CommonRoad-Reach: A Toolbox for Reachability Analysis of Automated Vehicles

Edmond Irani Liu*, Gerald Würsching*, Moritz Klischat, and Matthias Althoff

***Abstract*— In recent years, reachability analysis has gained considerable popularity in motion planning and safeguarding of automated vehicles (AVs). While existing tools for reachability analysis mainly focus on general-purpose algorithms for formal verification of dynamical systems, a toolbox tailored to AV-specific applications is not yet available. In this study, we present CommonRoad-Reach, which is a toolbox that integrates different methods for computing reachable sets and extracting driving corridors for AVs in dynamic traffic scenarios. Our toolbox provides a Python interface and an efficient C++ implementation for real-time applications. The toolbox is integrated within the CommonRoad benchmark suite and is available at commonroad.in.tum.de.**

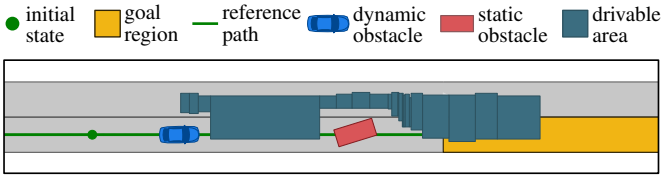


Fig. 1: Exemplary computation result of our toolbox for a simple scenario with a static and dynamic obstacle. Our tool computes the reachable set of the ego vehicle over time considering the initial state and (time-varying) forbidden states. The projection of the reachable set onto the position domain yields a collision-free drivable area shown here for step $k = 28$.

I. Introduction

Compared with human-driven vehicles, highly automated vehicles (AVs) are expected to offer increased road safety and passenger comfort, reduced emissions and travel time. Major challenges, such as strict safety guarantees in critical situations, decision making, and motion planning in small solution space, are yet to be resolved to fully unfold these benefits. Reachability analysis, which determines the set of all reachable states (also referred to as *reachable set*) of a system over time, is a powerful technique to address these challenges. An example of a reachable set is shown in Fig. 1. Despite reachability analysis having been well-researched over the past decades with continuous improvements in scalability and tightness [1], publicly available toolboxes are either not real-time capable for AV-specific applications or do not take into account time-varying forbidden states originating from traffic participants present in the scenario. This study presents a toolbox for computing the reachable sets of AVs for motion planning applications. The toolbox integrates two methods presented in our previous works: polytopic set propagation [2] and graph-based propagation [3].

A. Related Work

1) Reachability analysis for road vehicles: General-purpose approaches for reachability analysis are primarily used for formal verification, i.e., checking whether a system can reach unsafe sets considering system dynamics and constraints, e.g., on input or disturbance bounds. These methods are useful for AVs to verify their motion plans, see, e.g., [4], [5]. However, motion planning or rigorous computation of safety metrics, such as time-to-react [6], requires especially efficient algorithms considering surrounding moving vehicles.

Several intriguing applications of reachability analysis for AVs have recently been proposed in the literature: Reachability analysis can be used to determine the set of states that ultimately result in a collision irrespective of the input of choice [7]. Set-based prediction using reachability analysis has been proposed to capture all possible future behaviors of surrounding traffic participants, which also considers limitations due to the field-of-view and traffic rules [8]–[10]. Online verification techniques that utilize safety layers based on reachability analysis can ensure the safety of motion planners [11]–[15]. A further line of research uses reachable sets to extract possible driving corridors, i.e., spatio-temporal constraints, for motion planning [16], [17]. For this application, set-based techniques are especially well suited in complex scenarios as they do not suffer from discretization effects and are computationally feasible under real-time constraints, in contrast to other methods, such as sampling-based [18] or combinatorial [19] approaches. Driving corridors extracted from reachable sets are integrated with different motion planners in [16], [20]. Furthermore, reachable sets are used to determine specification-compliant planning space [21] and to negotiate conflicting planning space among a group of cooperating vehicles [22].

