Reinforcement Learning as an Alternative to Reachability Analysis for Falsification of AD Functions

Tobias Johansson
Safe Vehicle Automation
Volvo Cars
Gothenburg, Sweden
tobias.johansson@volvocars.com

Angel Molina Acosta
Safe Vehicle Automation
Volvo Cars
Gothenburg, Sweden
angel.molina.acosta@volvocars.com

Alexander Schliep
Computer Science and Engineering
Gothenborg University
Gothenburg, Sweden
alexander@schlieplab.org

Paolo Falcone
Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
paolo.falcone@chalmers.se

Abstract

Reachability analysis (RA) is one of the classical approaches to study the safety of autonomous systems, for example through falsification, the identification of initial system states which can under the right disturbances lead to unsafe or undesirable outcome states. The advantage of obtaining exact answers via RA requires analytical system models often unavailable for simulation environments for autonomous driving (AD) systems. RA suffers from rapidly rising computational costs as the dimensionality increases and ineffectiveness in dealing with nonlinearities such as saturation. Here we present an alternative in the form of a reinforcement learning (RL) approach which empirically shows good agreement with RA falsification for an Adaptive Cruise Controller, it can deal with saturation, and, in preliminary data, compares favorably in computational effort against RA. Due to the choice of reward function, the RL's estimated value function provides insights into the ease of causing unsafe outcomes and allows for direct comparison with the RA falsification results.

1 Introduction

Introducing autonomous driving (AD) functions to the general public comes with a great responsibility of ensuring that the AD systems behave safely. Therefore, validation and verification standards [1, 2] have been devised and approaches for testing AD software have been proposed. One such approach based on access to an efficient AD simulation environment is falsification, aiming at identifying examples where an AD system fails to perform according to specifications [3]. This can be achieved by controlling factors, such as the behavior of surrounding road users [4, 5] or sensing errors. Due to the inherent complexity of AD systems, machine learning (ML) based methods have proven useful to find falsification examples. From a safety analysis perspective, these can provide evidence that a control algorithm is unsafe by finding one or more failure cases.

System safety is often assessed by deriving formal safety guarantees with, for example, reachability analysis (RA) [6, 7, 8, 9]. With RA, sets of initial states can be computed for which a system will

Machine Learning for Autonomous Driving Workshop at the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

always remain safe even in the presence of (bounded) disturbances. Computationally, RA scales poorly with the size of the problem, i.e. the dimension of the state-space or the number of constraints imposed on states and inputs. Moreover, RA is limited to systems where dynamics can be described *explicitly* by relatively simple analytical models rather than the complex simulation models more representative of AD simulation environments. Recently, significant progress has been made towards mitigating the scalability issues of RA using tools from ML to approximate RA results. Fisac *et al.* [10] introduced a discount factor to the Hamilton-Jacobi (HJ) equation, for the first time enabling application of methods from reinforcement learning (RL) to approximate the solution to Hamilton-Jacobi reachability (HJR). A closely related work [11] uses findings from periodic activation functions for neural networks to approximate the solution to the partial differential equations that arise in HJR.

In contrast to existing work where results from RA are approximated via RL for the purpose of finding sets of safe states, we instead use RL and RA to determine sets of potentially *unsafe* states, i.e. states for which a system can be driven to failure by iterative application of disturbances. The main contributions in this work are as follows.

- We formulate the problem of falsifying a specification; i.e. finding state-disturbance trajectories for which the system ends in an unsafe or undesired state. We present methods to solve the falsification task based on RA and RL.
- We propose a simple, yet effective, reward function to train an RL agent and solve a falsification task. The reward function is designed to allow for comparison with falsification results from RA.
- For the task of falsifying an Adaptive Cruise Controller, we empirically demonstrate that the estimated value function obtained by training an RL agent using Proximal Policy Optimization closely matches the sets obtained via RA.
- In preliminary data, we demonstrate that RL-algorithms may serve as an alternative solution for linear reachability analysis when performing falsification of systems with large state-dimension.

In Section 2 we formulate the problem and give an overview of RA and RL. The methodology for computing falsification examples and comparing results from RA and RL is introduced in Section 3. Section 4 describes the simulation setup in detail, and simulation results are presented in Section 5. We end with stating conclusions and suggestions for future work in Section 6.

2 Background and Problem Description

A common approach to system verification identifies all states for which a system is guaranteed to be safe; i.e., states from which the system cannot be driven to failure regardless of any applied (bounded) disturbance. In contrast, we focus on identifying unsafe or undesired states from which it is possible to drive the system to failure under some carefully chosen sequence of disturbances. We refer to this task as falsification. Following a brief introduction of RA and RL, the next section will give a formal description of the problem.

Reachability analysis for systems with linear dynamics. Reachability analysis comprises techniques for analyzing the evolution of a system either forward or backward in time. Forward reachability analysis determines which states a system may end up at when starting from a given set of initial states after some time interval; backward reachability identifies sets of states from which a system can end up in a given set of states. In this work we adopt the backwards RA approach over the forwards approach as the former allows to identify as large as possible sets of states that can evolve into the set of interest.

Methods for backwards RA are tailored to the type of model describing the system. Here, we focus on backwards RA for discrete-time, linear systems, i.e. models consisting of linear equations where the time variable is discretized, and constraints on the states and disturbances given as polyhedral sets. For such systems, methods to compute backward reachable sets subject to linear constraints are well studied [12, 13, 14]. These methods require operations on convex sets such as projection to lower dimensions and redundant constraint elimination. A major drawback of these methods is that the complexity of computing reachable sets can grow exponentially with the dimension of the state. Nevertheless, various authors have obtained useful results on verification by means of RA for systems with small state dimension [6, 7].

