensive reachability problem in real-time [15]. For example, Althoff et al [16] developed an approach to verify specific autonomous maneuvers. However, their runtime can prove too costly for general cases. In [5], [6], the authors propose a real-time reachability technique that iteratively computes and refines the reachable set. This technique bounds the runtime for the analysis and favors quick results over accuracy.

Alternatively, precomputation methods have been proposed to tackle the time complexity of the problem. These entail the use of tables [17], stochastic reachability analysis approaches [18] [19], and the combining of pre-computed reachability sets with the posterior probabilities evaluated by Bayesian estimation [20].

Furthermore, advances in artificial intelligence and machine learning through several techniques such as deep neural networks have enabled their employment in validation of such systems. For instance, Djeridane et al [21] train a feed-forward neural network to compute viability sets for a two-dimensional nonlinear continuous system. In [22], a probabilistic approach is considered where a neural network is trained to estimate the density of the reachable set. Alternatively, Rubies-Royo et al [23] present a neural network classifier to approximate the optimal controller in Hamilton-Jacobi reachability. for control-affine system. Furthermore, Phan et al [24] leverage neural networks to solve the state classification problem for hybrid systems. An extension of this approach by Bortolussi et al [25] provides estimates of the predictive uncertainty. The last two approaches – by training the neural network to output a binary classification of each state of a hybrid system – provide only partial information about the solution of the reachability problem. Our paper goes beyond this and employs neural networks to learn the shape of a reachable set of a system.

III. Preliminaries

Let $\mathbb{R}$ be the set of real numbers, $\mathbb{R}_+$ is the set of non-negative real numbers. Also, $\mathbb{N}$ is the set of natural numbers including 0. Given two sets A and B, B^A denotes the set of all functions from A to B. That is, for any $f \in B^A$ we have $f : A \to B$. Let $T \in \mathbb{R}_+$ denote the bounded simulation time.

We take a general approach to modeling autonomous mobile systems. We fix $R = [0, T] \subseteq \mathbb{R}_+$. We view a system Σ as a mapping from initial conditions $\mathcal{R}_0 \subseteq \mathcal{X}$, control input signals $\mathbf{U} \subseteq U^R$, and input disturbance $\mathbf{W} \subseteq \mathcal{W}^R$ to output signals $\mathcal{Y}^R$. Here, U is a compact set of input values (input space) at each point in time, $\mathcal{W}$ is the compact set of input disturbance which is bounded by an interval $[\underline{w_i}, \overline{w_i}]$ for each dimension i, and $\mathcal{Y}$ is the set of output values (output space). A bounded-time execution of a system may be viewed as a function $\Delta_\Sigma : \mathcal{R}_0 \times \mathbf{U} \times \mathbf{W} \times \mathbb{R}_+ \to \mathcal{Y}^R$, which takes as an input an initial state $r_0 \in \mathcal{R}_0$. a control input signal $u \in \mathbf{U}$, an input disturbance $w \in \mathbf{W}$, and bounded time $t \in \mathbb{R}_+$. The system generates a unique trajectory $\eta : \mathbb{R}_+ \to \mathcal{Y}$. That is, the system is deterministic.

A. Reachability Analysis

The reachability problem is defined as follows: given a set of initial states, compute all possible states starting from the initial region. Formally, we define the reachable set of the system at time $t \in \mathbb{R}_+$, from a set of initial conditions $\mathcal{R}_0$ as

$$\begin{aligned} Reach_t(\mathcal{R}_0) = \{\eta(t) \mid r_0 \in \mathcal{R}_0, u \in \mathbf{U}, \\ w \in \mathbf{W}, \eta = \Delta_\Sigma(r_0, u, w, t)\} \end{aligned} \quad (1)$$

The reachable set for a time interval $I = [T_{start}, T_{end}]$ is defined as

$$Reach_I(\mathcal{R}_0) = \bigcup_{t \in I} Reach_t(\mathcal{R}_0) \quad (2)$$

B. Recurrent Neural Networks

A recurrent neural network (RNN) is a function that receives time-series (sequential) data $x(t)$ as inputs, where $t \in \{0, 1, ...T\}$ and generates an output sequence in the output space $Y \subseteq \mathbb{R}^n$. The output of a RNN at time t is a function over the input at time t and the output of the hidden state at time $t - 1$. Specifically:

$$a(t) = W_{hh} \cdot h(t-1) + W_{hx} \cdot x(t) + b_h \quad (3)$$

$$h(t) = \theta(a(t)) \quad (4)$$

$$y(t) = W_{hy} \cdot h(t) + b_y \quad (5)$$

where W_{hh}, W_{hx} are weight matrices, b_h, b_y are bias vectors, and θ is an activation function. This structure of hidden states functions as a memory of the network and current output is conditioned on its previous state. This enables RNNs to capture complex signals that span over multiple time periods.

