EPiC Series in Computing

Volume 80, 2021, Pages 184–194

8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)

Verification of Collision Avoidance for CommonRoad Traffic Scenarios

Niklas Kochdumper, Philipp Gassert, and Matthias Althoff

Technische Universität München,
Department of Informatics,
Munich, Germany
{niklas.kochdumper,philipp.gassert,althoff}@tum.de,

Abstract

We propose a benchmark for the verification of autonomous vehicles. By considering different traffic scenarios from the CommonRoad database, we obtain several thousands of different verification tasks, where the verification problem is to prove that the considered tracking controller safely follows a given reference trajectory despite disturbances and measurement errors. The dynamic of the car is described by a nonlinear kinematic single-track model. Since the feedback matrix for the tracking controller is time-varying, the dynamic of the controlled system changes constantly. Because of this, the proposed benchmark is well-suited to evaluate how robustly reachability tools can handle changing system dynamics.

1 Introduction

Currently, the performance of verification approaches and tools in research articles or in the ARCH competition [9] is evaluated rather sparsely using a few specific benchmarks only. Because of the small number of existing benchmarks, experts and tool developers currently tune their algorithms individually for each benchmark. However, this sometimes obscures the ability of certain approaches since users cannot tune the algorithms to the same extent. Motivated by these shortcomings, we propose a benchmark that automatically generates thousands of verification tasks, and is therefore well-suited for a more exhaustive analysis of verification approaches and tools.

The main idea of the proposed benchmark is to make use of the CommonRoad framework [3], which contains a large database of traffic scenarios for motion planning of autonomous vehicles. A visualization of an exemplary traffic scenario is shown in Fig. 1. For many of these scenarios, solutions in the form of collision-free trajectories are already available in CommonRoad. However, these solutions consider a nominal vehicle model only and are therefore potentially unsafe in a real-world scenario, where model uncertainties, disturbances, and measurement errors have to be considered. To counteract such disturbances, we control the car with a controller which

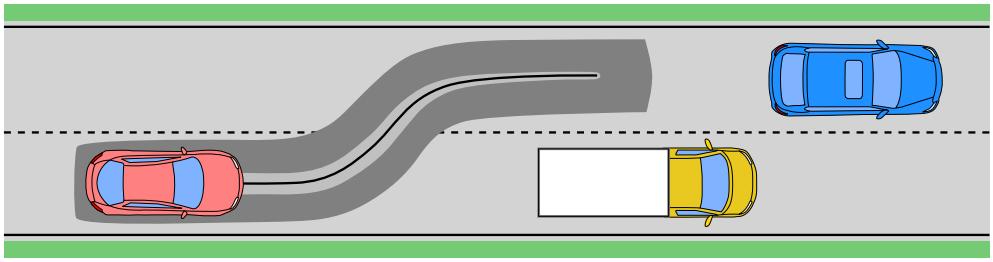

Figure 1: Exemplary traffic scenario, where the planned reference trajectory is shown as a black line, the potential positions of the vehicle's reference point are shown in light gray around the line, and the corresponding space occupied by the car is shown in dark gray.

aims to track the given reference trajectory and consequently keeps the vehicle's reference point in a small tube around the trajectory. The verification task is to prove that this tracking controller is robustly safe by showing that the space occupied by the car does not intersect with the road boundary or other vehicles present in the scenario.

As mentioned above, CommonRoad contains thousands of different traffic scenarios, from which we could generate thousands of verification tasks. Initially, we consider 100 different traffic scenarios, which are provided in the software package for this benchmark proposal. Moreover, to make the use of the benchmark as easy as possible, we provide software for the collision checks. Details about this are provided later in Sec. 5. In addition to performance evaluation of reachability tools, the proposed benchmark also has high practical relevance since the possibility to verify robust safety of planned trajectories in real-time would be a major leap forward in the field of autonomous driving.

2 Related Work

Besides CommonRoad, there also exist several other tools and frameworks for the performance evaluation and testing of autonomous driving approaches. Similar to CommonRoad, GeoScenario [20] defines a domain specific language for representing motion planning tasks for autonomous vehicles and provides a database of traffic scenarios. Moreover, the scenario architect in [23] presents a graphical user interface for designing multi-vehicle traffic scenarios. Some frameworks for the simulation and testing of autonomous vehicles are BARK [6], CoInCar-Sim [17], F1TENTH [18], and VeriCAV [13], where BARK and CoInCar-SIM focus on interactive behavior in multi-agent scenarios. For evaluating the performance of motion planners it is advantageous to especially consider critical scenarios for which the solution space is small. The automated generation of critical traffic scenarios is therefore a research topic of special interest [4, 11, 12].