If a model of the system is available in the form of differential equations describing the time evolution of the system states, RA can solve a falsification task as follows: specify a set of states that violate or falsify a certain specification, e.g. states representing an unsafe condition, and then apply backward reachability to determine initial states and a sequence of disturbances for which the system will end up in the specified set of falsifying states.

Reinforcement learning. RL deals with problems of sequential decision-making under uncertainty. An agent acts in an environment and receives feedback in the form of a new state and a reward that results from the action taken by the agent. The environment is often described using a discrete-time Markov decision process (MDP) defined by a quintuple consisting of a set of states $\mathcal{X}$, an initial state $x_0 \in \mathcal{X}$, a set of actions $\mathcal{W}$, a transition probability $P(x_{k+1}|x_k, w_k)$ and a reward probability $P(r_{k+1}|x_k, w_k)$. The goal of the agent is to maximize the total expected reward by tuning its policy $\pi(x)$, i.e. how actions are chosen depending on the current state.

The value $V_\pi(x)$ of a policy π at state $x \in \mathcal{X}$ is defined as the total expected (discounted) reward starting from x and following policy π,

$$V_\pi(x) = \mathbb{E}_{w_k \sim \pi(x_k)} \left[\sum_{k=0}^{N} \gamma^k r(x_k, w_k) \mid x_0 = x \right], \tag{1}$$

where $\gamma \in [0, 1)$ is a discount factor, $r : \mathcal{X} \times \mathcal{W} \to \mathbb{R}$ is the reward function, and N is the possibly infinite horizon. RL has been successfully applied to a wide range of problems such as playing Go and Atari games [15, 16] and for motion planning in real-world environments [17, 18]. To apply RL for falsification one has to choose which disturbances to control as well as a suitable reward function adapted to the falsification task at hand. Examples of RL being used for falsification purposes includes adaptive stress testing [19, 20] and related approaches [4, 21].

Other falsification methods. In addition to RL and RA several other methods—e.g. simulated annealing [22], ant-colony optimization [23] and rapidly-exploring random trees (RRT) [24, 5]—have been used for falsification, see the comprehensive survey by Corso *et al.* [3]. Closely related to the application discussed here is [5], where the authors propose a method for falsification of Adaptive Cruise Control (ACC) systems based on RRTs. The authors apply RRTs to explore backward and forward in time and thus arrive at falsification examples that reveal flaws in ACC systems. The authors do not discuss how well their solution cover the set of all possible initial conditions that can be driven to a failure, an aspect that is addressed in our work.

2.1 Problem description

The falsification task we address is to determine for which initial conditions a system can be driven to violate a given specification in the presence of bounded additive disturbances. The system analyzed in this work is described by the generic difference equation

$$x_{k+1} = Ax_k + Bu_k + Ew_k, \tag{2}$$

where $k \in \mathbb{N}$ denotes the sampling instant; $x_k \in \mathbb{R}^n$ is the state, $u_k \in \mathbb{R}^m$ is the control input, and $w_k \in \mathbb{R}^p$ is a disturbance; and matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $E \in \mathbb{R}^{n\times p}$. The state, input, and disturbance are constrained respectively as $x_k \in \mathcal{X} \subseteq \mathbb{R}^n$, $u_k \in \mathcal{U} \subseteq \mathbb{R}^m$, $w_k \in \mathcal{W} \subseteq \mathbb{R}^p$. We conduct our falsification analysis on a system with dynamics as in (2) subject to control law

$$u_k = f(x_k, w_k), \tag{3}$$

with $f(\cdot)$ being a state-feedback law that complies to the input bounds, i.e. $f(x_k, w_k) \in \mathcal{U}$ for $x_k \in \mathcal{X}$, $w_k \in \mathcal{W}$. In precise terms, the falsification task is stated as follows:

Problem 1. *Determine a non-empty set of states $S_{\text{init}} \subset \mathcal{X}$ such that for each point $x_0 \in S_{\text{init}}$, a sequence of admissible disturbances can be chosen driving the system starting at x_0 into a set $S_{\text{fail}} \subset \mathcal{X}$, where a given specification is violated, in at most N time steps. That is, $x_0 \in S_{\text{init}} \implies \exists w_0, \cdots, w_{j-1} \in \mathcal{W}$ such that $x_j \in S_{\text{fail}}$, for some integer $j \leq N$, where x_k is given by* (2)-(3) *for $k \in \{1, \ldots, j\}$.*

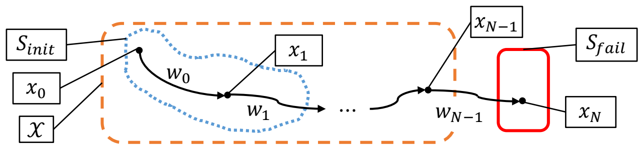

Figure 1: A schematic view on Problem 1.

Solving Problem 1 requires finding a set of initial conditions S_{init} and admissible disturbance sequences that drive the system into S_{fail} (see Figure 1 for a visual intepretation of Problem 1). The primary objective of this paper is to compare sets S_{init} determined using RL against a set S^*_{init} obtained via RA.

3 Methodology

Falsifying with reachability analysis. This section describes the methodology used to compute reachable sets and how these can be applied to falsification. We borrow concepts described in [13, chp. 11] on reachable and controllable sets.

For systems with closed-loop dynamics of the form

$$x_{k+1} = Ax_k + Ew_k + a, \tag{4}$$

where a is the constant affine term, Problem 1 can be solved exactly by computing reachable sets, as long as the sets $S_{\text{fail}}, S_{\text{safe}}, \mathcal{X}, \mathcal{W}$ presented in section 2.1 are convex polyhedra.