C. Minkowski Sum

Minkowski sums are used to compute the vector sum of each pair of points in convex shapes. As such, they enable the summation of convex polygons. The Minkowski sum of two sets A and B is defined as a set with the sum of all elements from A and all elements of B, such that:

$$\mathcal{X} \oplus \mathcal{Y} := \{x + y : x \in \mathcal{X} \ and \ y \in \mathcal{Y}\} \quad (6)$$

IV. Neural Reachability

In this section, we introduce the notion of neural reachability for approximating the reachable states of deterministic dynamical systems in bounded time.

Definition 1 (Neural Reachability): Given a system Σ, set of initial conditions $\mathbf{U}$, a disturbance set $\mathbf{W}$, and bounded time interval $I = [T_{start}, T_{end}]$ neural reachability estimates the reachable set of the system $Reach_I(\mathcal{R}_0)$ through a DNN.

Algorithm 1 Learning DNN for Reachability Analysis

```
Input: Template directions $m$; disturbance set $\mathcal{W}$; time horizon $T$; time step $\Delta t$; training data sets $\mathcal{D}_i = \{\mathcal{R}_0, u, k, b_i\}$ for $i = 1 : m$; data set length $Len$; desired underapprox. level $\epsilon$, state space $\mathcal{X}$; loss function $L$
Output: DNN $\mathcal{N}$
Function TRAIN($u, \mathcal{X}, \mathcal{W}, T, \Delta t$)
 1: for $k = 1 : Len$ do
 2:     $u$, $\mathcal{R}_0 \leftarrow$ RAND($U$), RAND($\mathcal{X}$)   # generate random control $u$, and initial set $\mathcal{R}_0$ from the state space
 3:     $\vec{b} \leftarrow$ COMPUTEREACH($u, \mathcal{R}_0, \mathcal{W}, T, \Delta t$)   # compute template polyhedron coefficient vector $\vec{b}$ of size $m$
 4:     for $i = 1 : m$ do
 5:         $\mathcal{D}_i \leftarrow \mathcal{D}_i \bigcup \{\mathcal{R}_0, u, k, b_i\}$   # append to dataset
 6:     end for
 7: end for
 8: $\mathcal{N} \leftarrow$ GRAD($\mathcal{D}, L$)   # Gradient descent with biased or MSE loss function
 9: if RETRAIN then
10:     $N \leftarrow$ RETRAIN($\mathcal{N}$)   # Reduce underapproximations
11: end if
12: return $\mathcal{N}$
Function RETRAIN($\mathcal{N}$)
 1: $u_{frac} \leftarrow$ fraction of underapproximate training data
 2: while $u_{frac} > \epsilon$ do
 3:     $\mathcal{D}_{retrain} \leftarrow$ GETUNDERAPPROXIMATIONS($\mathcal{D}, \mathcal{N}$)
 4:     Retrain $\mathcal{N}$ with $\mathcal{D}_{retrain}$
 5: end while
 6: return $\mathcal{N}$
Function GETUNDERAPPROXIMATIONS($\mathcal{D}, \mathcal{N}$)
 1: Forward propagate learned model $\mathcal{N}$ on dataset $\mathcal{D}$ to obtain $\{b_i^{pred}\}_{k=1}^{L}$
 2: for $b_i^{pred} \in \{b_i^{pred}\}_{k=1}^{L}$ do
 3:     if $b_i^{pred} < b_i$ then
 4:         $\mathcal{D}_{retrain} \leftarrow \mathcal{D}_{retrain} \bigcup \{\mathcal{R}_0, u, k, b_i\}$
 5:     end if
 6: end for
 7: return $\mathcal{D}_{retrain}$
```

A. Learning Reachable Sets

One popular approximation algorithm for reachability analysis is based on set representation using template polyhedra [13], with predefined normal vectors.

Definition 2 (Template Polyhedra [13]): Given a set $\mathcal{D} = \{\ell_1, ...\ell_m\}$ of vectors in $\mathbb{R}^n$ called template directions, a template polyhedron $\mathcal{P}_D \subseteq \mathbb{R}^n$ is a polyhedron where there exists coefficients $\{b_1, ..., b_m\} \in \mathbb{R}$ such that $\mathcal{P}_D = \{x \in \mathbb{R} \mid \bigwedge_{\ell_i \in \mathcal{D}} \ell_i \cdot x \leq b_i\}$.

