 The original paper was published in *IEEE Control Syst. Lett.*. **6**, pp. 1574-1579, 2022. http://dx.doi.org/10.1109/LCSYS.2021.3125717

Safe and Resilient Practical Waypoint-following for Autonomous Vehicles

Qin Lin, Stefan Mitsch, André Platzer, *Senior Member, IEEE*, and John M. Dolan, *Senior Member, IEEE*

***Abstract*— We combine theorem proving and reachability analysis for cyber-physical systems verification to arrive at a practical approach to safe waypoint-following for an autonomous mobile vehicle controlled by a learning-enabled controller. We propose a robust monitor verifying short-term and long-term safety simultaneously at runtime, thereby combining the benefits of both theorem proving and reachability analysis. The proposed novel monitor architecture allows temporary violation of long-term safety while maintaining short-term safety to recover to a state with long-term safety. The recovery is based on a fallback model predictive controller. The experiments conducted in a high-fidelity racing car simulator demonstrate that our framework is safe and resilient in path tracking scenarios, in which avoiding collision with the race track boundary and obstacles is required.**

***Index Terms*— Reachability analysis, Theorem proving, Safety verification, Safe control, Learning-enabled control**

I. Introduction

LEARNING-ENABLED components (LECs) are being deployed in autonomous systems. However, reliability is a crucial concern because many autonomous systems are safety-critical. In this letter, we are particularly interested in the safety assurance in the control layer. The Simplex architecture [1] has been used for a safety-assured closed-loop control system and provides a way to operate learning-enabled controllers safely even when they are themselves not formally verified. In such a setup there are two controllers: one is a high-performance LE controller, e.g., a neural network (NN)-based controller; the other one is a high-assurance fallback controller. A monitor decides between operating the LE controller and the fallback controller. When the control action of the LE controller is not verified to be safe, we switch to the fallback controller. Besides the high-assurance fallback controller, correctness guarantees for this approach hinge on the correctness of the monitor for the decision logic.

Formal verification approaches have been leveraged to design the monitor. There are three main categories: 1) *Barrier certificates*. The idea is to find a barrier that cannot be crossed by the system's trajectory. A barrier certificate is usually defined as a Lyapunov-like function [2]. In the control domain, the certificates can be control barrier functions (CBFs) used as hard safety constraints (by analogy with the safety condition of a monitor in Simplex) for computing optimal control (by analogy with fallback control in Simplex). The framework works as a safety filter, which unifies the monitor and fallback control synthesis in an optimization program. There also exist other types of safety filters, e.g., predictive filters [3]. 2) *Reachability analysis*. This is based on forward state exploration over a bounded time horizon using set computations [4]. A reachability-based monitor performs all computations online and is satisfied when the forward-projected states do not intersect with unsafe states. Assuming model correctness, it provides safety guarantees up to the bounded time horizon. It is worth noting that forward reachability is fundamentally different from backward reachability [5]. 3) *Theorem proving*. Hybrid systems theorem proving does not require online state exploration, but shifts computation effort offline to prove time-unbounded correctness properties e.g., safety and liveness. Theorem proving may use and thus formally verify invariant properties, e.g., differential invariants or barrier certificates [6]. Even most of a runtime verification proof [7], which shows that a satisfied monitor provably implies system safety, is pre-computed offline: the resulting monitor conditions in real-arithmetic represent the remaining proof obligations that are verified at runtime from observations and control decisions. The approach validates (rather than assumes) model correctness at runtime and transfers unbounded-time offline proofs to system runtime. A promising direction is to combine formal verification with learning, e.g., safe reinforcement learning [8].

These approaches have different theoretical advantages, this leaves the question of how to best approach correctness for practice. We summarize comparisons between reachability analysis and theorem proving. First, theorem proving can verify safety and liveness in infinite time without exploring all possible states, whereas reachability analysis using state exploration in unbounded time is generally undecidable [9]. Second, the state exploration of reachability analysis repeats in a new control cycle with an updated initial state. For computations to succeed, it is important that the initial regions be small and the time horizon short. That is why safety verification is also done online from the current state by checking the intersection of the reachable states and a pre-defined unsafe set. A well-known bottleneck of reachability analysis is the computation burden. Theorem proving shifts computations offline to verify

This paragraph of the first footnote will contain the date on which you submitted your paper for review. This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092.