Given the system (4) and a set $S \subset \mathcal{X}$ the k-steps controllable set $\mathcal{C}_k(S)$ is defined recursively as

$$\mathcal{C}_k(S) := \{x \in \mathcal{X} : \exists w \in \mathcal{W},\ Ax + Ew + a \in \mathcal{C}_{k-1}(S)\}, \tag{5}$$

with $k \geq 1$ and $\mathcal{C}_0(S) := S$. Set $\mathcal{C}_k(S)$ contains all states in $\mathcal{X}$ for which an admissible disturbance-sequence exists that steers the state into the set S in k-steps or less. The set of admissible disturbances $\Phi_k(x, S)$ associated to a state $x \in \mathcal{C}_k(S)$ is defined as

$$\Phi_k(x, S) := \{w \in \mathcal{W} :\ Ax + Ew + a \in \mathcal{C}_{k-1}(S),\ x \in \mathcal{C}_k(S)\}. \tag{6}$$

Sets $\mathcal{C}_k(\cdot)$ and $\Phi_k(\cdot, \cdot)$ can be computed exactly via operations on convex sets, the details of which can be found in [13].

To solve Problem 1 via RA for systems as in (4), we compute sets $\mathcal{C}_k(S_{\text{fail}})$ and $\Phi_k(x, S_{\text{fail}})$ for $k = 1, \cdots, N$. The set $\mathcal{C}_k(S_{\text{fail}})$ contains all initial states $x_0 \in \mathcal{X}$ for which an admissible disturbance sequence exists that causes, in k steps or less, a violation of the specification, i.e. achieving $x_j \in S_{\text{fail}}$ for some $j \leq k$. Hence, the exact solution to Problem 1 for systems with dynamics as in (4) is given by $S^*_{\text{init}} = \bigcup_k \mathcal{C}_k(S_{\text{fail}})$ with $k = 1, \cdots, N$. To retrieve a falsification example from the RA results, we choose an initial condition $x_0 \in \mathcal{C}_N(S_{\text{fail}})$ and apply disturbances $w_0 \in \Phi_N(x_0, S_{\text{fail}}), \cdots, w_{N-1} \in \Phi_1(x_{N-1}, S_{\text{fail}})$ to achieve $x_N \in S_{\text{fail}}$. See also Figure 1.

Falsifying with RL. To implement RL for falsification purposes there is a wide range of viable algorithms, including previously proposed approaches using actor-critic policy gradient methods such as Trust Region Policy Optimization (TRPO) [20], Proximal Policy Optimization (PPO) [25] and Advantage Actor Critic (A2C) [4]. When action spaces are discrete, another possibility is Deep Q-learning, similar to what has been explored for finding safe sets in [10]. We implement RL for falsification using PPO with Generalized Advantage Estimation (GAE) [26], mainly due to its proven success for continuous control tasks.

In practice, the system equations (2) must be implemented in a simulation environment that provides feedback to the RL algorithm for each sampled disturbance vector in terms of rewards. The chosen reward function is paramount for finding falsification examples, and may be closely connected to the considered falsification problem. Existing literature concerning falsification and variants thereof proposes both handcrafted rewards [19, 20] and more general rewards based on temporal logic [27]. We propose a reward function enabling identification of potentially unsafe states directly from an estimated value function $\hat{V}(x)$ obtained from a critic network. For optimal identification of S_{init} a binary reward function for which a reward is given only if S_{fail} is reached would be natural, since it directly reflects the probability of reaching S_{fail} within the given horizon. Unfortunately, the sparsity of the reward signal in the binary case often hinders the agent to learn a good policy, Hence, reward shaping is frequently used in practice, thus we do likewise.

Comparing RA and RL. One of the key contributions in this work is a comparison of falsification results obtained via RA and RL, respectively. To this end, we compare the set S_{init} obtained from RL with the sets $\mathcal{C}_k(S_{\text{fail}})$ obtained via RA. Since we both obtain an action policy and a value function estimate during RL training with PPO, we can directly evaluate the estimated value function for any state $x \in \mathcal{X}$. Moreover, by designing the reward function to output positive values only if the system enters S_{fail} within k time steps and output negative values otherwise, a positive value function estimate suggests that the learned policy is likely to drive the system into a failure in at most k steps. We perform the comparison of S_{init} with $\mathcal{C}_k(S_{\text{fail}})$ by evaluating the learned estimate of the value function (VF) for a large number of states and classify the states into groups with positive or negative, respectively, estimated VF. We then check how the groups correspond to $\mathcal{C}_k(S_{\text{fail}})$ using both visual inspection and by computing binary classification metrics, such as the rate of false negatives.

4 Simulation Experiment

To investigate how well RL can approximate RA sets we restrict our analysis to systems where RA admits an exact and tractable solution. We consider an adaptive cruise control (ACC) setting where the ego vehicle should follow a target vehicle and keep a given time-gap. The falsification task is to generate acceleration inputs to the target vehicle and to inject sensor errors for the measured inter-vehicle gap and target vehicle's velocity as seen by the ego vehicle.

We use an ACC model from the literature [28, chp. 6]. The system dynamics are described by the following equation

$$\dot{x}_t = \begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} a_t^0 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} a_t^1, \tag{7}$$

where subscript t denotes time. The state vector is $x_t = \left(\delta_t \; v_t^0 \; v_t^1\right)^T$, where δ_t is the difference between the position of the front bumper of the ego vehicle, denoted as p_t^0, and the position of the rear bumper of the target vehicle, denoted as p_t^1, so $\delta_t := p_t^0 - p_t^1$ (see Figure 2); v_t^0 and v_t^1 denote the forward velocity of the ego and target respectively; a_t^0 is the acceleration of the ego; a_t^1 is the acceleration of the target. In what follows, we drop the explicit dependency on t.