A template polyhedron enables us to represent sets by defining the vector of coefficients b_i. We utilize neural networks to predict the coefficient b_i for each predefined direction ℓ_i. The neural network inputs are the initial set $\mathcal{R}_0$, represented with box directions using support functions; the

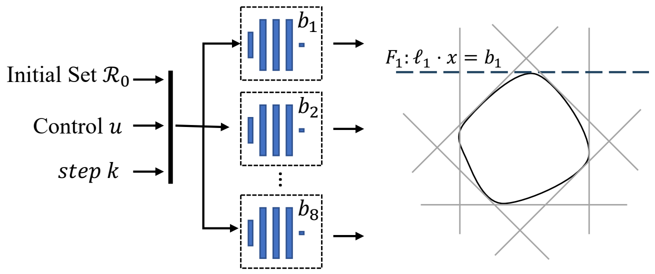

Fig. 1: Given the state x of the system, a control input u and a step k for a given time step Δt, we utilize eight recurrent neural networks to compute the coefficients $b_1...b_8$ for the template polyhedron with octagonal directions to approximate the reachable set of the system.

control actions; and the step. It then estimates a coefficient b_i representing the support value for a predefined support direction and given step.

We combine the outputs b_i^{pred} from n neural networks to form a fixed direction polyhedral reach image with n template directions, for a given time interval as shown in Fig. 1. The number of template directions of the polyhedron is a design choice which depends on the application. A higher number enables a tighter overapproximation of the reachable set, but increases the number of neural networks required for computation. We note that each neural network can be processed in parallel.

The step k, another input to the neural network, denotes the time interval we are interested in. In particular, the neural network computes the set $Reach_{[(k-1)\Delta t; k\Delta t]}(R_0)$ when provided with step k as input. Here, we assume that the time frame $[0; T]$ is divided into $T/\Delta t$ partitions of the size Δt.

The training process for the neural network is shown in function TRAIN (see Alg. 1). The training data is generated by using a reachability analysis tool, which performs analysis of the system and outputs reachable sets projected on the desired dimensions. These actions are performed by function COMPUTEREACH in Alg. 1 (line 3). We assume that this function generates the vector of coefficients b_i for each predefined direction ℓ_i. As illustrated in Fig. 1, for each octagonal direction, an individual neural network is trained to learn a coefficient b_i.

Once the training data is generated, we utilize an off-the-shelf gradient descent algorithm from the PyTorch library [26] which employs a mean square error (MSE) loss function to train the neural network as shown in Alg. 1 (line 8).

The resulting neural network comprises of three hidden layers of 200 neurons each. We employ recurrent layers to best capture the properties of reachability analysis. We also append a feed-forward layer to output the coefficient values.

B. Minimizing underapproximations

Due to the approximative nature of neural networks, the output reachable sets are often inexact. This means the sets may be under or over approximative, with the former being more problematic. Under-approximation means that unsafe states are identified as safe which in turn can lead to disastrous consequences. Thus, to ensure safety, under-approximations of the reachable set must be avoided, whilst maintaining close over-approximations for accurate results. For this purpose, we consider the following methods:

Bloating. A straightforward method where the output is bloated by a conservative, empirical factor. The simplicity of the technique is alluring, and the empirical bloating factor can be tuned for different requirements.
Biased loss function. This technique uses a biased loss function that favors over-approximation while training the neural network. Technically this idea is implemented by penalizing the network more for under-approximations. We use biased loss techniques in functions TRAIN (line 8) and RETRAIN (line 4).
Iterative retraining. In a post-training step, under-approximations that are found are used to iteratively retrain the neural network with a biased loss function. This refinement of the model helps make the output reachset more conservative and reduce the number of under-approximations. This approach is described in function RETRAIN in Alg. 1.

C. Statistical Guarantees

As the results produced by neural networks are inherently approximative, we conduct hypothesis tests with unseen, randomly generated test data to provide statistical guarantees. These tests are performed on the trained neural network after techniques for reducing underapproximations are employed. Motivated by statistical model checking approaches [24], [27], we adapt sequential probability ratio tests, and namely follow Wald's sequential probability ratio test (SPRT). The main advantage of these tests is they do not require a prescribed number of tests but rather accept a hypothesis once enough evidence is recognized. Thus, we construct tests in the form of hypothesis testing that verify that the predicted support values for the reachset meet a chosen level of accuracy, $\mathcal{P}_A \geq \theta_A$ and fall below a prescribed level of underapproximations, $\mathcal{P}_U \leq \theta_U$. These guarantees are accurate to confidence intervals (α, β), quantifying the strength of the test. To ensure feasibility of tests, we must relax the hypotheses by a small indifference region, δ. Thus, we test $H_0 : \mathcal{P}_A \geq \theta_x + \delta$ against $H_1 : \mathcal{P}_A \leq \theta_x - \delta$.