Many different strategies exist for solving motion planning tasks for autonomous vehicles: discretization-based planners, which include rapidly exploring random trees [8] and state lattices [27], discretize the search space by considering a finite set of possible motions only. Other approaches formulate motion planning as a continuous optimization problem [16,26], or precom-

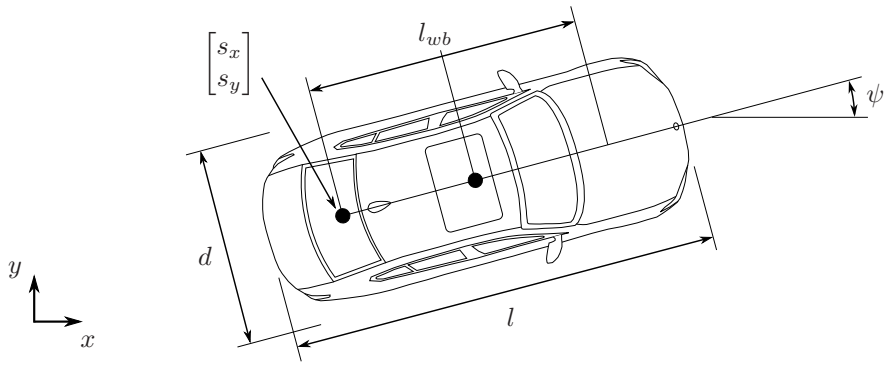

Figure 2: Visualization of the vehicle dimensions and the vehicle reference point $[s_x \ s_y]^T$ on the rear axis.

pute possible driving corridors to simplify the motion planning task [10,15]. Machine learning has also been successfully applied to motion planning of autonomous vehicles [21,25]. While the approaches listed above consider nominal vehicle models only, there also exist some techniques that explicitly consider disturbances and measurement errors to guarantee robust safety [22,24].

3 Benchmark Description

After introducing the general setup in Sec. 1, the benchmark is described here in detail. The car considered is a BMW 320i, which has a length of $l = 4.508$m, a width of $d = 1.61$m , and a wheelbase length of $l_{wb} = 2.578$m (see Fig. 2). Its dynamic behavior is represented by a kinematic single-track model [3, Sec. III.B]:

$$\begin{aligned} \dot{\delta}(t) &= u_1(t) + w_1(t) \\ \dot{\psi}(t) &= \frac{v(t)}{l_{wb}} \tan(\delta(t)) \\ \dot{v}(t) &= u_2(t) + w_2(t) \\ \dot{s}_x(t) &= v(t) \cos(\psi(t)) \\ \dot{s}_y(t) &= v(t) \sin(\psi(t)), \end{aligned} \tag{1}$$

where $t \in \mathbb{R}_{\geq 0}$ is the time and the system states are the steering angle $\delta(t)$, the vehicle heading $\psi(t)$, the vehicle velocity $v(t)$, and the x and y positions of the vehicle's reference point $s_x(t), s_y(t)$ on the rear axis (see Fig. 2). The control inputs for the car are the velocity of the steering angle $u_1(t)$ and the acceleration $u_2(t)$. We use the shorthands $\dot{x}(t) = f(x(t), u(t), w(t))$ to denote the open-loop system in (1), where $x(t) = [\delta(t) \ \psi(t) \ v(t) \ s_x(t) \ s_y(t)]^T$ is the state vector, $u(t) = [u_1(t) \ u_2(t)]^T$ is the vector of control inputs, and $w(t) = [w_1(t) \ w_2(t)]^T$ is the vector of uncertain inputs. The uncertain inputs, which capture model uncertainties and disturbances acting on the vehicle, are bounded by the set $w(t) \in \mathcal{W} = [-0.02, 0.02]\text{rad}\,\text{s}^{-1} \times [-0.3, 0.3]\text{m}\,\text{s}^{-2}$, which we estimated from measurements of the real car. In addition, owing to actuator limits, the control inputs are restricted to the set $u(t) \in \mathcal{U} = [-0.7, 0.7]\text{rad}\,\text{s}^{-1} \times [-11, 11]\text{m}\,\text{s}^{-2}$.