2) Existing toolboxes for reachability analysis: Recently, several publicly available toolboxes for reachability analysis have been developed. Tools such as Flow* [23] and SpaceEx [24] offer efficient C++ implementations of set representations and reachability algorithms for linear [24] and nonlinear [23] hybrid systems. Although C++-based tools have good performance, their compilation overhead makes them difficult to use for prototyping. Hence, reachability tools written in just-in-time compiled or interpreted languages are desirable; examples include the MATLAB-based tool CORA [25], JuliaReach [26], or the Python-based tool HyLAA [27].

* The first two authors have contributed equally to this work.
All authors are with the Department of Informatics, Technical University of Munich, 85748 Garching, Germany.
{edmond.irani,gerald.wuersching,moritz.klischat, althoff}@tum.de

B. Contributions

Existing toolboxes are either not capable of excluding time-varying forbidden states [23]–[26], [28], or are based on inherently inefficient Hamilton–Jacobi–Bellmann solvers [29]–[32]. Despite the aforementioned applications of reachable sets in recent works, a toolbox implementing those methods is not publicly available. Our open-source toolbox is tailored to AV-specific applications and

- integrates two approaches for computing reachable sets, i.e., using polytopic set propagation and graph-based propagation;
- extracts collision-free driving corridors that can be used as planning constraints for motion planners;
- provides Python and C++ implementations of the algorithms, offering convenient prototyping and real-time computation to the users; and
- is integrated within the CommonRoad[1] benchmark suite [33], which offers a simulation framework, an extensive scenario database, and various tools for motion planning of automated vehicles.

The remainder of this study is structured as follows: Sec. II introduces necessary preliminaries and Sec. III highlights the implementation details of our toolbox. In Sec. IV, we demonstrate the key features of our toolbox in numerical experiments. Lastly, we draw conclusions in Sec. V.

II. Preliminaries

A. System Description

The scenarios in our toolbox are described in the CommonRoad format, which represents the road network using lanelets [34] and models environment elements such as obstacles and traffic signs. We predict the motion of dynamic obstacles using their most-likely trajectories; however, our toolbox is also adaptable to any other prediction method such as set-based prediction [8]. Both the global Cartesian coordinate system and a local curvilinear coordinate system that is aligned with a reference path [34] can be used to compute the reachable set. The possible reference path is the centerline of a lane or a path through the road network leading from the initial state to a specified goal region.

Let us introduce some necessary notations: We denote by $k \in \mathbb{N}_0$ a discrete step corresponding to a continuous time $t_k = k\Delta_t$, with a predefined time increment $\Delta_t \in \mathbb{R}_+$. For a given dynamical system, $\boldsymbol{x}_k \in \mathcal{X}_k \subset \mathbb{R}^4$ represents a state in the state space $\mathcal{X}_k$, and $\boldsymbol{u}_k \in \mathcal{U}_k \subset \mathbb{R}^2$ represents an input in the input space $\mathcal{U}_k$, each at step k. Let $\square$ be a variable, we denote its minimum and maximum values by $\underline{\square}$ and $\overline{\square}$, respectively. Since our toolbox facilitates the computation in Cartesian and curvilinear coordinate systems, we introduce the general subscripts $\zeta \in \{x, s\}$ and $\eta \in \{y, d\}$ to indicate the direction of a variable in the corresponding coordinate frame. Thus, (x, y) denotes the global coordinates in the Cartesian frame, and (s, d) denotes the longitudinal coordinate s and lateral coordinate d in a local curvilinear frame. Computations within the Cartesian coordinate system do not rely on a reference path and handle unstructured scenarios better. In contrast, computations within a curvilinear coordinate system are better suited for structured scenarios with lanes.