Qin Lin and John Dolan are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA qinlin,jdolan@andrew.cmu.edu

Stefan Mitsch and André Platzer are with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. smitsch,aplatzer@cs.cmu.edu

unbounded-time/-space safety conditions using provers such as KeYmaera X [10]. The resulting arithmetic condition can be easily and quickly checked for the control action at runtime. Third, while theorem proving-based approaches can support fairly general dynamic systems, their complexity and human intervention are reduced for simpler models; therefore, existing safety verification results of vehicle dynamic systems [11], [12] avoid sophisticated models. At the expense of time-bounded predictions and extensive online computations, reachability analysis is able to scale with less human intervention to nonlinear models with slip angle and tire force.

In this letter, we address a general waypoint-following problem for an autonomous vehicle. Our previous work [12] used ModelPlex [7] to develop a theorem proving-based monitor. The ModelPlex monitor consists of a *model monitor* and a *controller monitor*: The model monitor performs online model validation and checks whether the verified model accurately represents the system dynamics by inspecting observed states. The controller monitor checks whether the proposed control is considered safe by the verified model by inspecting the current state and the control output. In this work, we focus on controller monitors and assume that the dynamic model is accurate and that the model disturbance has been captured by the tolerance setting in the monitor. The monitor's arithmetic safety condition is proved offline using the KeYmaera X theorem prover [10]. If the monitor is satisfied, we have a formal guarantee that the vehicle remains able to reach the waypoint and that it will respect a desired speed range once it reaches the waypoint. The model [12] uses relative coordinates to represent waypoints in the reference frame of the vehicle, which makes the resulting monitors "blind" to global constraints such as road limits; those are the responsibility of our reachability analysis-based monitors. Note that this condition is "invariant-like", i.e., it needs to be satisfied in each control cycle to guarantee "long-term" safety. However, in practice, we observe that disturbances and other effects that are not represented in the model make it difficult for a vehicle to always satisfy the monitors. This can be problematic when the fallback control is conservative (e.g., slows down the vehicle) so that even small violations disrupt efficient control.

The first research question is: *Is there a robust monitor to ensure safety while minimizing the intervention?* To answer this question, we relax the long-term safety by allowing small violations with a manageable hazard in short term. We introduce reachability analysis to verify short-term safety. Thus, small violations of the long-term safety when there is a short-term online safety guarantee will not invoke fallback control. The fallback controller only intervenes when the nominal controller violates the short-term safety.

The second research question is: *Can we design a fallback controller to steer the vehicle back to a state without short-term and long-term safety violations?* To answer this question, we use an efficient sampling approach to obtain a safe state from the monitor's safety condition and the safe set based on the reachability analysis. The safe state is used as a target state for a fallback model predictive controller (MPC).

We make the following contributions in this letter:

1) To the best of our knowledge, this is the first work integrating theorem proving and reachability for a runtime monitor verifying short-term and long-term safety simultaneously.
2) We design a novel Simplex-like framework to guarantee safety and minimize interventions.
3) We demonstrate in a simulator involving high-fidelity racing (RC) cars. The RC car is controlled by a neural network. Using our framework, the car can safely follow the planned trajectory with minimized risks of running outside the track and colliding with obstacles.

II. Short-term and long-term safety monitors

A. Long-term waypoint following monitors

We summarize the main features of our prior waypoint path following model [12]. We use hybrid programs with ordinary differential equations (ODEs) to concisely describe control laws and kinematics of our system, and verify properties about hybrid programs using differential dynamic logic (dL). The dynamics model of the vehicle is in Eq. (1).

$$\mathsf{dyn} \equiv t := 0;\ \{x' = v(ky-1), y' = -vkx, v' = a, \\ t' = 1\ \&\ t \leq T \wedge v \geq 0\} \tag{1}$$

where a relative coordinate system is used: the vehicle's position is placed at the origin, and (x, y) are the relative coordinates of a waypoint. Positive x points forward, and positive y points to the left. Positive k represents clockwise motion of the waypoint towards the vehicle (when the waypoint is on the left side of the vehicle), $k = 0$ is straight-ahead, and negative k means counter-clockwise motion (when the waypoint is on the right side of the vehicle). v and a are the longitudinal velocity and acceleration, respectively. $t' = 1$ is a clock, and T is the upper bound on the time step of a control cycle. The vehicle does not drive in reverse: $v \geq 0$.