The ACC requests an acceleration a_{req} for the ego vehicle as given by the following expression [28]:

$$a_{\text{req}} = -\tfrac{1}{h}\left(v^0 - \hat{v}^1 + k_p\left(\hat{\delta} + L + hv^0\right)\right), \tag{8}$$

where h is the desired time-gap, k_p is the proportional gain, and L is the spacing desired at standstill between the ego and the target; variables $\hat{v}^1 = v^1 + e^v$ and $\hat{\delta} = \delta + e^\delta$ are erroneous measurements of v^1 and δ respectively, where e^v and e^δ are sensor errors. The ACC policy in (8) aims at keeping a constant time-gap h between ego and target. For simplicity, the parameters in (8) are chosen as $h = k_p = L = 1$, values for which (8) achieves stable closed-loop dynamics for (7).

Setting a^0 in (7) equal to a_{req} from (8) and applying a zero-order-hold discretization, we arrive at a discrete-time model of the form (4) with state $x_k = \left(\delta_k \; v_k^0 \; v_k^1\right)^T$ and disturbance $w_k = \left(a_k^1 \; e_k^v \; e_k^\delta\right)^T$. The discretization is performed with a sampling interval $T_s = 0.1\ s$. The discrete-time linear model is used for the reachability analysis. To increase realism, the ego acceleration is limited to $a^0 \in [-0.8g, 0.2g]$ where $g = 9.81$ m/s^2 is the gravitational acceleration. The allowed disturbances are restricted to $\left(a^1 \; e^v \; e^\delta\right)^T \in \mathcal{W} = [-0.8g, 0.2g] \times [-0.5, 0.5] \times [-0.5, 0.5]$, with units m/s and m for e^v and e^δ respectively.

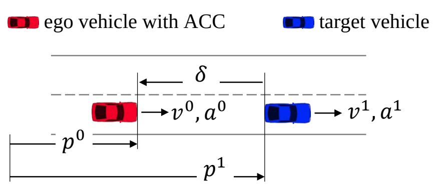

Figure 2: Illustration of the ACC simulation setting.

Remark 1 (Input saturation). *Introducing bounds on the control input, i.e. the ego vehicle's acceleration a^0, presents a challenge for the RA method because the closed-loop dynamics subject to saturation become piecewise affine: different linear (affine) dynamics are active in different regions of the state-space. Hence, computing reachable sets requires inspecting if the computed sets lie across regions with different dynamics and split the sets accordingly before stepping into the recursive computation. This leads to a branching effect that can easily become intractable. To overcome this issue, we restrict the reachability analysis to the region of the state-space where the requested acceleration a_{req} in* (8) *remains within the specified bounds for all possible values of the disturbance w such that the dynamics* (4) *remain valid.*

In line with Remark 1, when conducting RA computations we handle constraints on a^0 by restricting the set of admissible states to

$$\tilde{\mathcal{X}} = \{x \in \mathcal{X} : \ a_{\text{req}} \in [-0.8g, 0.2g], \ \forall w \in \mathcal{W}\}\,, \tag{9}$$

with a_{req} as in (8). Setting the set of admissible states to $\tilde{\mathcal{X}}$ poses a limitation to the falsification examples that can be obtained with RA and it also limits the comparison we can provide between RL and RA. Nevertheless, the falsification examples obtained via RA under said limitation still offer a good benchmark to compare the RL results.

Remark 2 (Exact solution to Problem 1). *In the following, when we refer to the exact solution to Problem 1 we are in fact referring to a set S^*_{init} containing all states $x \in \tilde{\mathcal{X}}$ that can be steered into S_{fail} without leaving $\tilde{\mathcal{X}}$ by applying an admissible disturbance sequence. Such a set S^*_{init} can be computed exactly via RA as described in Section 3 and is a subset of a possibly larger set $\bar{S}^*_{\text{init}}$ containing all states $x \in \mathcal{X}$ that can be steered into S_{fail} by evolving, inside $\mathcal{X}$, trajectories for which the constraints on a^0 can become active.*

The falsification task consists in steering the state into a set that represents a rear-end collision between the ego and the target vehicle. Before reaching collision, the vehicles should drive with non-negative velocity, i.e. reversing is not allowed. To this end, we define the following sets

$$\mathcal{X} = \left\{x \in \mathbb{R}^3 : v^0 \geq 0, v^1 \geq 0\right\}, \tag{10}$$

$$S_{\text{safe}} = \{x \in \mathcal{X} : \delta < 0\}\,, \tag{11}$$

$$S_{\text{fail}} = \{x \in \mathcal{X} : \delta \geq 0\}\,. \tag{12}$$

Notice that $\delta \geq 0$ corresponds to the ego vehicle's front bumper being ahead of the target vehicle's rear bumper, i.e. a rear-end collision.

We use proximal policy optimization (PPO) [29] with generalized advantage estimation (GAE) [26] from RLlib [30] to train the RL agent. The policy π_θ is parameterized by a two-layer feed-forward neural network with 64 units and tanh activations in each layer which generates concentration parameters α, β for a three-dimensional Beta(α, β) distribution scaled to the allowed disturbances. The Beta distribution naturally handles bounded action spaces and has been proven successful for continuous control tasks [31]. We use a separate network with the same size as the policy network to estimate the value function. We define the reward function as

$$r_k = \begin{cases} 100 & \text{if } x_k \in S_{\text{fail}},\ k \leq N, \\ -10|\delta_k| & \text{if } x_k \notin S_{\text{fail}},\ k = N, \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where δ_k is defined as above at time step k. Similar to [20], we include the gap $|\delta_k|$ in the reward for non-fail scenarios as this accelerates the learning process because an increasingly negative reward is given the farther apart the vehicles are at the end of the simulation horizon if no failure occurs.