There exist many different tracking controllers for autonomous driving, a comparison of which is provided in [7]. Motivated by the combination of simplicity and good tracking performance, we use the controller

$$u_{fb}\big(\widehat{x}(t), t\big) = u_{ref}(t) + K(t)\big(\widehat{x}(t) - x_{ref}(t)\big), \tag{2}$$

where $K(t) \in \mathbb{R}^{2\times 5}$ is a time-varying feedback matrix and $u_{ref}(t) \in \mathbb{R}^2$ are the control inputs corresponding to the reference trajectory $x_{ref}(t) \in \mathbb{R}^5$. Both, $K(t)$ and $u_{ref}(t)$ are piecewise constant over time, where we denote a time interval with constant values by $\tau_i = [t_{i-1}, t_i]$, $i = 1, \ldots, N$. The variable $\widehat{x}(t) := x(t) + v(t)$ in (2) denotes the measured system state, which is the actual system state disturbed by the measurement error $v(t) \in \mathbb{R}^5$. The measurement error is bounded by the set $v(t) \in \mathcal{V} = [-0.0004, 0.0004]\text{rad} \times [-0.0004, 0.0004]\text{rad} \times [-0.006, 0.006]\text{m}\,\text{s}^{-1} \times [-0.002, 0.002]\text{m} \times [-0.002, 0.002]\text{m}$, which we estimated from measurements of the real car. Inserting the control law in (2) into (1) yields the dynamics of the closed-loop system:

$$\underbrace{\begin{bmatrix} \dot{x}(t) \\ \dot{x}_{ref}(t) \end{bmatrix}}_{\dot{\widetilde{x}}(t)} = \begin{bmatrix} f\big(x(t), u_{ref}(t) + K(t)\big(x(t) + v(t) - x_{ref}(t)\big), w(t)\big) \\ f\big(x_{ref}(t), u_{ref}(t), \mathbf{0}\big) \end{bmatrix}, \tag{3}$$

where we extend the original system state by the reference trajectory state. This is necessary since only the control inputs $u_{ref}(t)$ for the reference trajectory are available, but not a closed formula for the reference trajectory $x_{ref}(t)$ itself. An alternative would be to finely sample from the reference trajectory, and then use these samples to implement the controller. However, this would significantly restrict the time step size that can be used by verification tools, so we prefer the variant with the extended system state in (3).

The considered time horizon $t \in [t_0, t_N]$ for verification is defined by the time horizon for the reference trajectory $x_{ref}(t)$. Moreover, the initial conditions for the differential equation in (3) are given as $\widetilde{x}(t_0) \in \mathcal{X}_0 = (x_0 \oplus \mathcal{V}) \times x_0$, where $x_0 = x_{ref}(t_0) \in \mathbb{R}^5$ is the initial state of the reference trajectory. Furthermore, both $w(t)$ and $v(t)$ can change arbitrarily over time: $\forall t \in [t_0, t_N] : w(t) \in \mathcal{W} \wedge v(t) \in \mathcal{V}$. Finally, the verification goal is to check if the following specifications are satisfied:

- **Collision avoidance:** The car should not collide with the static obstacles including the road boundary or the dynamic obstacles (e.g. other traffic participants).
- **Input constraint:** The tracking control law in (2) should satisfy the input constraint $\forall t \in [t_0, t_N] : u_{fb}(\widehat{x}(t), t) \in \mathcal{U}$.

A possible future extension to the proposed benchmark would be to consider traffic rules, which would result in additional temporal logic specifications [14].

4 Verification using Reachability Analysis

We now describe how the benchmark can be verified using reachability analysis. An overview for the verification process using reachability analysis is shown in Fig. 3. We first compute a tight enclosure of the reachable set $\mathcal{R}(t)$ for the closed-loop system in (3) defined as

$$\mathcal{R}(t) := \Big\{ \xi\big(t, \widetilde{x}(t_0), w(\cdot), v(\cdot)\big) \;\Big|\; \widetilde{x}(t_0) \in \mathcal{X}_0,\ \forall t^* \in [t_0, t] :\ w(t^*) \in \mathcal{W} \wedge v(t^*) \in \mathcal{V} \Big\},$$

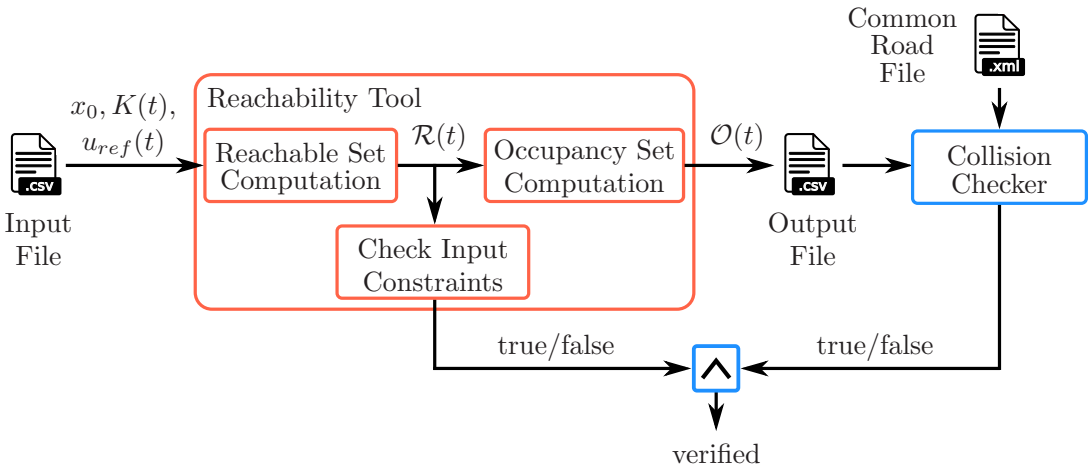

Figure 3: Overview of the verification process using reachability analysis.