The perfect path for the vehicle to follow is padded with a tolerance size ϵ of an annular section, see Eq. (2) and Fig. 1.

$$\mathsf{Ann} \equiv |\mathsf{k}|\epsilon \leq 1 \wedge \left|\frac{\mathsf{k}(\mathsf{x}^2 + \mathsf{y}^2 - \epsilon^2)}{2} - \mathsf{y}\right| < \epsilon \tag{2}$$

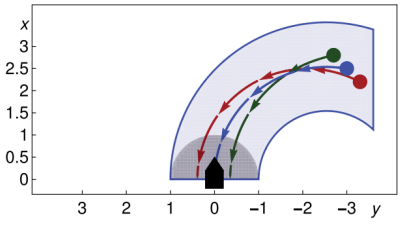


Fig. 1: Annular section through the blue waypoint $(2.5, -3)$. Trajectories inside the section are legitimate, e.g., trajectories for green/red waypoints with slightly different curvatures.

The controller ctrl in Eq. (3) receives a waypoint (x, y) and desired speed range $[vl, vh]$ from a high-level planner and checks Feas to determine whether the waypoint is reachable given the steering and acceleration/deceleration.

$$\mathsf{ctrl} \equiv (\mathsf{x}, \mathsf{y}) := *;\ [\mathsf{vl}, \mathsf{vh}] := *;\ ?\mathsf{Feas};\ \mathsf{a} := *;\ ?\mathsf{Go} \tag{3}$$

It further produces an acceleration or deceleration choice a, subject to condition Go. The precise characterization of Feas and Go is crucial for long-term safety, but not required for understanding, cf. [12]. The important aspect for our purposes here is that Feas and Go are predictive, i.e., they need to decide whether following the chosen (x, y), $[vl, vh]$, and a along the dynamics of dyn preserves desired control properties.

The overall control objective of reaching the waypoint and respecting the desired speed range is implied by an invariant J, which is proved to be maintained in each control cycle so that: 1) the vehicle follows the plan closely; 2) it drives at speeds that let it achieve the speed limits in the remaining distance to the goal. The exact formal definition of J can be found in [12]: J captures the annular section between the vehicle and the waypoint, as well as how much space is needed to close the gap between current vehicle speed and speed limit.

The ModelPlex [7] automatically turns the program ctrl into a controller monitor that checks the predictive conditions Feas and Go and provably maintains J at runtime. In the implementation, we check two consecutive states $(s_i, \tilde{s}_i)$ in every control cycle online with the controller monitor. Specifically, we verify whether the current control choice γ_i—made by an untrusted controller in state s_i, result captured in $\tilde{s}_i$—corresponds to a correct choice per ctrl *before it is executed*; if γ_i satisfies the controller monitor, executing γ_i is guaranteed to steer the current state s_i to a future state s_j that maintains the safety invariant J as long as the vehicle dynamics continue to behave in a way allowed by dyn. In this letter, our focus is the controller monitor by assuming the model is (nearly) accurate, i.e., the model is precisely described or the model disturbance has been captured by the tolerance ϵ. The checking procedure is extremely efficient, since the controller monitor is composed of arithmetic conditions that are evaluated on concrete values.

B. Reachability analysis

We use Flow* [13] for nonlinear forward reachability analysis. Given a dynamic model defined by ODEs, state and control actions with box-bounded uncertainty, and a prediction horizon, Flow* computes the reachable set. All possible future trajectories will be included in the reachable set with a theoretical guarantee owing to an over-approximation.

The dynamic model is $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where $\boldsymbol{x}(t) := (x_1(t), \cdots, x_n(t))^T \in \mathbb{R}^n$ and $\boldsymbol{u(t)} := (u_1(t), \cdots, u_m(t))^T \in \mathbb{R}^m$. The reachability analysis essentially tries to solve an *initial value problem*, i.e., given condition $\boldsymbol{x}(0) \in \mathcal{X}_0 \subset \mathbb{R}^n$ and a closed and bounded set $\mathcal{U}$ for $\boldsymbol{u}$, the *reachable set* of the system in a given time interval $[0, T_h]$ is defined by $\underset{T_h}{\textbf{reach}}(\boldsymbol{x}(0), \mathcal{U}) := \{\varphi_f(\boldsymbol{x}(0), \boldsymbol{u}(0), t) \,|\, \boldsymbol{x}_0 \in \mathcal{X}_0, \boldsymbol{u}(t) \in \mathcal{U}, t \in [0, T_h]\}$, where $\varphi_f(\boldsymbol{x}_0, \boldsymbol{u}_0, t)$ is the solution of the ODE with initial state $\boldsymbol{x}_0$ and control $\boldsymbol{u}(t)$.

In practice, we compute the time-dependent reachable set iteratively for discrete time increments $\Delta t \in \mathbb{R}^+$. At each time point $t_k = k\Delta t$ with $k \in \mathbb{N}$, we define the reachable set: $\mathcal{R}_{k+1} := \underset{t_{k+1}}{\textbf{reach}}(\mathcal{R}_k, \mathcal{U}),\ \mathcal{R}_0 = \mathcal{X}_0$.