We abstract the system dynamics of a hypothetical ego vehicle using a point-mass model. This abstraction ensures that the reachable set of the high-fidelity model is always a subset of that of the simplified model; alternative abstractions can be found in [35], [36]. The state of the system is modeled as $\boldsymbol{x}_k = (p_{\zeta,k}, v_{\zeta,k}, p_{\eta,k}, v_{\eta,k})^\mathsf{T}$, and the system accepts inputs $\boldsymbol{u}_k = (a_{\zeta,k}, a_{\eta,k})^\mathsf{T}$, where p, v, a denote position, velocity, and acceleration, respectively. The discrete-time system dynamics of the ego vehicle is

$$\boldsymbol{x}_{k+1} = \begin{pmatrix} 1 & \Delta_t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{x_k} + \begin{pmatrix} \frac{1}{2}\Delta_t^2 & 0 \\ \Delta_t & 0 \\ 0 & \frac{1}{2}\Delta_t^2 \\ 0 & \Delta_t \end{pmatrix} \boldsymbol{u}_k. \quad (1)$$

The velocities and accelerations in both directions of the coordinate system are bounded by

$$\underline{v}_\zeta \le v_{\zeta,k} \le \overline{v}_\zeta,\ \underline{v}_\eta \le v_{\eta,k} \le \overline{v}_\eta, \quad (2a)$$

$$\underline{a}_\zeta \le a_{\zeta,k} \le \overline{a}_\zeta,\ \underline{a}_\eta \le a_{\eta,k} \le \overline{a}_\eta. \quad (2b)$$

These bounds are chosen to consider the kinematic limitations within the adopted coordinate system, see, e.g., [37]. Note that within the Cartesian coordinate system, we over-approximate Kamm's friction circle with a box; within curvilinear coordinate systems, we assume that the ego vehicle follows the reference path and thus use more conservative parameters for the longitudinal and lateral driving directions.

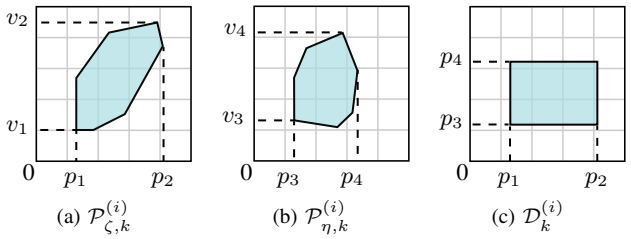
B. Reachable Set

Let $\boldsymbol{x}_0$ be the initial state and $\boldsymbol{u}_{[0,k]}$ be an input trajectory between steps 0 and k. We use $\chi_k(\boldsymbol{x}_0, \boldsymbol{u}_{[0,k]})$ to represent the solution of (1) at step k. Given the occupancy of the ego vehicle $\mathcal{Q}(\boldsymbol{x}_k) \subset \mathbb{R}^2$ and the set of time-varying occupancies of all obstacles $\mathcal{O}_k \subset \mathbb{R}^2$ at step k, we define the set of forbidden states as $\mathcal{F}_k = \{\boldsymbol{x}_k \in \mathcal{X}_k | \mathcal{Q}(\boldsymbol{x}_k) \cap \mathcal{O}_k \neq \emptyset\}$. In this study, the reachable set of the ego vehicle at step k is defined as the set of states reachable from the initial set of states $\mathcal{X}_0$ while avoiding the set of forbidden states $\mathcal{F}_\tau$ for every step $\tau \in \{0, \dots, k\}$ [2]:

$$\mathcal{R}_k^*(\mathcal{X}_0) := \Big\{ \chi_k(\boldsymbol{x}_0, \boldsymbol{u}_{[0,k]}) \Big| \exists \boldsymbol{x}_0 \in \mathcal{X}_0, \forall \tau \in \{0, \dots, k\}, \exists \boldsymbol{u}_\tau \in \mathcal{U}_\tau : \chi_\tau(\boldsymbol{x}_0, \boldsymbol{u}_{[0,\tau]}) \notin \mathcal{F}_\tau \Big\}.$$

Subsequently, we omit $\mathcal{X}_0$ for brevity. Obtaining $\mathcal{R}_k^*$ is in general computationally expensive; therefore, we compute its over-approximation $\mathcal{R}_k$. As a set representation, we choose the union of base sets $\mathcal{R}_k^{(i)}$, $i \in \mathbb{N}$. Each base set $\mathcal{R}_k^{(i)} = \mathcal{P}_{\zeta,k}^{(i)} \times \mathcal{P}_{\eta,k}^{(i)}$ is a Cartesian product of two convex polytopes $\mathcal{P}_{\zeta,k}^{(i)}$ and $\mathcal{P}_{\eta,k}^{(i)}$, which represent the reachable positions and velocities in the (p_ζ, v_ζ) and (p_η, v_η) planes, respectively (Fig. 2a–b). This representation is beneficial compared to other set representations in [1, Tab. 1] since