V. CASE STUDY

In this section, we apply the methods developed in the paper to a vehicle model [28]. Specifically, we demonstrate the application of an online neural reachability method based on neural networks to predict potentially undesired situations. This decision algorithm is employed as an online monitoring system that aims to verify the safety of a control system. Figure 2 shows an overview of our framework.

In section V-F we then extend our evaluation by incorporating our decision algorithm as part of a resilient simplex architecture [29].

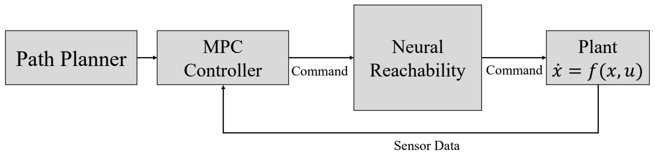

Fig. 2: *Overview of developed framework with neural network reachability deployed for online monitoring of control system.*

A. Vehicle Model

The ego vehicle dynamics are defined as follows:

$$\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \cdot cos(\theta) \\ v \cdot sin(\theta) \\ v \cdot tan(u_\delta)/L \\ u_a \end{bmatrix} \tag{7}$$

We consider the car-like model where the dynamics are of the center of rear axle. The state $x \in \mathbb{R}$ consists of vehicle location (x_1, x_2), heading angle θ and vehicle speed v. The inputs to the system are the steering angle u_δ and acceleration u_a.

We consider a system with dynamics uncertainty in the range $[-0.1, 0.1]$. This is used to account for disturbances in the position and orientation during the vehicle's motion. These can often arise due to unaccounted slip, wind resistance, etc. Thus, the system's safety must be verified while taking into account these uncertainties.

We present the results of our empirical evaluation. To evaluate our neural reachability algorithm in the online setting, we ran our framework on a MacBook Pro laptop with 16GB RAM and an eight core Apple M1 CPU @3.20GHz processor.

B. MPC Control

We deploy linear model predictive control (MPC) [30] to provide a sequence of actions for the vehicle. In our implementation, we adapt the MPC controller from a pre-existing library [31].

MPC is an advanced control technique that aims to find control actions for a time horizon while satisfying problem constraints. The controller outputs a sequence of commands that are found to be optimal for the car to follow in a specified prediction horizon. The number of commands depends on the prediction horizon and the sample time, where a command is assigned for each step k defined by the sample time. We deploy an MPC controller with a prediction horizon of $2s$ and a sample time of $0.4s$. Thus, each call of the controller returns five sequential commands for each input. Since our system has two control inputs, there are ten inputs to the neural network in total.

C. Neural Networks for Reachable Set Computation

We train neural networks to learn the reachable sets for the center of the rear axle of the vehicle driven by the advanced MPC controller. The system is modeled as a hybrid system with five discrete jumps that represent the change in control inputs.

We generate training data by running Flow* [14] reachability analysis with different MPC control actions and initial states and project the produced reachable sets on position variables. The control actions are bound by physical and scenario-based constraints on the vehicle such as maximum speed and acceleration. We generate 100,000 data points by randomly sampling of inputs in the valid state space. The average runtime of Flow* for data generation is $5.87s$, which showcases the need for our approach for online verification.

In order to further reduce the state space for training data, we identify invariant variables that do not impact the vehicle behavior. Namely, the vehicle dynamics and thus reachable states are indifferent to initial position and orientation. These can be later accounted for in a post-processing step by translation and rotation of the reachable set, respectively. Thus, we generate training data with initial position and orientation set to zero. This helps reduce the state space as well as the required inputs to the neural network. Since initial position and orientation are always set at zero, they are not pertinent to the network for generation of reach-sets. This entails that the only required state input to the neural network is the initial velocity. Thus, the network has twelve inputs comprising of one state input, ten control inputs for the sequence of commands, and the step k.

D. Accounting for the size of the car

To ensure safety of the vehicle, we compute the reachable set of the "whole" vehicle (rather than the reachable set of the center of rear axle only). For this purpose, we account for the dimensions of the vehicle and possible orientations and then add this up (via Minkowski sum) with the reachable set we computed for the center of the rear axle. In more detail, the workflow looks as follows:

Dimensions of vehicle. We first start with a box representing the dimensions of the vehicle. We assume that the length l_c is 2.5 meters and width w_c is 2 meters. The center of the rear axle is placed at point (0, 0).

Uncertainty in vehicle orientation. We account for the orientation of the vehicle and the uncertainty in this orientation (due to uncertain dynamics). For this purpose, we first train two additional neural networks to learn reachable orientations $\mathcal{R}_\theta$, and hence work out the maximum and minimum orientation of the vehicle in a time interval. We then compute median orientation θ_{mid}. We calculate the maximum uncertainty in orientation by considering the maximum deviation of reachable orientations from the computed median:

$$\Delta\theta = max_{\theta^* \in \mathcal{R}_\theta} |\theta^* - \theta_{mid}| \quad (8)$$

We enlarge the image (Fig. 3 on the left) to account for this uncertainty.