It is not possible to compute the exact set $\mathcal{R}_k$ in general, due to the fact that getting the solution φ_f is difficult. We compute an over-approximation denoted by $\hat{\mathcal{R}}_k$. A Taylor model is used for the representation of an over-approximated reachable set.

Definition 2.1: (Taylor Model): A Taylor Model (TM) of order k is denoted by a pair $(p, \mathbb{I})$, wherein p is a Taylor polynomial of at most degree k, and $\mathbb{I}$ is a *remainder interval*. Given a TM $(p, \mathbb{I})$ and a function φ_f which are both over the same domain $\mathbb{R}^n$, φ_f is over-approximated by $(p, \mathbb{I})$, denoted by $\varphi_f \in (p, \mathbb{I})$, i.e., $\varphi_f(\boldsymbol{x}) \in p(\boldsymbol{x}) + \mathbb{I}, \forall \boldsymbol{x} \in \mathbb{R}^n$.

The TM flowpipe construction is a process to get over-approximation sets $\hat{\mathcal{R}}_1, \cdots, \hat{\mathcal{R}}_k := (p_1, \mathcal{I}_1), \cdots, (p_k, \mathcal{I}_k)$. In this letter, a *kinematic bicycle model* is used. All state and control actions with uncertainty and the resulting reachable set at each state dimension are represented by bounding boxes with minimum and maximum values.

1) Kinematic bicycle model:

$$\begin{aligned} \dot{X} &= v_x \cos(\psi + \beta) \\ \dot{Y} &= v_x \sin(\psi + \beta) \\ \dot{\psi} &= v \sin(\beta)/l_r \\ \dot{v}_x &= a \\ \beta &= \arctan\left(l_r \frac{\tan(\delta)}{l_f + l_r}\right) \end{aligned} \tag{4}$$

where δ is the steering angle, a is the longitudinal acceleration, X and Y are coordinates in a global initial frame. $\dot{X}$ and $\dot{Y}$ are the corresponding velocity and ψ and $\dot{\psi}$ are the yaw and the yaw rate, v_x is the longitudinal velocity, β is the slip angle of the current velocity of the center of mass with respect to the longitudinal axis of the car, and l_f and l_r represent the distance from the center of mass of the vehicle to the front and rear axles, respectively.

Definition 2.2: (Cartesian occupancy): The occupancy of a vehicle at time t is a subspace of its global positions in a Cartesian inertial frame projected from the reachable set, $\textbf{proj}(\hat{\mathcal{R}}_k) := \text{rectangle}(\underline{X}_k, \bar{X}_k, \underline{Y}_k, \bar{Y}_k)$

An unsafe control causing collision is detected when the Cartesian occupancy of the ego vehicle has any intersection with the pre-defined unsafe set, i.e., $\textbf{proj}(\hat{\mathcal{R}}_k) \cap \mathcal{S}_{unsafe} \neq \emptyset$.

III. Proposed Framework

Combinations of the diagnostic results of ModelPlex (d^m) and reachability (d^r) monitors are shown in Tab. I. Their values 1 and 0 denote safe and unsafe, respectively.

TABLE I: Reachability and ModelPlex

Case	Diagnosis	Note
1	$d^r = 0 \ \wedge \ d^m = 0$	OA violated; WP unreachable
2	$d^r = 0 \ \wedge \ d^m = 1$	OA violated; WP reachable
3	$d^r = 1 \ \wedge \ d^m = 0$	OA respected; WP unreachable
4	$d^r = 1 \ \wedge \ d^m = 1$	OA respected; WP reachable

OA: obstacle avoidance; WP: waypoint; $\wedge$: logical conjunction

In Fig. 2, we divide the whole space into three sets: a forbidden set (cases 1 and 2 in Tab. I), a temporarily unsatisfactory set (case 3 in Tab. I), and a desirable set (case 4 in Tab. I). They are also called the red zone, yellow zone, and green zone, respectively. The red zone is forbidden, since a violation is imminent and fallback control is needed to

steer the vehicle to leave the red zone. In the yellow zone, the vehicle does not have a long-term guarantee of reaching the waypoint, but we avoid fallback control since there is no imminent short-term safety concern and long-term waypoint following can be re-established. In the green zone, the vehicle has both short-term and long-term guarantees. In summary, we allow the vehicle to mildly violate the long-term requirement, but prevent it from entering the red zone.