[1] https://commonroad.in.tum.de/



(a) $\mathcal{P}_{\zeta,k}^{(i)}$ (b) $\mathcal{P}_{\eta,k}^{(i)}$ (c) $\mathcal{D}_k^{(i)}$

Fig. 2: Polytopes and drivable area of a base set $\mathcal{R}_k^{(i)}$.

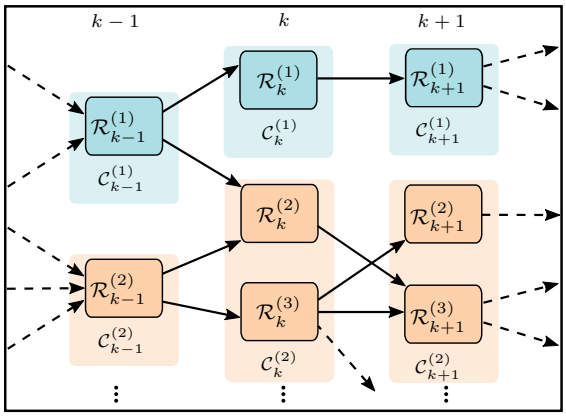


Fig. 3: Example of a reachability graph $\mathcal{G}_\mathcal{R}$. Nodes of the same color have connected drivable areas within one step and belong to one driving corridor (see Sec. II-C and Fig. 4).

polytopes are closed under linear maps and intersections; because of the restriction to two-dimensional polytopes, the unfavorable computational complexity of polytopes for high dimensions is irrelevant to our application. To simplify the notation, we denote the collection of $\mathcal{R}_k^{(i)}$ by $\mathcal{R}_k$, i.e., $\mathcal{R}_k = \{\mathcal{R}_k^{(1)}, \dots, \mathcal{R}_k^{(i)}\}$. The projection of $\mathcal{R}_k^{(i)}$ onto the position domain yields a axis-aligned rectangle $\mathcal{D}_k^{(i)}$ (Fig. 2c), whose union is referred to as *drivable area* $\mathcal{D}_k$. Similarly, we use $\mathcal{D}_k$ to denote the collection of $\mathcal{D}_k^{(i)}$.

We also require the reachability graph $\mathcal{G}_\mathcal{R}$, which stores the spatio-temporal relationships of the base sets $\mathcal{R}_k^{(i)}$ as a directed, acyclic graph (Fig. 3). In $\mathcal{G}_\mathcal{R}$, each node corresponds to one base set $\mathcal{R}_k^{(i)}$ and a connecting edge between two base sets $\mathcal{R}_k^{(i)}$ and $\mathcal{R}_{k+1}^{(j)}$ indicates that $\mathcal{R}_{k+1}^{(j)}$ is reachable from $\mathcal{R}_k^{(i)}$ after one step.

C. Driving Corridor

We follow the definition of *driving corridors* presented in [16]. At step k, the drivable area $\mathcal{D}_k$ may be disconnected due to the presence of obstacles, thus we introduce the notion of *connected sets* $\mathcal{C}_k^{(n)} \subseteq \mathcal{D}_k, n \in \mathbb{N}_0$ within the drivable area. A sequence of connected sets over steps 0 to k_f, where k_f is the final step, yields a driving corridor $\mathcal{C}(\cdot) = (\mathcal{C}_0^{(m)}, \dots, \mathcal{C}_{k_f}^{(n)})$. At every step, the drivable area can contain several connected sets, thus multiple driving corridors may exist. Fig. 3 shows two different driving corridors identified within a reachability graph. Therein, nodes of the same color collectively represent one driving corridor. The two driving corridors are visualized in Fig. 4. Each corridor corresponds