Vehicle orientation within a time interval. We compute the set representing all the reachable orientations of the car $\mathcal{R}_c$ by rotating the enlarged box by θ_{mid} (see Fig. 3 on the right)

Full reachable set. In order to obtain the full reachable set $\mathcal{R}_T$, we compute the Minkowski sum of the vehicle occupation set $\mathcal{R}_c$ and the set of possible positions of the rear axle center $\mathcal{R}_s$ predicted by the neural network

$$\mathcal{R}_T = \mathcal{R}_c \oplus \mathcal{R}_s \quad (9)$$

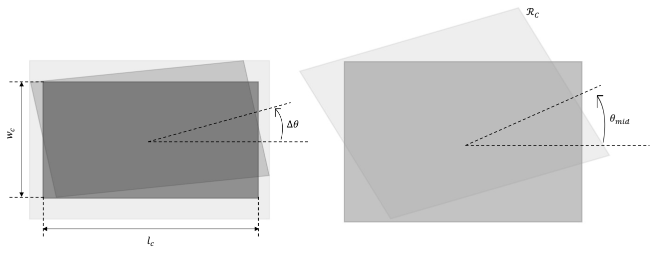

Fig. 3: *Enlargement and rotation of vehicle occupation to account for orientation and uncertainty.*

E. Online Monitoring

In order to showcase the effectiveness of our approach, we employ it for online monitoring of a control system. In particular, we utilize our neural reachability method as a decision module to ensure safety of the system. The module checks the safety of the controller commands at each step and alerts the system if unsafe behavior is observed.

However, as the controller does not account for uncertainty in the dynamics, the vehicle behavior can become unsafe. Thus, the decision module must consider uncertainty to verify safety of the controller's command sequence.

1) Scenario: We consider a scenario where a path planning algorithm generates a safe trajectory for the vehicle to follow while avoiding obstacles. The MPC controller then solves an optimization problem to best follow the specified trajectory. We deploy a cubic spline path planner to generate trajectories for different goals that are safe for the vehicle to follow, whilst ignoring uncertainties. For exact dynamics, the controller will safely guide the vehicle along the path to the desired goal. However, as uncertainties are introduced, safety is violated and collision with obstacles is plausible. This makes our online monitoring system necessary.

The time horizon used by MPC is two seconds and the time interval for one reach set is 0.02 seconds, meaning we have 100 $steps$. Thus, to compute the reachable set up to the full time horizon of two seconds, we must run the neural network 100 times for each support direction. Since we have octagonal direction in a two-dimensional space, we employ eight individual neural networks (for each of the considered template directions). Since each inquiry to a neural network outputs a support value for a given direction and step, these inquiries can be processed concurrently, which in turn makes the whole process highly parallelizable. This helps reduce run-time and perform the analysis in real-time. We note that with specialized hardware the run-time can be reduced even further as each call to the neural network can be parallelized.

2) Evaluation: We construct five different maneuvers that a vehicle typically undertakes. These maneuvers are (1) driving in a straight lane close to a curb, (2) taking a left turn, (3) taking a right turn, (4) making a U-turn, and (5) changing

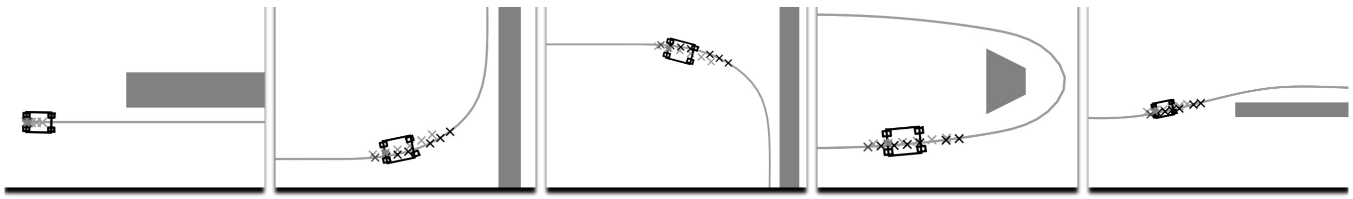

Fig. 4: *Vehicle maneuvers (from left to right): (1) driving in a straight lane close to a curb, (2) taking a left turn, (3) taking a right turn, (4) making a U-turn, and (5) changing lanes due to a roadblock*

lanes due to a roadblock (see Fig. 4). Each maneuver is safe if no uncertainties are present. However, as uncertainty in the dynamics is introduced, the maneuver's trajectory is no longer verified. We observe that with the deployment of our neural reachability algorithm, all the unsafe situations are identified and prevented as the vehicle comes to a halt. Fig. 5 visualizes an example of a computed reachset where a vehicle is undertaking a left turn.