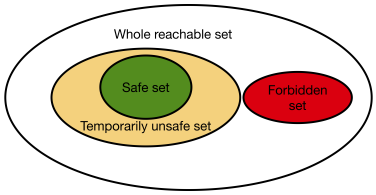


Fig. 2: Safe, temporarily unsafe, and forbidden sets

The recovery mechanism is detailed in Alg. 1 and Alg. 2. Alg. 1 is the main routine, which executes in every control cycle. An intuitive illustration is shown in Fig. 3. The steering control for the ModelPlex monitor is the waypoint's curvature, while reachability analysis uses steering angle. For consistency, a transform based on Ackermann steering geometry is used, i.e., $k = \tan(\delta)\cos(\beta)/(l_f + l_r)$.

The idea of recovery is to maintain a recovery state sequence srb and a recovery control sequence crb in the yellow zone, though no fallback is needed. The diagnostic results are from ModelPlex $diag_m(.)$ and reachability $diag_r(.)$ (Line 3 in Alg. 1), respectively. srb and crb are empty in the beginning (Lines 1–2 in Alg. 1). When the vehicle is predicted to stay in the green zone, srb and crb are appended with the predicted safe state $\tilde{\boldsymbol{s}}_i$ and the safe control action $\boldsymbol{u}_i$, respectively. f is the vehicle dynamic model. No intervention is needed to override the current control action (Lines 4–7 in Alg. 1).

When the vehicle is predicted to enter the yellow zone (see t_0 in Fig. 3), the subroutine Sampling(.)—Alg. 2—is called to sample the neighborhood control of the last safe control to obtain safe future states for potential recovery use. The neighborhood size $\boldsymbol{\epsilon}_c$ is a tunable parameter. Note that since we only maintain a recovery sequence, only one recovery state and its associated control action are kept, though there might exist multiple safe states and control actions. We pick up the state with the closest Euclidean distance to the waypoint in the Cartesian space (Line 13 in Algo. 2).

The recovery sequences continue growing until we have a risk of entering the red zone (see t_4 in Fig. 3). We retrieve the last element in srb as the safe target state to invoke a fallback controller to steer the vehicle to leave the red zone (see Line 13 in Algo. 1). The recursive feasibility of getting a target safe state from ModelPlex can be formulated as $Pr(\boldsymbol{s}(t_i) \in \mathcal{S}_{safe}, | \boldsymbol{s}(t_{i-1}) \in \mathcal{S}_{safe}, \boldsymbol{u}_{i-1} \in \mathcal{U})$, where $\mathcal{S}_{safe}$ is the safe set from ModelPlex, i.e., the probability of two consecutive states both being safe. We assume that $\forall \boldsymbol{s}(0) \in \mathcal{S}_{safe}$, there exists a one-step reachable neighborhood $\Omega := \underset{[t_{i-1},t_i]}{\textbf{reach}}(\boldsymbol{s}(0), \mathcal{U})$, and $\Omega \cap \mathcal{S}_{safe} \neq \emptyset$. A larger sampling space implies a higher likelihood covering Ω, thus ensure the safety of the next sampled state. An intuitive illustration is in Fig. 1: the safe annular section is a perfect trajectory with an infinite number of neighborhood trajectories under small perturbations. With a larger sampling space, it is more likely to sample the next safe state. In practice, choosing the nearest waypoint can be a backup strategy if the sampling space is too narrow to miss the safe state.

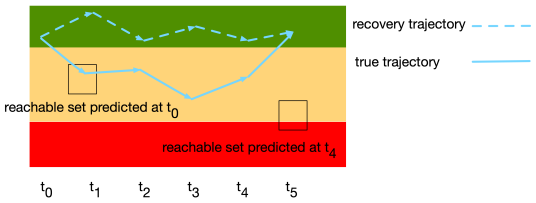


Fig. 3: Recovery mechanism illustration

Algorithm 1 Main recovery framework

Input: $\boldsymbol{u}_i$ from the nominal controller, state $\boldsymbol{s}_i$, and waypoint's global coordinate $wp_i = (X_i, Y_i)$
Output: New control action $\boldsymbol{u}_i'$