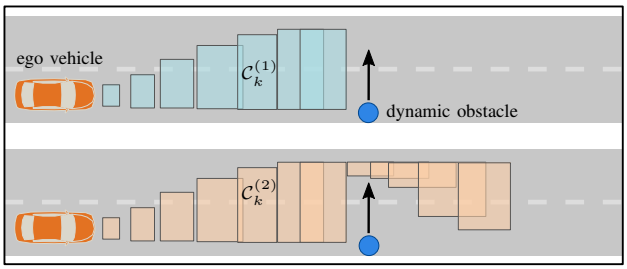


Fig. 4: Example of two driving corridors extracted from the reachability graph in Fig. 3. The driving corridors correspond to a braking maneuver (top) and an evasive maneuver (bottom). Each rectangle represents a connected set $\mathcal{C}_k^{(n)}$ at different steps $k \in \{0, \dots, k_f\}$.

to a possible maneuver of the ego vehicle with respect to the crossing obstacle.

Driving corridors can be separated into longitudinal and lateral driving corridors. This is particularly useful for motion planners that independently plan the longitudinal and lateral motions. A possible implementation is provided in [16]: Given a longitudinal driving corridor and the positions $p_{\zeta,k}$ of a planned longitudinal trajectory, a lateral driving corridor is obtained by identifying the connected sets within the longitudinal driving corridor, which contain the positions $p_{\zeta,k}$ for all $k \in \{0, \dots, k_f\}$. The search space for the lateral planning problem is thus further constrained by the fact that a lateral driving corridor in [16] is a subset of a longitudinal driving corridor.

III. Implementation Details

A. Overview

CommonRoad-Reach provides both the computation of reachable sets and the extraction of driving corridors. Subsequently, we present an overview of the toolbox and highlight its core modules (Fig. 5). The toolbox consists of the following core modules:

- `ReachableSetInterface` serves as the user interface for setting configurations, computing reachable sets and drivable areas, and extracting driving corridors.
- `ReachableSet` (Python) is an abstract superclass that instantiates the subclasses for computing reachable sets using either polytopic set propagation or graph-based propagation method.
- `PyReachableSet` is a Python implementation of the reachable set computation using the polytopic set propagation method described in [2].
- `CppReachableSet` interacts with `ReachableSet` (C++), which is a C++ implementation of the polytopic set propagation method.
- `PyGraphReachableSetOffline` precomputes reachability graphs for the graph-based reachability analysis according to [3].
- `PyGraphReachableSetOnline` loads the precomputed reachability graphs and performs the online computations of the graph-based reachability analysis.
- `DrivingCorridorExtractor` implements functions to extract possible driving corridors from a reach-

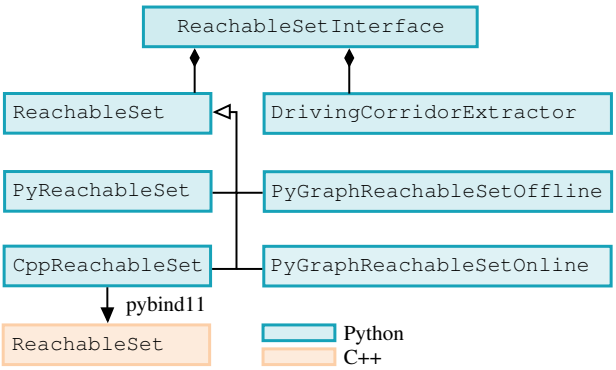


Fig. 5: UML class diagram of the core modules of our toolbox.

ability graph $\mathcal{G}_\mathcal{R}$.