where $\xi(t, \widetilde{x}(t_0), w(\cdot), v(\cdot))$ denotes the solution to (3) at time t starting from the initial state $\widetilde{x}(t_0) \in \mathbb{R}^{10}$ for an input signal $u(\cdot)$ and an measurement error signal $v(\cdot)$. The reachable set contains all possible positions of the vehicle's reference point. However, we additionally have to consider the dimensions of the car for collision checking. From the reachable set, we therefore compute a tight enclosure of the occupancy set $\mathcal{O}(t)$ defined as [2, Sec. V]

$$\mathcal{O}(t) := \left\{ \begin{bmatrix} s_x + \cos(\psi)\epsilon_1 - \sin(\psi)\epsilon_2 \\ s_y + \sin(\psi)\epsilon_1 + \cos(\psi)\epsilon_2 \end{bmatrix} \middle| \begin{array}{l} [\delta \ \psi \ v \ s_x \ s_y]^T \in \mathcal{R}(t), \\ \epsilon_1 \in \left[-\frac{l}{2} + \frac{l_{wb}}{2}, \frac{l}{2} + \frac{l_{wb}}{2} \right], \ \epsilon_2 \in \left[-\frac{d}{2}, \frac{d}{2} \right] \end{array} \right\}, \tag{4}$$

which is the space occupied by the vehicle. Our provided collision checker can then be used to check if the occupancy set intersects with static or dynamic obstacles. This requires the exportation of the computed occupancy set as a .csv-file with a specific format. Details about this file format and on how to use the collision checker are provided later in Sec. 5. What remains is to check if the input constraints are satisfied. For this, we first compute a tight enclosure of the set of applied control inputs $\mathcal{U}_{fb}(t)$ defined as

$$\mathcal{U}_{fb}(t) := \left\{ u_{ref}(t) + K(t)\big(x(t) + v(t) - x_{ref}(t)\big) \ \middle| \ \begin{bmatrix} x(t) \\ x_{ref}(t) \end{bmatrix} \in \mathcal{R}(t), \ v(t) \in \mathcal{V} \right\}. \tag{5}$$

To check the input constraints, we have to test if the set of applied control inputs in (5) is contained in the set of admissible control inputs for all times: $\forall t \in [t_0, t_N] : \ \mathcal{U}_{fb}(t) \subseteq \mathcal{U}$.

```
code
└── check_collision.py
data
├── scenarios
│   ├── <scenario_name>.xml
│   ├── <scenario_name>.xml
│   └── ...
└── inputs
    ├── <scenario_name>_controls.csv
    ├── <scenario_name>_controls.csv
    └── ...
results
└── solutions
    ├── <scenario_name>_occupancies.csv
    ├── <scenario_name>_occupancies.csv
    └── ...
```

```
code
└── check_collision.py
data
├── scenarios
│   ├── ESP_Inca-7_1_T-1.xml
│   ├── FRA_Sete-1_1_T-1.xml
│   └── ...
└── inputs
    ├── ESP_Inca-7_1_T-1_controls.csv
    ├── FRA_Sete-1_1_T-1_controls.csv
    └── ...
results
└── solutions
    ├── ESP_Inca-7_1_T-1_occupancies.csv
    ├── FRA_Sete-1_1_T-1_occupancies.csv
    └── ...
```

Figure 4: Directory structure of the provided software package, where the general format is shown on the left and a concrete example is shown on the right.

5 Implementation Details

For ease of use, we provide the data for 100 different traffic scenarios as well as a collision checker in a software package, which we published as a CodeOcean compute capsule[1]. The reference trajectories for all provided traffic scenarios are robustly safe, so that all problems can be verified as safe. In this section, we give a detailed description of the involved data formats and explain how the collision checker can be used. The directory structure for the provided software package is shown in Fig. 4.