1 **if** *i==0* **then**
2 $srb = [\,]$, $crb = [\,]$.
3 $d_i^m = diag_m(\boldsymbol{s}_i, \boldsymbol{u}_i, wp_i)$; $d_i^r = diag_r(\boldsymbol{s}_i, \boldsymbol{u}_i)$;
4 **if** $d_i^m = 1 \ \wedge \ d_i^r = 1 \triangleright$ `in green zone`
5 **then**
6 $srb = [\tilde{\boldsymbol{s}}_i]$, $crb = [\boldsymbol{u}_i]$; $\triangleright$ $\tilde{\boldsymbol{s}}_i := f(\boldsymbol{s}_i, \boldsymbol{u}_i)$
7 **return** $\boldsymbol{u}_i$;
8 **else if** $d_i^m = 0 \ \wedge \ d_i^r = 1 \triangleright$ `in yellow zone`
9 **then**
10 call Sampling(srb, crb, $\boldsymbol{s}_i, wp_i$);
11 **return** $\boldsymbol{u}_i$;
12 **else**
13 Use $srb_{size(srb)-1}$ as a target in MPC to compute $\boldsymbol{u}_i'$;
14 **return** $\boldsymbol{u}_i'$;
15 **end**

Algorithm 2 Sampling

Input: srb, crb, $\boldsymbol{s}_i$, wp_i
Output: updated srb, crb

1 Initialize a temporary state set $sp = \{\}$ and a temporary control action set $cp = \{\}$.
2 **for** *control action* $\boldsymbol{c}_j$ *in* $[crb_{size(srb)-1} - \boldsymbol{\epsilon}_c, crb_{size(srb)-1} + \boldsymbol{\epsilon}_c]$ **do**
3 $d_j^m = diag_m(srb_{size(srb)-1}, \boldsymbol{c}_j, wp_i)$;
4 $d_j^r = diag_r(srb_{size(srb)-1}, \boldsymbol{c}_j)$;
5 **if** $d_j^m = 1 \ \wedge \ d_j^r = 1 \triangleright$ `in green zone`
6 **then**
7 $sp = sp \cup f(srb_{size(srb)-1}), \boldsymbol{c}_j)$;
8 $cp = cp \cup \boldsymbol{c_j}$;
9 **else**
10 continue;
11 **end**
12 **end**
13 $ind = \underset{k}{\text{argmin}} ||sp_k \downarrow \mathbb{C} - wp_i||, k \in \{0, 1, \cdots, size(sp) - 1\}$
14 $srb.append(sp_{ind})$, $crb.append(cp_{ind})$;
15 **return** srb and crb;

IV. Experimental results

We conduct experiments in an open-source racing car simulator [14]. The video can be found online[1].

A. Vehicle dynamic model for control

The dynamic model for control does not have to be the same as the aforementioned models. To deal with the arbitrary geometry of a road, we use the curvilinear coordinate [14], which is convenient for control, since the road boundary can be handled as a lateral position constraint in the optimization. The transformation between curvilinear and global coordinates is available in [14]. The new dynamic model is $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$, where $\boldsymbol{x}_k = [v_{x,k}, v_{y,k}, \phi_k, e_{\phi,k}, s_k, e_{y,k}]^T$, $\boldsymbol{u}_k = [a_k, \delta_k]^T$, $v_{x,k}$ and $v_{y,k}$ are longitudinal and lateral velocity, respectively, ϕ_k is yaw rate, $e_{\phi,k}$ is yaw error from the path, s_k is the curvilinear distance traveled along the centerline of the track, and $e_{y,k}$ is the lateral error from the path.

B. Neural network-based path tracking controller

Due to convenience of training, imitation learning is used in this letter, which has a demonstrator to obtain state-to-control mapping data [15]. We use a fully-connected FNN as a LE controller. The training dataset is from samples of the Stanley controller [16], i.e., $\mathcal{D} = (\boldsymbol{\xi}_0, \boldsymbol{u}_0), (\boldsymbol{\xi}_1, \boldsymbol{u}_1), \cdots, (\boldsymbol{\xi}_n, \boldsymbol{u}_n)$, where $\boldsymbol{\xi_i} = [e_\phi, e_y, e_{v_x}]^T$, e_{v_x} is the velocity error with respect to the desired velocity. The cost function can be expressed as a squared loss function: $J = \sum_{i=0}^{n} ||F(\boldsymbol{\xi}_i) - \boldsymbol{u}_i||$. The training is done by using back-propagation to minimize the loss offline.