B. Reachable Set Computation

1) Polytopic set propagation method: This method is implemented in Python and C++. In addition to [2], we support the computation in a local curvilinear coordinate system of the ego vehicle. The computation executes the following steps, as shown in Fig. 6:

1) Propagation: Each base set $\mathcal{R}_{k-1}^{(i)} \in \mathcal{R}_{k-1}$ of the previous step is propagated according to the system model (1), resulting in the propagated sets $\mathcal{R}_k^{\mathrm{P},(i)}$ and their corresponding drivable areas $\mathcal{D}_k^{\mathrm{P},(i)}$ projected onto the position domain.
2) Repartition: $\mathcal{D}_k^{\mathrm{P},(i)}$ are merged and repartitioned using a sweep line algorithm and a segment tree. This step helps reduce the number of rectangles and thus reduces overall computation time.
3) Collision detection: The repartitioned rectangles $\mathcal{D}_k^{\mathrm{P},(q)}$ are checked for collision with obstacles using the CommonRoad Drivability Checker [38]. The colliding rectangles are recursively split into two new equally sized rectangles along their longer axis. This is repeated until there is no more collision or the diagonal of the rectangle is smaller than a user-specified threshold.
4) Creation of new base sets: The new base sets $\mathcal{R}_k^{(j)}$ are created by determining the reachable velocities for the collision-free drivable areas $\mathcal{D}_k^{(j)}$.

We refer the readers to [2, Alg. 1] for more computational details.

2) Graph-based propagation method: The second provided method is the graph-based propagation method presented in [3]. In contrast to the polytopic propagation, the reachable sets are derived by traversing a precomputed reachability graph $\mathcal{G}_\mathcal{R}$ and removing edges that collide with the forbidden states. To use a generic, offline-computed graph for any initial state $\boldsymbol{x}_0$ and forbidden states $\mathcal{F}_k$, we precompute it assuming an initial state $\boldsymbol{x}_{0,\mathrm{off}} = \mathbf{0}$ (four-dimensional zero vector, see (1)), without considering $\mathcal{F}_k$, and by discretizing the reachable sets $\mathcal{R}_k$ using a regular grid in the position plane. We add additional edges in $\mathcal{G}_\mathcal{R}$ representing the reachability between sets $\mathcal{R}_k^{(i)}$ and $\mathcal{R}_{k+\kappa}^{(j)}$ with $\kappa \in \{1, \dots, n_{\mathrm{MS}}\}$ across up to $n_{\mathrm{MS}} \in \mathbb{N}$ steps to avoid the accumulation of discretization errors during the online phase. The online computation subsequently executes the following steps (Fig. 7) to account for the initial state and forbidden states:

1) Translation: To consider a given initial state $\boldsymbol{x}_0$, the offline-computed reachable set $\mathcal{R}_k$ is translated along the trajectory of the zero-input response $\chi_k(\boldsymbol{x}_0, \mathbf{0}_u)$ starting in $\boldsymbol{x}_0$.
2) Determining reachability: To determine the reachability of each set $\mathcal{R}_k^{(i)}$, we check which of the corresponding nodes in $\mathcal{G}_\mathcal{R}$ are connected for all $\kappa \in \{1, \dots, n_{\mathrm{MS}}\}$ to at least one parent node $\mathcal{R}_{k-\kappa}^{(j)}$. Nodes are removed from $\mathcal{G}_\mathcal{R}$ if they are not reachable. This principle is shown in Fig. 8.
3) Removal of forbidden states: Nodes whose corresponding set $\mathcal{R}_k^{(i)}$ intersects with $\mathcal{F}_k$ are removed from $\mathcal{G}_\mathcal{R}$. We perform this check by discretizing obstacles using the same road grid as in the computation of $\mathcal{R}_k$.

After the above steps, we can optionally traverse the graph backward in time, starting at the final step k_f to discard all nodes from which the final set $\mathcal{R}_{k_f}$ cannot be reached. We apply a similar principle as in step 2 and remove nodes $\mathcal{R}_{k-1}^{(j)}$ from $\mathcal{G}_\mathcal{R}$ without any reachable children in $\mathcal{R}_k$.