Each maneuver is simulated 100 times to completion, i.e. until collision happens or the vehicle reaches its goal, with the introduced uncertainty. The percentage of collisions without online monitoring are then recorded. We then introduce our online monitoring system where the collisions are avoided and measure execution times. Based on the simulations, we record the the maximum observed execution time (Max ET) as well as the Mean execution time (Mean ET) for our algorithm. These times contain the time taken to run all neural networks as well as time for necessary transformations. The results are recorded in Table I. We observe that uncertainty can make a considerable fraction of simulations unsafe, e.g., 33% of these in case of a U-turn maneuver, while, again, our approach allows to eliminate all of these. In addition, a reachable set can be computed on average within only 41 ms. This is a stark difference to Flow* which needs on average 5870 ms, i.e. two orders of magnitude more, to solve the same problem.

TABLE I: Results for neural reachability for several maneuvers. Collision %: percentage of simulations which resulted in a collision; Mean ET: mean execution time in milliseconds; Max ET: max execution time in milliseconds.

Maneuver	Collision %	Mean ET	Max ET
Straight	9	41.3	79.1
Left turn	27	41.3	125
Right turn	34	41.1	124
U-turn	33	41.1	129
Change lane	29	41.4	125

In the following, we also elaborate on two properties of our approach which can be utilized to reduce execution time even further while checking for unsafe behavior. Firstly, safety can be evaluated on-the-fly, i.e. checking for intersections with an obstacle can be computed for each reach set at every step iteratively. This means that unsafe control commands can be identified as soon as possible, i.e. before the whole reachable set is computed. Thus, we record the mean execution time where a collision is detected (Mean ETC) as well as the maximum execution time (Max ETC). Another key benefit is that the reachable sets for different time steps k, i.e. different time intervals, can be computed in no particular order. Note that traditional reachability analysis methods [14] do not provide this flexibility because reachability sets for a particular time interval require the knowledge of reachable sets at an earlier time interval. We exploit this observation by computing reach sets in reverse order, i.e. computing sets from step $k = 100$ to $k = 1$ and checking for intersection with unsafe regions on the fly. We report the mean (Mean RETC) and maximum (Max RETC) execution time for this approach. These results are shown in Table II. We observe that although the values of Mean ETC are only marginally lower than those of Mean ET, doing an on-the-fly check helps decreasing the maximum execution time by up to the factor of three. The computation of reachable sets in reverse order enables further performance boost in the sense of both mean and max execution times. In particular, in this setting, the mean execution time (Mean RETC) is less than the half of the Mean ETC values consistenly in all considered scenarios. In addition, this approach allows to considerably reduce the maximum execution time. In fact, we see that for the most of maneuvers the maximum execution time is only slightly larger than the mean execution time. These results can be justified by the fact that the vehicle constantly forecasts its environment and the possible obstacles tend to be detected for larger values of the time step k.

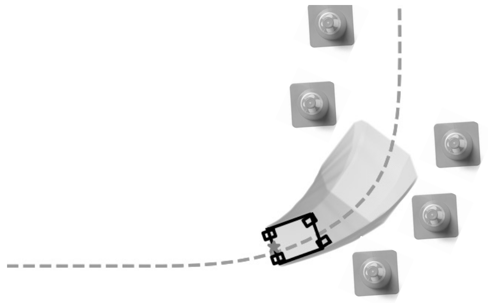

Fig. 5: *Visualization of computed reachable states for full vehicle undertaking a left turn in a time horizon of* $2s$.

F. *Simplex Architecture*

We further evaluate our neural reachability approach by extending our framework to the simplex architecture [29]. In this framework, an online monitoring system switches back and forth between an unverified, advanced controller and a safe, baseline controller. We employ neural reachability as the switching logic, as such, it checks the safety of the advanced controller's command sequence and switches to the

TABLE II: Results for neural reachability with on-the-fly safety check. Mean ETC: mean execution time with on-the-fly collision check in milliseconds; Max ETC: max execution time with on-the-fly collision check in milliseconds; Mean RETC and Max RETC are similar to Mean and Max ETC, respectively, but the collision check is done in reverse order, i.e. for later time intervals first.

Maneuver	Mean ETC	Max ETC	Mean RETC	Max RETC
Straight	40.6	43.1	15.5	16.0
Left turn	39.6	74.2	16.2	49.6
Right turn	39.6	42.7	15.5	16.0
U-turn	37.9	39.4	15.7	16.2
Change lane	40.6	81.6	16.1	56.9

safe controller if unsafe behavior is detected. Fig. 2 shows an overview of our simplex framework The framework is deployed in a simulated environment on the f1tenth gym platform [32], where an autonomous vehicle aims to drive around a track efficiently and safely, while avoiding static obstacles. The f1tenth gym platform is an open-source, well documented, low-cost platform that enables researchers to conduct asynchronous, and realistic vehicle simulations which investigate planning, and advanced control. It is built on the OpenAI Gym [33] toolkit which enables deterministic stepping and resetting of the physical car.