Input File

For each traffic scenario, the initial state x_0, the control inputs for the planned reference trajectory $u_{ref}(t)$, and the feedback matrix for the control law $K(t)$ are provided in a .csv-file with the name <scenario_name>_controls.csv (see Fig. 4). The format of the file is shown in Tab. 1, where the first line of the file stores the initial state $x_0 = x(t_0) = [\delta(t_0)\ \psi(t_0)\ v(t_0)\ s_x(t_0)\ s_y(t_0)]^T$. The remaining lines contain the time-varying reference inputs $u_{ref}(t)$ and the time-varying feedback matrix $K(t)$, where each line corresponds to a time interval $\tau_i = [t_{i-1}, t_i]$, $i = 1, \dots N$ in which the values for $u_{ref}(t)$ and $K(t)$ are constant:

$$u_{ref}(\tau_i) = \begin{bmatrix} u_{ref,1}^{(i)} \\ u_{ref,2}^{(i)} \end{bmatrix}, \quad K(\tau_i) = \begin{bmatrix} K_{1,1}^{(i)} & K_{1,2}^{(i)} & K_{1,3}^{(i)} & K_{1,4}^{(i)} & K_{1,5}^{(i)} \\ K_{2,1}^{(i)} & K_{2,2}^{(i)} & K_{2,3}^{(i)} & K_{2,4}^{(i)} & K_{2,5}^{(i)} \end{bmatrix}, \quad i = 1, \dots, N.$$

[1] https://codeocean.com/capsule/0922183/tree/v1

Table 1: Format of the input .csv-file storing the initial state, the control inputs for the planned reference trajectory, and the feedback matrix for one scenario.

t_0	$\delta(t_0)$	$\psi(t_0)$	$v(t_0)$	$s_x(t_0)$	$s_y(t_0)$							
t_1	$u_{ref,1}^{(1)}$	$u_{ref,2}^{(1)}$	$K_{1,1}^{(1)}$	$K_{1,2}^{(1)}$	$K_{1,3}^{(1)}$	$K_{1,4}^{(1)}$	$K_{1,5}^{(1)}$	$K_{2,1}^{(1)}$	$K_{2,2}^{(1)}$	$K_{2,3}^{(1)}$	$K_{2,4}^{(1)}$	$K_{2,5}^{(1)}$
t_2	$u_{ref,1}^{(2)}$	$u_{ref,2}^{(2)}$	$K_{1,1}^{(2)}$	$K_{1,2}^{(2)}$	$K_{1,3}^{(2)}$	$K_{1,4}^{(2)}$	$K_{1,5}^{(2)}$	$K_{2,2}^{(2)}$	$K_{2,2}^{(2)}$	$K_{2,3}^{(2)}$	$K_{2,4}^{(2)}$	$K_{2,5}^{(2)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_N	$u_{ref,1}^{(N)}$	$u_{ref,2}^{(N)}$	$K_{1,1}^{(N)}$	$K_{1,2}^{(N)}$	$K_{1,3}^{(N)}$	$K_{1,4}^{(N)}$	$K_{1,5}^{(N)}$	$K_{2,1}^{(N)}$	$K_{2,2}^{(N)}$	$K_{2,3}^{(N)}$	$K_{2,4}^{(N)}$	$K_{2,5}^{(N)}$

CommonRoad File

The CommonRoad framework stores each traffic scenario in an .xml file with the name <scenario_name>.xml (see Fig. 4), using the specific CommonRoad XML format[2] as defined in [3, Sec. V]. This file stores a formal representation of the road network, static and dynamic obstacles, and the corresponding planning problem defined by the initial state and goal set. For the proposed benchmark, we only require the static and dynamic obstacles. If reachability analysis is used for verification, the data required from the CommonRoad file is automatically extracted by the provided collision checker, so that the user does not need to be concerned with the CommonRoad file. For other verification approaches, we recommend to use the CommonRoad input-output package[3] to efficiently read and manipulate the data stored in CommonRoad files.

Output File

If reachability analysis is used for verification, the computed occupancy set $\mathcal{O}(t) \subset \mathbb{R}^2$ for each scenario has to be stored in a corresponding output .csv-file with the name <scenario_name>_occupancies.csv (see Fig. 4), to use the provided collision checker. Since all common reachability tools compute the reachable set for consecutive time intervals $\widetilde{\tau}_i = [t_{s,i}, t_{e,i}]$, $i = 1, \ldots, M$, one occupancy set $\mathcal{O}(\widetilde{\tau}_i)$ is stored for each time interval $\widetilde{\tau}_i$. Note that the time intervals for reachability analysis $\widetilde{\tau}_i$ are in general not identical to the time intervals τ_i in which the values for $u_{ref}(t)$ and $K(t)$ are constant. We require that the occupancy set $\mathcal{O}(\widetilde{\tau}_i)$ for each time interval $\widetilde{\tau}_i$ is represented as a polygon defined by a tuple of polygon vertices $p_j^{(i)} \in \mathbb{R}^2$:

$$\mathcal{O}(\widetilde{\tau}_i) = (p_1^{(i)}, \ldots, p_{Q_i}^{(i)}), \quad \text{with } p_j^{(i)} = \begin{bmatrix} p_{j,1}^{(i)} \\ p_{j,2}^{(i)} \end{bmatrix}, \quad j = 1, \ldots, Q_i, \; i = 1, \ldots, M.$$

The format for the output .csv-file is shown in Tab. 2. Two consecutive lines always store the occupancy set $\mathcal{O}(\widetilde{\tau}_i)$ for one time interval $\widetilde{\tau}_i$, where the first column stores the start and end time of the time interval and the remaining columns store the vertices of the polygon.