Since (13) depends on the simulation time-step we have to include time information in either the policy, e.g. by using an RNN, or in the state representation given to the policy network. We chose the latter option partly to avoid introducing extra complexity, and partly because adding time to the state representation results in a problem specification more closely related to the one proposed in RA. In particular, instead of specifying the current time step in the state we give the remaining number of time steps before a collision should occur, i.e. the remaining simulation horizon. Doing so allows for comparing S_{init}, as estimated by the value function network, with the exact set obtained from backwards RA on a per-timestep basis.

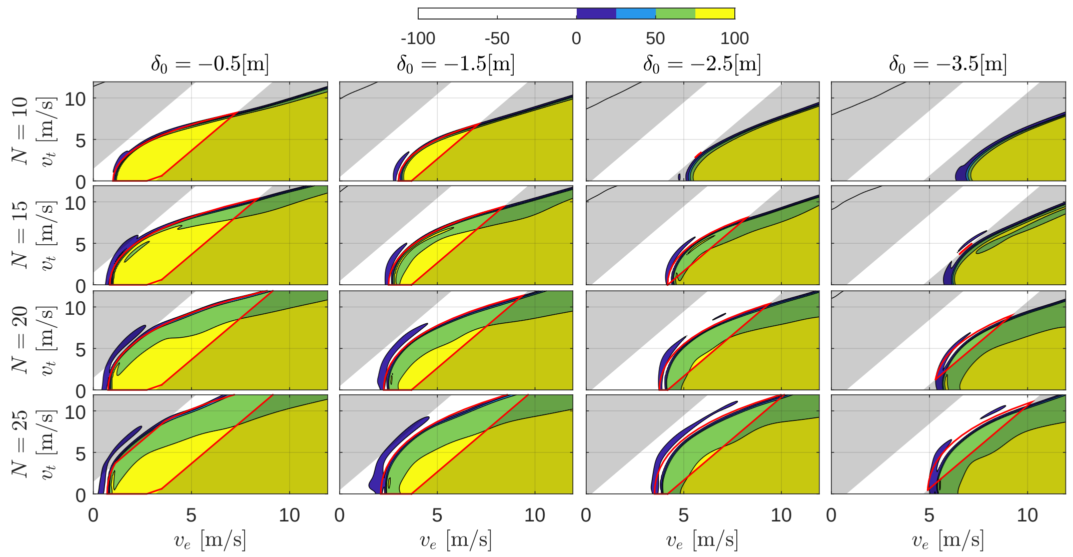

Figure 3: In this figure, the result of evaluating the predicted value function on a uniform grid of 200×200 points for four distinct values δ_0 and four values of simulation horizon N is displayed. The red solid outline indicates the boundary of a controllable set obtained via RA, which is sliced at a constant value of δ. The regions shaded in gray corresponds to the states for which the ACC rule (8) requests accelerations outside the specified bounds; i.e., those not amenable to analysis with RA. The colored regions indicate the value of the estimated value function.

5 Results

As described in the previous section, we compute controllable sets $\mathcal{C}_k(S_{\text{fail}})$ for $k = 1, \ldots, 25$ with the help of the MPT Toolbox in MATLAB [32]. The computation of RA sets took around 2 seconds on a personal computer with an 8-core Intel processor running at 2.6 GHz. The sets $\mathcal{C}_k(S_{\text{fail}})$ increase in complexity as k increases. For instance, set $\mathcal{C}_1(S_{\text{fail}})$ is described by 5 inequalities while $\mathcal{C}_{20}(S_{\text{fail}})$ requires 40 inequalities. We trained the RL agent with the PPO implementation from RLlib on a 12-core 3.4-GHz machine for 10 million steps, taking around 2.5 hours. However, in almost all cases the average reward stabilized after around 600,000 steps, taking approximately 10 minutes. The initial states were uniformly sampled as $(\delta,\ v^1,\ v^0,\ N) \in [0, 5] \times [0, 12] \times [0, 12] \times \{1, 2, \ldots, 30\}$.

In Figure 3 we show level sets of the predicted value function, cf. equation (1)), learned by the RL agent for four values of δ_0, four simulation horizons N, and a 200×200 uniform grid of ego and target vehicle velocities. The red outline in the subfigures indicate the reachable sets $\mathcal{C}_k(S_{\text{fail}})$ sliced at the given value δ_0 and with $k = N$. The gray shaded area correspond to regions of the state-space where the requested ego vehicle's acceleration a_{req} is outside the specified bounds. Notice that the level sets for the estimated value function take negative values mainly outside of the region covered by the RA sets. This is a desired result and is a consequence of the chosen reward function (13); moreover, this result indicates that it is possible to approximate the RA sets using the predicted value function learned via RL. The results in Figure 3 were obtained training an RL agent with discount factor $\gamma = 0.99$, learning rate 10^{-4} and a clipping parameter of 0.3 for PPO. As pointed out in Remark 1, our RA analysis is restricted to the region $\tilde{\mathcal{X}}$ in which the acceleration request for the ego vehicle remains within bounds; see the strip that is not shaded in Fig. 3. However, the RL agent is trained with the possibility of choosing actions leading to a_{req} being outside the bounds. Before applying a_{req} to the ego vehicle's motion the request is clipped to the saturation bounds. In this way, the RL agent learns how to choose disturbances to steer the state into S_{fail} even for those state-trajectories that travel through the saturation regions, something that is difficult to achieve with RA. According to the estimated value function, the RL agent is more likely to successfully steer the state into S_{fail} by choosing disturbances that cause input saturation.