C. Fallback controller based on MPC

We use a MPC to design the fallback controller by online-solving the following finite-horizon optimization problem:

$$\min_{\boldsymbol{U}} \quad J = \sum_{k=0}^{N-1} (\tilde{\boldsymbol{x}}_k Q \tilde{\boldsymbol{x}}_k + \boldsymbol{u}_k R \boldsymbol{u}_k) + \tilde{\boldsymbol{x}}_N Q_N \tilde{\boldsymbol{x}}_N \tag{5}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{6}$$

$$\boldsymbol{x}_k \in \mathcal{X} \tag{7}$$

$$\boldsymbol{u}_k \in \mathcal{U} \tag{8}$$

$$(\mathcal{X} \downarrow \mathbb{C}) \cap \mathcal{O} = \emptyset \tag{9}$$

where $\boldsymbol{U} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_{N-1}]$ is the sequence of control actions, Q, R, and Q_N are weighting coefficient matrices, and $\tilde{\boldsymbol{x}}_k = \boldsymbol{x}_k - \boldsymbol{x}_k^{\boldsymbol{r}}$ and $\tilde{\boldsymbol{x}}_N = \boldsymbol{x}_N - \boldsymbol{x}_N^{\boldsymbol{r}}$ are differences between states and references. The reference state is obtained using the recovery introduced in Section III. $\mathcal{X}$ and $\mathcal{X}_f$ are state constraints; e.g., the vehicle is required not leave the track by setting $abs(e_{y,k}) \leq$ *track_width* for $\forall k \in \{0, 1, \cdots, N\}$. $\mathcal{U}$ is the control limit. Eq. (9) says the projection of the vehicle's state space ($\mathcal{X}$) into Cartesian space ($\mathbb{C}$) does not have any intersection with obstacles ($\mathcal{O}$). We use the Euclidean relative distance between the vehicle (v) and an obstacle (o): $(e^v_{y,k} - e^o_{y,k})^2 + (s^v_{y,k} - s^o_{y,k})^2 \geq$ *safety_margin* as a constraint. We only have a convergence guarantee, since the optimization just reaches the safe state as closely as possible. However, we ensure not to enter the red zone with state constraints.

[1]https://www.youtube.com/watch?v=gmeureiWX5k

The NN controller's result on an L-shaped map is shown in Fig. 4. The tracking performance is satisfactory. The yellow dots illustrate positions where the ModelPlex monitor raises alarms. There are two main categories of abnormal controls: 1) the waypoint is not ahead of the vehicle: this occasionally happens when the vehicle just passes the waypoint, and the new waypoint has not been updated; 2) invalid steering (steering not in direction of goal): this happens when the vehicle's heading is not pointing at the waypoint (the annular section does not cover the waypoint) or the controller is correcting a large deviation mildly. These alarms are not caused by a faulty monitor, but are rather the result of a simplified model that does not account for wheel slip, counter-steering, and corrections of over-steering. They can be mitigated by an improved model in hybrid systems theorem proving, or by better-tuned parameters such as ϵ to allow more tolerance to include additional legitimate behaviors. However, more complex models increase proof effort, while tuning involves tedious trial-and-error efforts. Reachability analysis can help us to estimate the risks of mild violations quantitatively. We observe that no alarm from the reachability analysis is raised since the vehicle stays in the centerline well without any hazard of running outside the track boundary. We conclude that if we only use the ModelPlex monitor, the fallback controller will be frequently invoked for small deviations, while the combined monitor is more robust.

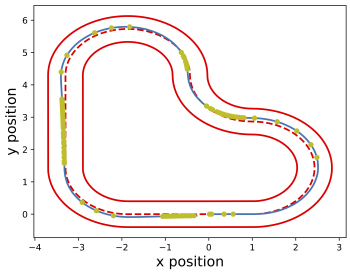
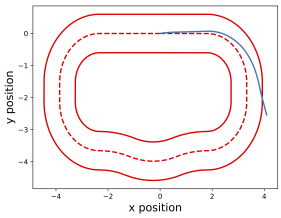


Fig. 4: Using FNN as a tracking controller in the L-shaped map. The vehicle's initial position is $(0, 0)$ and it runs in the counter-clockwise direction.

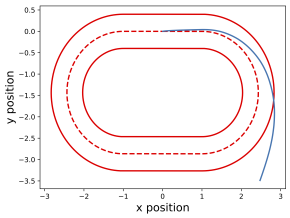
The results using the same FNN controller in the G-shaped and the rounded-square maps are shown in Fig. 5a and Fig. 5b, respectively. The blue trajectories show that the vehicle runs outside the track. The generalization of FNN is not satisfactory because new curvatures of the roads and their corresponding tracking control actions are not included in the training dataset.

This imperfection can be leveraged to demonstrate the effectiveness of our integrated monitor and the fallback controller, see Fig. 6a and Fig. 6b. The green dots are for positions in the green zone and the red dots are for positions in the red zone. When deviations are small, the vehicle is in the yellow zone. However, when deviations are large and the vehicle is in danger of hitting the track boundary, reachability can detect the unsafe control and invoke the fallback controller to recover.