[2] https://gitlab.lrz.de/tum-cps/commonroad-scenarios/-/blob/master/documentation/XML_commonRoad_2020a.pdf

[3] https://commonroad.in.tum.de/commonroad-io

Table 2: Format for the output .csv-file storing the occupancy set for one scenario demonstrated by an exemplary occupancy set.

$t_{s,1}$	$p_{1,1}^{(1)}$	$p_{2,1}^{(1)}$	$p_{3,1}^{(1)}$	$p_{4,1}^{(1)}$		
$t_{e,1}$	$p_{1,2}^{(1)}$	$p_{2,2}^{(1)}$	$p_{3,2}^{(1)}$	$p_{4,2}^{(1)}$		
$t_{s,2}$	$p_{1,1}^{(2)}$	$p_{2,1}^{(2)}$	$p_{3,1}^{(2)}$			
$t_{e,2}$	$p_{1,2}^{(2)}$	$p_{2,2}^{(2)}$	$p_{3,2}^{(2)}$			
$t_{s,3}$	$p_{1,1}^{(3)}$	$p_{2,1}^{(3)}$	$p_{3,1}^{(3)}$	$p_{4,1}^{(3)}$	$\dots$	$p_{Q_3,1}^{(3)}$
$t_{e,3}$	$p_{1,2}^{(3)}$	$p_{2,2}^{(3)}$	$p_{3,2}^{(3)}$	$p_{4,2}^{(3)}$	$\dots$	$p_{Q_3,2}^{(3)}$
$\vdots$	$\vdots$					
$t_{s,M}$	$p_{1,1}^{(M)}$	$p_{2,1}^{(M)}$	$p_{3,1}^{(M)}$	$p_{4,1}^{(M)}$	$\dots$	$p_{Q_M,1}^{(M)}$
$t_{e,M}$	$p_{1,2}^{(M)}$	$p_{2,2}^{(M)}$	$p_{3,2}^{(M)}$	$p_{4,2}^{(M)}$	$\dots$	$p_{Q_M,2}^{(M)}$

Collision Checker

To make the use of the proposed benchmark as easy as possible, we provide a Python script `check_collision.py`, which checks if the occupancy sets stored in the output files collide with the road boundary or the other vehicles of the considered traffic scenarios. This script is based on the CommonRoad drivability checker in [19]. Before running the script, one therefore first has to install the CommonRoad drivability checker using the detailed installation instructions provided on the CommonRoad website[4]. Another requirement for the collision checker to run properly is that the directory structure shown in Fig. 4 is used. Finally, the collision checker can be executed from the console as follows:

```
python3 check_collision.py
python3 check_collision.py <scenario_name>
```

If the collision checker is executed without inputs, it considers all scenarios for which an output file exists that stores the occupancy sets, and displays a list to the console with scenarios where collisions occurred. Otherwise, if the name of one scenario is passed as an additional input argument, the collision checker only considers the specified scenario. In this case, if a collision occurs for the specified scenario, an animation visualizing the collision is shown.

6 Numerical Experiments

We now provide some numerical results for the proposed benchmark obtained by the reachability toolbox CORA [1], which is implemented in MATLAB. All computations are carried out on a 2.9GHz quad-core i7 processor with 32GB of memory. In particular, we consider the conservative linearization algorithm [5], which computes a tight enclosure of the reachable set by linearizing the nonlinear system dynamics on the fly. Moreover, to compute a tight enclosure of the occupancy set, we first compute a Taylor series expansion of the nonlinear function in

[4] https://commonroad.in.tum.de/docs/commonroad-drivability-checker/sphinx/index.html