To offer a quantitative comparison of the results shown in Fig. 3, we propose the following evaluation metric. From the uniform grids used to sample the value function estimates, we select all points that belong to the RA set $\mathcal{C}_k(S_{\text{fail}})$ (the region inside the solid red outline in Fig. 3). Then we compute a

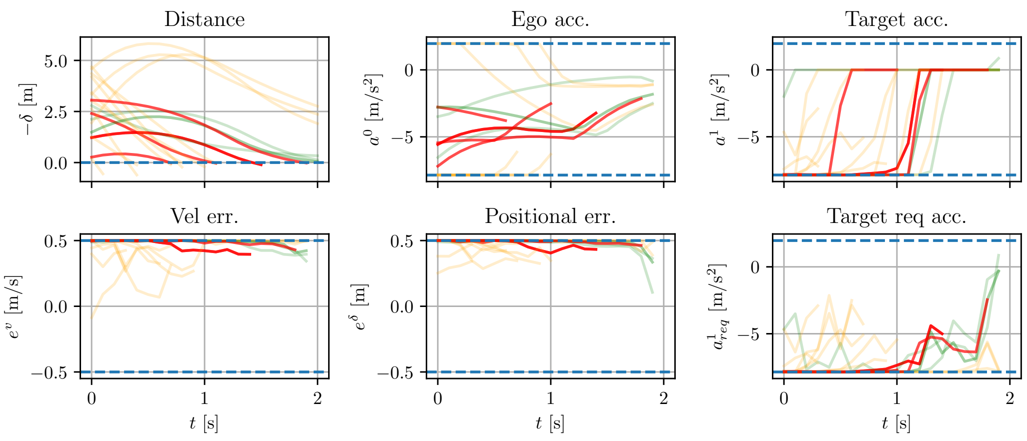

Figure 4: Simulated state-disturbance trajectories obtained from the trained RL agent using a simulation horizon of $N = 20$ timesteps. The dashed blue lines indicate the bounds specified for the signal. The red lines are trajectories that enter the set S_{fail} without the ego acceleration reaching its saturation bounds. Green lines are trajectories that do not reach S_{fail} and for which the ego acceleration remains within bounds. Orange lines are trajectories for which the ego acceleration reaches saturation.

metric ρ that is the number of grid-points inside the RA sets with negative predicted VF divided by the total number of grid-points inside the RA set. The metric ρ can be interpreted as a measure of false-negatives in the RL results. Values of ρ close to 0 indicate that the RA sets are well covered by the estimated value function. The values of ρ for the grids used in Figure 3 are reported in Table 1. Empty entries in Table 1 correspond to values of the pair (N,δ_0) for which the chosen grid has less than 10 points inside the corresponding RA set. The ratio of false-negatives is low for most values of (N,δ_0). It is only for $\delta_0 = -3.5$ that we see values of ρ not close to zero, which indicates that there is room for improvement when training the RL agent.

Figure 4 depicts example disturbance-trajectories generated by the RL agent for a selection of initial conditions. The upper left plot shows how the gap between ego and target vehicles evolves. The bottom row displays disturbance sequences chosen by the agent. Since we do not allow for negative velocities, the requested accelerations are clipped to ensure that this does not occur. This explains why the requested target acceleration (right bottom figure) differs from the applied target acceleration (right top). As soon as a crash occurs the simulation ends, explaining the different trajectory lengths in the figure.

Preliminary results on a higher-dimensional problem. While the running time for our RL implementation of the simple ACC example introduced above is significantly higher than for the RA counterpart, the unfavorable scaling of RA poses computational challenges. Indeed, we have conducted initial experiments on a model describing a lateral vehicle controller (briefly explained in the Appendix) where the RA approach employed in this article became intractable due to the

Table 1: Numerical values for the metric ρ, which can be interpreted as false-negatives rate, for the same data shown in Fig. 3. The ρ values indicate good agreement between RA and RL for $\delta_0 = -0.5$, -1.5, and -2.5.

N	δ_0			
	-0.5	-1.5	-2.5	-3.5
10	0.003	0.009	–	–
15	0.008	0.016	0.08	–
20	0.0016	0.019	0.066	0.23
25	0.0013	0.02	0.099	0.17

dimensionality of the state space. There were in total 11 dimensions compared to 3 for the ACC controller. We aborted computations with RA when after 24 hours only 3 reachable sets, i.e. $\mathcal{C}_k(S_{\text{fail}})$, with $k = 1, 2, 3$ had been computed which provides falsification trajectories lasting at most 0.3 seconds. Despite the large dimension, it is still possible to find falsification examples using RL without incurring unreasonable increases in running time. For the experiment in the appendix we executed one million timesteps in the simulator during training, taking around 30 minutes in total. While the value function evaluation is relatively cheap (a forward pass in the value function network for each initial state being evaluated) the complexity also grows with the dimensionality of the space. To which extent this can be mitigated remains to be explored. A complete analysis of this simulation scenario will be left for a journal version of this submission. This will include finding approaches to assess the coverage of initial conditions that can be falsified using RL, and a possible iterative approach to comparing the falsification results from RA and RL by reducing the dimensionality through removing e.g. sensor errors from the disturbance definition or replacing the controller with a simpler alternative.

6 Conclusion and future work

We provide a comparison of falsification to system safety using reinforcement learning against results from reachability analysis for linear systems. The results show that it is possible to apply RL to aproximate the sets of unsafe states identified via RA. In particular, we have seen that by augmenting the state representation for the RL agent with the remaining simulation horizon and using a time dependent reward function, it is possible to identify states that can fail within different simulation horizons, which parallels the results from backward RA.