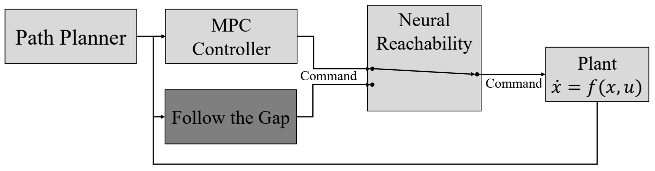

Fig. 6: *Overview of simplex framework.*

1) Controllers: We utilize the previously developed MPC controller as the advanced controller to drive the vehicle efficiently around the track. However, with the presence of randomly placed static obstacles, the controller is no longer safe. Thus, it is wrapped with a safe controller to avoid collisions. Verifiable safe controllers are often hard to develop and require rigorous mathematical specifications [34]. Furthermore, "safety" is relative to the application domain and individual perspective [35]. Thus, we opt for using Follow the Gap method [36] as our baseline controller to safely avoid collisions. This controller is simplistic, robust, simple to tune and debug.

2) Scenario: In the considered scenario, we deploy an F1/10 vehicle driven by the simplex architecture to drive around a racetrack safely and efficiently with randomly placed cones (see Fig. 7). The advanced MPC controller is given default control and neural reachability continuously checks the safety of each command sequence by approximating the corresponding reachable set. If a potential collision with an obstacle is detected, it switches control to the Follow the Gap method which retains control until a command sequence of the MPC controller is deemed safe. Once this holds, control of the vehicle is switched back to the MPC controller to follow the optimal trajectory to drive around the track.

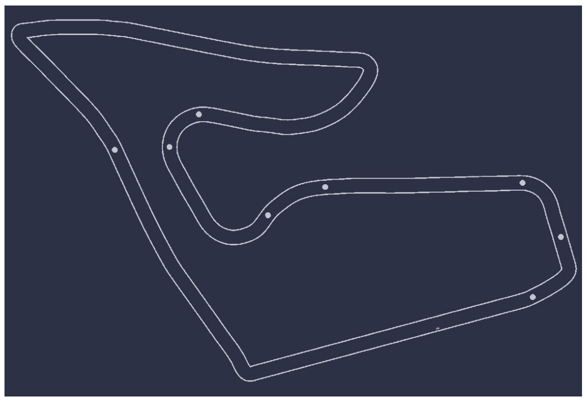

Fig. 7: *F110 Spielberg track with randomly placed cones.*

3) Evaluation: We randomly generate obstacles around the racetrack and employ our simplex framework to drive the vehicle. We reduce the time horizon for the neural reachability decision module to 0.6s so that it switches to the safe controller only when a collision within an obstacle is detected within that time horizon. This is to avoid being over conservative and prematurely switching to the Follow the Gap method whilst also providing sufficient time for the algorithm to react when an unsafe situation is detected

We record that for our experiments controller usage is 88.9% for the MPC and 11.1% for the Follow the Gap method. We showcase that our simplex framework can safely and efficiently drive the vehicle around the track whilst avoiding obstacles by switching between the controllers. The video for the implementation can be found at https://youtu.be/SPpvIwQ2VrY

G. *Statistical Guarantees*

Due to the general uncertainty in the output of neural networks, we provide probabilistic guarantees on the validity of the computed reach-set for the case-study. Namely, we provide guarantees on correctness and accuracy. The guarantees show that $\mathcal{P}_A > 99.3\%$ (the outputted support values that are within 0.2 of the true value are greater than 99.3%) and $\mathcal{P}_U < 0.05\%$ (the percentage of underapproximation in data are less than 0.05%).

VI. Conclusion and Future Work

We have presented a technique for real-time verification, by learning deep neural networks the reachability analysis problem. Our experiments in the case study show the efficacy of the developed solution for online monitoring. We further demonstrate that by utilizing key properties of our approach, and namely, on the fly safety validation and non-chronological computation of reachsets, runtime is significantly reduced. We further showcase that neural reachability can be employed as part of the simplex architecture for safe and efficient control. Subsequently, we provide statistical guarantees that show high accuracy and a low under-approximation rate of the predicted reachset.

In our case study, we have limited our consideration to static obstacles. In the future, we aim to extend our framework to moving obstacles and opponent vehicles to fully illustrate the effectiveness of our approach. Furthermore, we aim to deploy our framework on hardware and evaluate it's performance.