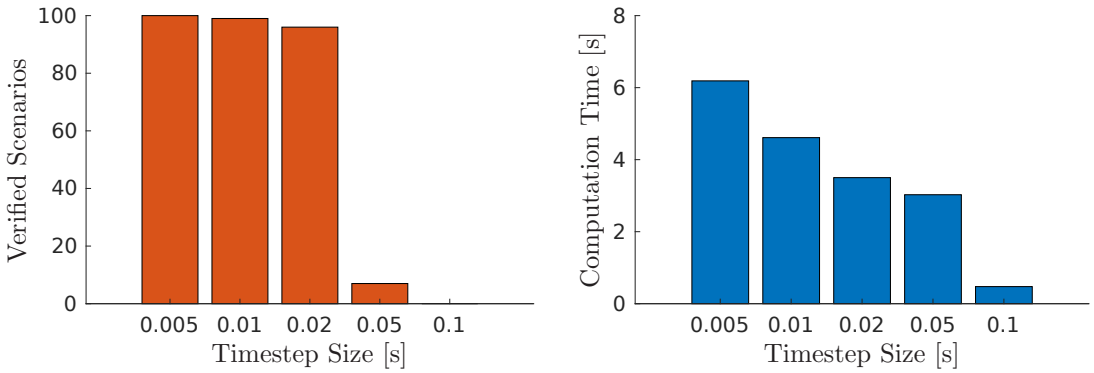

Figure 5: Number of successfully verified traffic scenarios (left) and average computation time for the verification of a planned trajectory with a time horizon of 1 second (right) for different reachability timestep sizes.

(4), which we then evaluate in a set-based manner. The results for different sizes of the reachability analysis time intervals $\widetilde{\tau}_i$ are shown in Fig. 5. If the chosen timestep size is small enough, CORA is able to successfully verify that the planned trajectories for all 100 of the considered traffic scenarios are robustly safe. Verification accelerates with larger timestep size, but then it is not possible to verify safety for all traffic scenarios. Finally, the computed occupancy set for one concrete traffic scenario is visualized in Fig. 6.

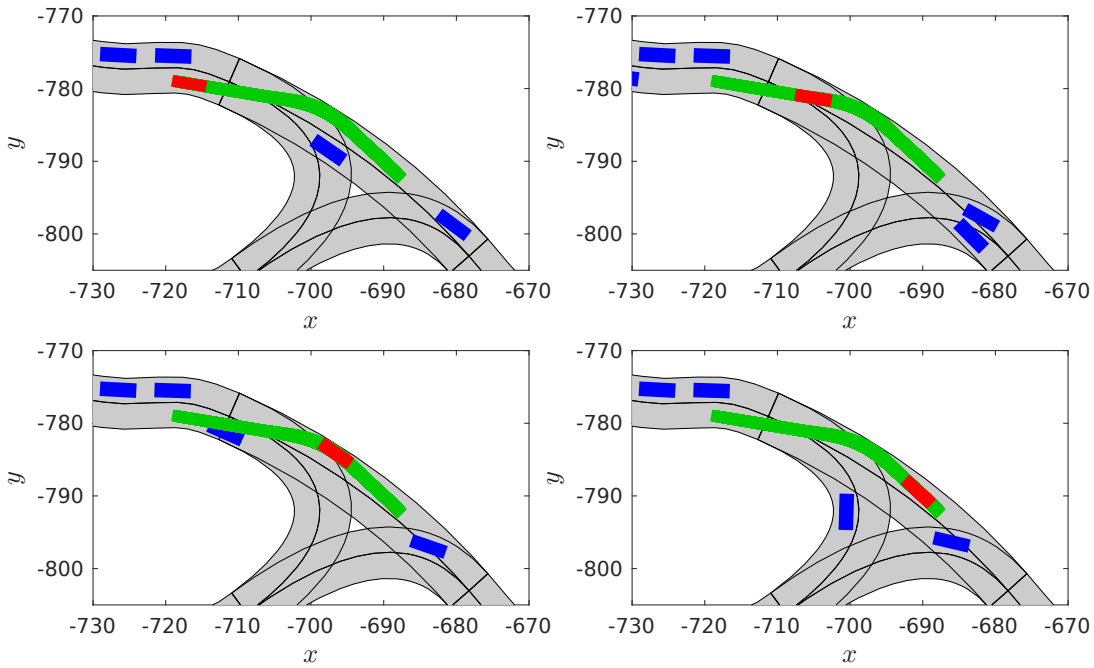

Figure 6: Visualization of the computed occupancy set for the CommonRoad scenario *BEL_Putte-4_2_T-1* at times 0s (top, left), 1s (top, right), 2s (bottom, left), and 3s (bottom, right). The occupancy set for the whole time horizon is depicted in green, the occupancy set for the current time is depicted in red, and the other vehicles are depicted in blue.

7 Conclusions

We introduced a novel benchmark with nonlinear continuous dynamics to verify planned trajectories for autonomous cars. Using the traffic scenarios from the CommonRoad database, it is possible to generate many thousands of different verification tasks, resulting in a benchmark that is well-suited for an exhaustive analysis of verification algorithms and tools. The fact that the dynamic of the system changes constantly through the use of time-varying feedback matrices provides an additional challenge, so that the proposed benchmark is a good fit for performance comparisons. Finally, numerical experiments with the reachability toolbox CORA demonstrate that it is possible to successfully verify all scenarios.