Here we have focused on comparing the initial conditions that can be driven into a failure set identified by RL and RA. Evaluating the thoroughness of RL falsification results in terms of the diversity of disturbance trajectories that achieve falsification could be an aspect worth exploring. The simulation example presented in this article was deliberately chosen to be simple. The reason for this was to allow for a clear and comprehensible comparison between falsification using both RA and RL. That falsification examples can be found via RL for a simple model is an expected result. The main takeaway from our work is the assessment of RL results against the exact solution given by RA and the fact that good coverage can be achieved.

The computational requirements for training the RL agent for the ACC example considered in this article are significantly higher than what is needed for RA. However, initial results from investigating an 11-dimensional lateral control model, for which RA becomes intractable, suggests that RL can find solutions for more complex systems without significantly increased computational effort. These results are consistent with previous analyses of the scalability of RA to higher-dimensional problems. A thorough investigation of this is left for future work.

Besides comparing RA and RL for higher-dimensional systems, we have identified various interesting aspects to explore in future works. One example is the evaluation of different reward functions that both facilitate efficient learning and identification of potentially unsafe states directly from the value function. Many methods for falsification of Signal Temporal Logic specification use some form of robustness score as an objective for optimization [33]. This aspect can be adapted as a reward in RL for falsification as well, and a deeper exploration and analysis is likely to prove useful in analyzing safety of more complex control systems. Another extension to our work is comparing RL falsification results against results from forward reachability analysis, which might be suitable for systems with higher state-dimension.

References

[1] ISO/PAS. 21448:2019 road vehicles — safety of the intended functionality. Standard, International Organization for Standardization, Geneva, Switzerland, Jan 2019.

[2] ISO. 26262:2018 road vehicles — functional safety. Standard, International Organization for Standardization, Geneva, Switzerland, Dec 2018.

[3] Anthony Corso, Robert J. Moss, Mark Koren, Ritchie Lee, and Mykel J. Kochenderfer. A Survey of Algorithms for Black-Box Safety Validation. *Retrieved from http://arxiv.org/abs/2005.02979*, 2020.

[4] S. Kuutti, S. Fallah, and R. Bowden. Training adversarial agents to exploit weaknesses in deep control policies. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 108–114, 2020.

[5] Markus Koschi, Christian Pek, Sebastian Maierhofer, and Matthias Althoff. Computationally efficient safety falsification of adaptive cruise control systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2879–2886. IEEE, 2019.

[6] Roozbeh Kianfar, Paolo Falcone, and Jonas Fredriksson. Safety verification of automated driving systems. *IEEE Intelligent Transportation Systems Magazine*, 5:73–86, 2013.

[7] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[8] Jonas Nilsson, Jonas Fredriksson, and Anders C.E. Ödblom. Verification of collision avoidance systems using reachability analysis. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, volume 19, pages 10676–10681, 2014.

[9] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, volume 2018-January, pages 2242–2253, 2018.

[10] Jaime F. Fisac, Neil F. Lugovoy, Vicenc Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. Bridging Hamilton-Jacobi safety analysis and reinforcement learning. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May, pages 8550–8556, 2019.

[11] Somil Bansal and Claire Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. *arXiv preprint arXiv:2011.02082*, 2020.

[12] Franco Blanchini and Stefano Miani. *Set Theoretic Methods in Control Second Edition*. Birkhäuser, Switzerland, 2014.

[13] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[14] Jonathan E. Gayek. A survey of techniques for approximating reachable and controllable sets. In *Proceedings of the IEEE Conference on Decision and Control*, volume 2, pages 1724–1729, 1991.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[17] Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059, 2018.

[18] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, 2019.

[19] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Adaptive stress testing with reward augmentation for autonomous vehicle validation. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 163–168. IEEE, 2019.

[20] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

[21] Xin Qin, Nikos Aréchiga, Andrew Best, and Jyotirmoy Deshmukh. Automatic testing and falsification with dynamically constrained reinforcement learning. *arXiv preprint arXiv:1910.13645*, 2019.

[22] Arend Aerts, Bryan Tong Minh, Mohammad Reza Mousavi, and Michel A. Reniers. Temporal logic falsification of cyber-physical systems: An input-signal-space optimization approach. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 214–223, 2018.

[23] Yashwanth Singh Rahul Annapureddy and Georgios E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 91–96, 2010.

[24] Cumhur Erkan Tuncali and Georgios Fainekos. Rapidly-exploring random trees for testing automated vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 661–666, 2019.

[25] M. Koren and M. J. Kochenderfer. Adaptive stress testing without domain heuristics using go-explore. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020.

[26] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[27] Xiao Wang, Saasha Nair, and Matthias Althoff. Falsification-based robust adversarial reinforcement learning. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 205–212. IEEE, 2020.

[28] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2017.

[31] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *International conference on machine learning*, pages 834–843. PMLR, 2017.

[32] M. Kvasnica, P. Grieder, and M. Baotic. Multi-Parametric Toolbox (MPT). *Available at http://control.ee.ethz.ch/m̃pt/*, 2004.

[33] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s), May 2013.

[34] Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control*. Wiley, 2001.