VII. Acknowledgments

This research was supported in part by the Air Force Office of Scientific Research under award number FA2386-17-1-4065. Any options, finding, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

References

[1] A. Jain, L. Del Pero, H. Grimmett, and P. Ondruska, "Autonomy 2.0: Why is self-driving always 5 years away?" *arXiv preprint arXiv:2107.08142*, 2021.

[2] G. Frehse, "PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech," in *Hybrid Systems: Computation and Control*, ser. LNCS, vol. 3414. Springer, 2005, pp. 258–273.

[3] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 3–17.

[4] 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*, 2019, pp. 169–178.

[5] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[6] S. Bak, T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *Proceedings - Real-Time Systems Symposium*, vol. 2015, pp. 138–148, 01 2015.

[7] M. Vitelli, Y. Chang, Y. Ye, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, and P. Ondruska, "Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies," *arXiv preprint arXiv:2109.13602*, 2021.

[8] I. M. Mitchell, "Comparing forward and backward reachability as tools for safety analysis," in *Hybrid Systems: Computation and Control*, A. Bemporad, A. Bicchi, and G. Buttazzo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 428–443.

[9] G. Frehse, C. L. 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 *Proceedings of the 23d CAV*, 2011.

[10] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *International workshop on hybrid systems: Computation and control*. Springer, 2000, pp. 20–31.

[11] A. Bemporad and M. Morari, "Verification of hybrid systems via mathematical programming," in *Hybrid Systems: Computation and Control*, F. W. Vaandrager and J. H. van Schuppen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 31–45.

[12] C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, 2003.

[13] G. Frehse, C. L. Guernic, A. DonzÃ¯Â¿Â½, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable Verification of Hybrid Systems," in *Proceedings of the 23d CAV*, 2011.

[14] X. Chen, E. Abraham, and S. Sankaranarayanan, "Flow*: An Analyzer for Non-Linear Hybrid Systems," in *Computer-Aided Verification (CAV)*, ser. LNCS, vol. 8044. Springer-Verlag, 2013, pp. 258–263.

[15] X. Chen and S. Sankaranarayanan, "Decomposed reachability analysis for nonlinear systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 13–24.

[16] 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.

[17] M. P. Owen, A. Panken, R. Moss, L. Alvarez, and C. Leeper, "Acas xu: Integrated collision avoidance and detect and avoid capability for uas," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 2019, pp. 1–10.

[18] M. Prandini, J. Lygeros, A. Nilim, and S. Sastry, "Randomized algorithms for probabilistic aircraft conflict detection," in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*, vol. 3. IEEE, 1999, pp. 2444–2449.

[19] J. Lygeros and M. Prandini, "Aircraft and weather models for probabilistic collision avoidance in air traffic control," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3. IEEE, 2002, pp. 2427–2432.

[20] Y. Chou, H. Yoon, and S. Sankaranarayanan, "Predictive runtime monitoring of vehicle models using bayesian estimation and reachability analysis," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2111–2118.

[21] B. Djeridane and J. Lygeros, "Neural approximation of pde solutions: An application to reachability computations," in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 3034–3039.

[22] Y. Meng, D. Sun, Z. Qiu, M. T. B. Waez, and C. Fan, "Learning density distribution of reachable states for autonomous systems," in *Conference on Robot Learning*. PMLR, 2022, pp. 124–136.

[23] V. Rubies-Royo, D. Fridovich-Keil, S. Herbert, and C. J. Tomlin, "A classification-based approach for approximate reachability," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7697–7704.

[24] D. Phan, N. Paoletti, T. Zhang, R. Grosu, S. A. Smolka, and S. D. Stoller, "Neural state classification for hybrid systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018, pp. 422–440.

[25] L. Bortolussi, F. Cairoli, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural predictive monitoring and a comparison of frequentist and bayesian approaches," *International Journal on Software Tools for Technology Transfer*, pp. 1–26, 2021.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

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

[28] M. Obayashi, K. Uto, and G. Takano, "Appropriate overtaking motion generating method using predictive control with suitable car dynamics," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 4992–4997.

[29] 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. IEEE, 1998, pp. 3504–3508.

[30] L. Wang, *Model predictive control system design and implementation using MATLAB®*. Springer Science & Business Media, 2009.

[31] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Pythonrobotics: a python code collection of robotics algorithms," *arXiv preprint arXiv:1808.10703*, 2018.

[32] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "textscf1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020, pp. 77–89.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[34] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue, "Formal specification for deep neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018, pp. 20–34.

[35] A. Reschka, "Safety concept for autonomous vehicles," in *Autonomous Driving*. Springer, 2016, pp. 473–496.

[36] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889012000838