Acknowledgments

The authors gratefully acknowledge the partial financial support of this work by the European Research Council (ERC) project justITSELF under grant agreement No 817629.

References

[1] M. Althoff. An introduction to CORA 2015. In *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[2] M. Althoff and J. M. Dolan. Set-based computation of vehicle behaviors for the online verification of autonomous vehicles. In *Proc. of the 14th IEEE Conference on Intelligent Transportation Systems*, pages 1162–1167, 2011.

[3] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. of the Intelligent Vehicles Symposium*, pages 719–726, 2017.

[4] M. Althoff and S. Lutz. Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In *Proc. of the Intelligent Vehicles Symposium*, pages 1326–1333, 2018.

[5] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Proc. of the International Conference on Decision and Control*, pages 4042–4048, 2008.

[6] J. Bernhard, K. Esterle, P. Hart, and T. Kessler. BARK: Open behavior benchmarking in multi-agent environments. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 6201–6208, 2020.

[7] D. Calzolari, B. Schürmann, and M. Althoff. Comparison of trajectory tracking controllers for autonomous vehicles. In *Proc. of the International Conference on Intelligent Transportation Systems*, 2017.

[8] E. Frazzoli, M. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[9] L. Geretti and et al. ARCH-COMP20 category report: Continuous and hybrid systems with nonlinear dynamics. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*, pages 49–75, 2020.

[10] T. Gu, J. M. Dolan, and J. W. Lee. Automated tactical maneuver discovery, reasoning and trajectory planning for autonomous driving. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 5474–5480, 2016.

[11] M. Klischat and M. Althoff. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *Proc. of the Intelligent Vehicles Symposium*, pages 2352–2358, 2019.

[12] C. Knies and F. Diermeyer. Data-driven test scenario generation for cooperative maneuver planning on highways. *Applied Sciences*, 10(22):Article 8154, 2020.

[13] T. Levermore and A. Peters. Test framework and key challenges for virtual verification of automated vehicles: the VeriCAV project. In *Proc. of the International Conference on Computer Safety, Reliability and Security*, 2020.

[14] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff. Formalization of interstate traffic rules in temporal logic. In *Proc. of the Intelligent Vehicles Symposium*, pages 752–759, 2020.

[15] S. Manzinger, C. Pek, and M. Althoff. Using reachable sets for trajectory planning of automated vehicles. *Transactions on Intelligent Vehicles*, 6(2):232–248, 2021.

[16] F. Molinario, N. N. Anh, and L. Del Re. Efficient mixed integer programming for autonomous overtaking. In *Proc. of the American Control Conference*, pages 2303–2308, 2017.

[17] M. Naumann, F. Poggenhans, M. Lauer, and C. Stiller. CoInCar-Sim: An open-source simulation framework for cooperatively interacting automobiles. In *Proc. of the Intelligent Vehicles Symposium*, pages 1879–1884, 2018.

[18] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam. F1TENTH: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, 123:77–89, 2020.

[19] C. Pek, V. Rusinov, S. Manzinger, and M. Althoff. CommonRoad drivability checker: Simplifying the development and validation of motion planning algorithms. In *Proc. of the Intelligent Vehicles Symposium*, pages 1013–1020, 2020.

[20] R. Queiroz, T. Berger, and K. Czarnecki. GeoScenario: An open DSL for autonomous driving scenario representation. In *Proc. of the Intelligent Vehicles Symposium*, pages 287–294, 2019.

[21] M. Riedmiller, M. Montemerlo, and H. Dahlkamp. Learning to drive a real car in 20 minutes. In *Proc. of the International Conference on Frontiers in the Convergence of Bioscience and Information Technologies*, pages 645–650, 2007.

[22] B. Schürmann and et al. Ensuring drivability of planned motions using formal methods. In *Proc. of the International Conference on Intelligent Transportation Systems*, 2017.

[23] T. Stahl and J. Betz. An open-source scenario architect for autonomous vehicles. In *Proc. of the International Conference on Ecological Vehicles and Renewable Energies*, 2020.

[24] S. Vaskov and et al. Guaranteed safe reachability-based trajectory design for a high-fidelity model of an autonomous passenger vehicle. In *Proc. of the American Control Conference*, pages 705–710, 2019.

[25] P. Wolf and et al. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. In *Proc. of the Intelligent Vehicles Symposium*, pages 993–1000, 2018.

[26] J. Ziegler, P. Bender, T. Dang, and C. Stiller. Trajectory planning for BERTHA – a local, continuous method. In *Proc. of the Intelligent Vehicles Symposium*, pages 450–457, 2014.

[27] J. Ziegler and C. Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 1879–1884, 2009.