A An example with higher dimensionality: lateral control of a single-track bicycle model

In contrast to RA we do not expect and exponential growth in computational efforts as the dimension of the state space increases when using reinforcement learning to solve the falsification task considered in this paper. We have started to explore a more complex control example in which the disturbance is the road curvature parameter along with sensor errors to falsify a controller attempting to keep a vehicle close to the center of the road. We use a single-track bicycle model system (see [28] for further details) described by the following update equations

$$\begin{aligned} x_t &= Ax_k + B\delta_k + E\dot{\psi}_k^{ref}, \\ y_k &= Cx_k + v_k, \end{aligned}$$

where $x_k = [e_k^1, \dot{e}_k^1, e_k^2, \dot{e}_k^2]^T$ is the state vector with e_k^1 the lateral offset of the vehicle with respect to the center of the road and e_k^2 the heading offset of the vehicle with respect to a tangent along the center of the road. The control input is the steering angle of the front wheels δ_k and the disturbance is the reference yaw rate $\dot{\psi}_k^{ref}$ associated to driving on a road with a certain curvature at a constant forward speed. The output vector is $y_k = [\tilde{e}_k^1, \tilde{e}_k^2]^T$ which contains noisy measurements of e_k^1, e_k^2 resulting from the additive sensor noise v_k. We design a dynamic output-feedback controller for the input δ_k by means of a standard Linear Quadratic Gaussian regulator with integral action on e_k^1 [34] as follows

$$\begin{aligned} \delta_k &= K\hat{x}_k, \\ \hat{x}_{k+1} &= A_e\hat{x}_k + B\delta_k + E\dot{\psi}_k^{ref} + Fy_k, \end{aligned}$$

where $\hat{x}_k$ is the controller internal state.

RL implementation The above update equations were implemented in Python and RL training was initialized using the PPO implementation from RLLib. We used PPO with a diagonal Gaussian action distribution, i.e. disturbances $w = [\dot{\psi}_k^{ref}, v_k^1, v_k^2]$ are sampled from $w \sim \mathcal{N}(w; \mu, \Sigma)$ where the mean vector $\mu \in \mathbb{R}^3$ and the diagonal of the covariance matrix $\Sigma \in \mathbb{R}^{3\times 3}$ are given by the action policy π_θ. We allow for sampling outside the specified disturbance bounds but before applying the disturbances to the system update equation, w is clipped to the specified bounds.

The reward function we used is given by

$$r_k = \begin{cases} 1000 & \text{if } x_k \in S_{\text{fail}},\ k \le N, \\ -\exp(\exp(\min(e_1 - e_1^{\text{lb}}, e_1^{\text{ub}} - e_1))) & \text{otherwise,} \end{cases} \tag{14}$$

where e_1^{lb} and e_1^{ub} represent the lower and upper limit of the allowed lateral offet. The second case in (14) was devised to help the agent learn to steer the vehicle close to the boundaries of the allowed lateral offset. Since we provide reward feedback in every step of the simulator, it is important to scale the penalty depending on the maximum simulation horizon N and the limits of the allowed lateral error to make sure that a positive return always represents a crashing scenario.

In contrast to the ACC example in the main text we have only trained the RL agent starting from a single initial condition $x_0 = 0, \hat{x}_0 = 0$. Moreover, we have not yet added the remaining simulation horizon to the state representation. Figure 5 shows state and disturbance trajectories for successful falsification examples. The agent learns to generate periodic signals for both the desired yaw rate and the lateral offset measurement error. This results in winding roads for which the controller cannot keep the lateral offset within the specified bounds. For this system the falsifying disturbance sequences are more complex than those observed in the ACC example.

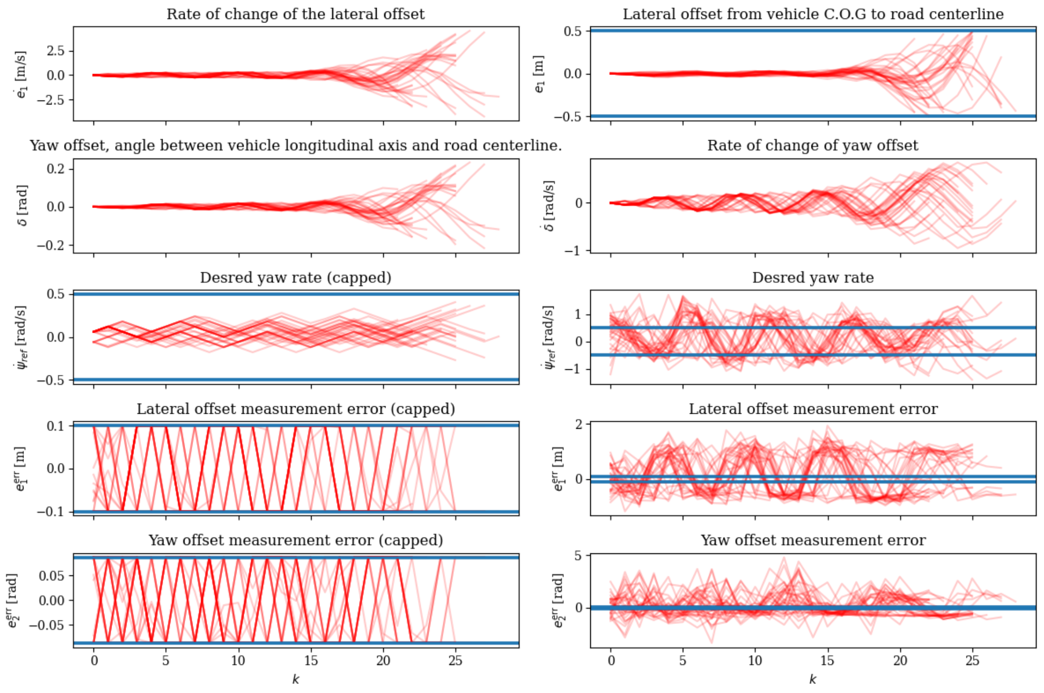

Figure 5: Example trajectories of states and disturbances when using the trained RL agent for disturbance generation. Starting from the zero state, the RL agent is able to generate disturbances that cause the ego vehicle to deviate from the center of the road further than allowed by the specifications. In this example, a failure is achieved for all generated trajectories.