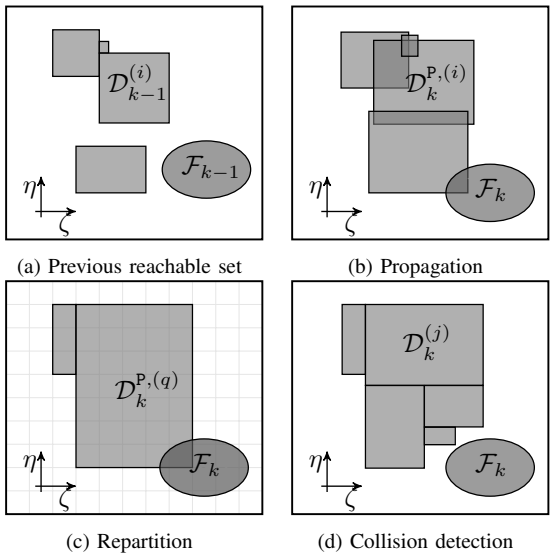


Fig. 6: Selected steps in the polytopic set propagation method. We show the reachable sets projected onto the position domain.

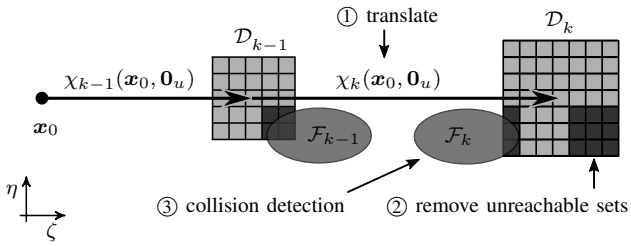


Fig. 7: Three main steps of the graph-based propagation method to remove unreachable nodes from the offline-computed graph. We show the corresponding reachable sets projected onto the position domain.

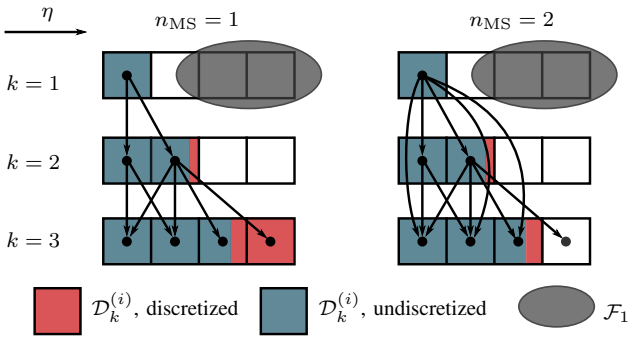


Fig. 8: One-dimensional example for reducing the discretization error of [3]: By using edges across $n_{\mathrm{MS}} > 1$ steps to determine the reachable sets, we prevent the aggregation of discretization errors in intermediate steps. The undiscretized drivable area is shown for comparison.

3) Comparison of the two methods: The polytopic set propagation method can handle time-varying bounds on velocities (2a) and accelerations (2b) of the vehicle originating from, e.g., road curvature, traffic rules, and handcrafted rules [21]. In contrast, this is not possible in the graph-based method due to the fixed parameters in the precomputation of the reachability graph. This characteristic causes greater over-approximation of the computation results in the curvilinear coordinate systems due to the previously mentioned time-varying velocity and acceleration constraints. On the other hand, the graph-based method skips geometric operations, such as splitting of reachable sets, thus the computation is often more efficient than the polytopic set propagation.

C. Driving Corridor Extraction

Our toolbox can further extract driving corridors from a reachability graph generated from Sec. III-B. To this end, we begin at the final step k_f and use a similar graph traversal procedure described at the end of Sec. III-B.2 with the following modifications: Before checking reachability between sets $\mathcal{R}_k^{(i)}$ and $\mathcal{R}_{k-1}^{(j)}$, we identify the connected sets $\mathcal{C}_k^{(n)}$ at each step k. Then, we perform the graph traversal procedure with $\kappa = 1$, i.e., we check the reachability of nodes within the connected set $\mathcal{C}_k^{(n)}$ over one step. After traversing $\mathcal{G}_{\mathcal{R}}$ backwards in time and identifying the connected sets $\mathcal{C}_k^{(n)}$ for all $k \in \{0, \dots, k_f\}$, the relationships between connected sets $\mathcal{C}_k^{(n)}$ and $\mathcal{C}_{k-1}^{(m)}$ of consecutive steps are stored in a separate graph $\mathcal{G}_C$ in which each path from $\mathcal{C}_0^{(m)}$ to $\mathcal{C}_{k_f}^{(n)}$ corresponds to a driving corridor (see Sec. II-C). Optionally, one can constrain driving corridors to end in a user-specified terminal set $\mathcal{I}_{k_f} \subseteq \mathbb{R}^2$ in the position domain (e.g., a set of goal states).