We also test in dynamic obstacle avoidance scenarios with layered planning and control. The local planner generates collision-free trajectories for the vehicle to follow. We expect

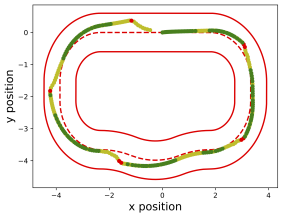


(a) G-shape (FNN)

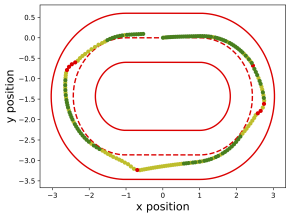


(b) Rounded-square (FNN)

Fig. 5: NN without fallback controller



(a) G-shape (ours)



(b) Rounded-square (ours)

Fig. 6: NN with fallback controller

that any large deviation from the planned trajectory will potentially cause collisions. The final trajectories of the ego vehicle and three dynamic obstacles are shown in Fig. 7. The black, orange, blue, and green trajectories are for the ego-vehicle and three moving obstacles, respectively. The minimum distance between the ego-vehicle and the obstacle in each time step is larger than the vehicle shape - $0.25m$.

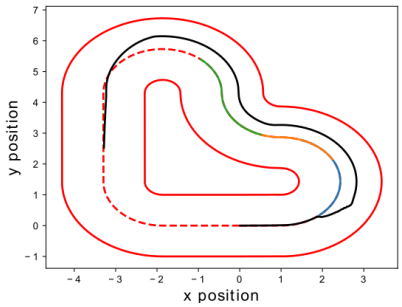


Fig. 7: Trajectories of all vehicles.

When the ego-vehicle is passing by the first obstacle, the fallback controller is invoked. This can have two causes: 1) the vehicle does not follow the planned trajectory closely enough; 2) the planned trajectory is insufficiently separated from the obstacle for even perfect execution to avoid a collision. Our method can handle either of these two cases once the reachable state has any intersection with the obstacle and invoke the fallback controller to leave the red zone.

V. Conclusions

We propose a novel Simplex architecture for safe and resilient waypoint-following for autonomous vehicles controlled by a neural network controller. Our new monitor combines theorem proving and reachability analysis and inspects short-term and long-term safety simultaneously online. The monitor is more robust since it relaxes the violation of long-term safety temporarily, but still offers a solid chance to recover to a safe state with short-term and long-term safety. The fallback controller is based on MPC, which derives the target safe state by a novel sampling approach. The simulation results in path tracking and obstacle avoidance scenarios demonstrate that our proposed framework ensures safety and minimizes interventions from the fallback controller. One limitation is that the recursive feasibility of obtaining a safe state relies on online sampling. One of our ongoing works is safety verification using interval instead of concrete float values for ModelPlex, which is expected to replace the sampling. Another future work is to consider the recursive feasibility of the MPC. Our results are important for the invariant synthesis for recursive feasibility since we determine the terminal safe set.

VI. Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Air Force and DARPA.

References

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

[2] S. Prajna, "Barrier certificates for nonlinear model validation," *Automatica*, vol. 42, no. 1, pp. 117–126, 2006.

[3] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Automatica*, vol. 129, p. 109597, 2021.

[4] O. Maler, "Control from computer science," *Annual Reviews in Control*, vol. 26, no. 2, pp. 175–187, 2002.

[5] I. M. Mitchell, "Comparing forward and backward reachability as tools for safety analysis," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2007, pp. 428–443.

[6] A. Platzer and Y. K. Tan, "Differential equation invariance axiomatization," *Journal of the ACM*, vol. 67, no. 1, pp. 1–66, 2020.

[7] S. Mitsch and A. Platzer, "Modelplex: Verified runtime validation of verified cyber-physical system models," *Formal Methods in System Design*, vol. 49, no. 1, pp. 33–74, 2016.

[8] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[9] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical computer science*, vol. 138, no. 1, pp. 3–34, 1995.

[10] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *International Conference on Automated Deduction*. Springer, 2015, pp. 527–538.

[11] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *International Symposium on Formal Methods*. Springer, 2011, pp. 42–56.

[12] B. Bohrer, Y. K. Tan, S. Mitsch, A. Sogokon, and A. Platzer, "A formal safety net for waypoint-following in ground robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2910–2917, 2019.

[13] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 258–263.

[14] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: a predictive control approach," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, 2019.

[15] A. Claviere, S. Dutta, and S. Sankaranarayanan, "Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 680–688.

[16] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*. IEEE, 2007, pp. 2296–2301.