As an additional option, lateral driving corridors (see Sec. II-C) can be determined using the same procedure by invoking the extraction with a driving corridor and a longitudinal trajectory $(p_{\zeta,0}, \dots, p_{\zeta,k_f})$ computed by, e.g., an optimization-based motion planner. The procedure for extracting the corridor is similar to the description above, with the addition that during backwards traversal of $\mathcal{G}_{\mathcal{R}}$, parent sets $\mathcal{R}_{k-1}^{(j)}$ that do not contain the position $p_{\zeta,k-1}$ of the given longitudinal trajectory are excluded.

TABLE I: Parameters Used in Numerical Experiments for Different Coordinate Systems

Parameter	Unit	Cartesian	Curvilinear
k_f	step	30	30
Δ_t	s	0.1	0.1
$\overline{v}_\zeta$	$\mathrm{m/s}$	20.0	20.0
$\underline{v}_\zeta$	$\mathrm{m/s}$	-20.0	0
$\overline{v}_\eta$	$\mathrm{m/s}$	20.0	4.0
$\underline{v}_\eta$	$\mathrm{m/s}$	-20.0	-4.0
$\overline{a}_\zeta$	$\mathrm{m/s^2}$	6.0	6.0
$\underline{a}_\zeta$	$\mathrm{m/s^2}$	-6.0	-6.0
$\overline{a}_\eta$	$\mathrm{m/s^2}$	6.0	2.0
$\underline{a}_\eta$	$\mathrm{m/s^2}$	-6.0	-2.0

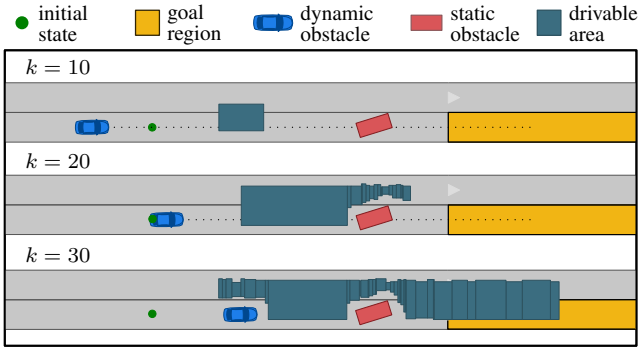


Fig. 9: Drivable areas in scenario I at different steps.

IV. Numerical Experiments

In this section, we demonstrate the key features of our toolbox by numerical experiments using three different scenarios from the CommonRoad benchmark suite with the following benchmark IDs:

I Urban road: `ZAM_Test-1_1_T-1:2020a`
II Intersection: `ARG_Carcarana-1_1_T-1:2020a`
III Highway: `USA_US101-6_1_T-1:2020a`

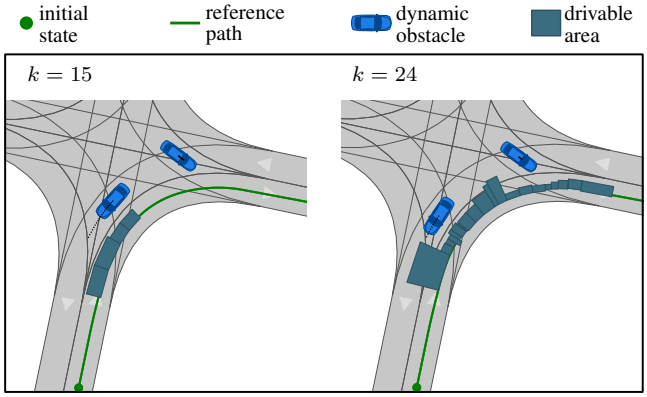
We list the main parameters used in the experiments for both coordinate systems in Tab. I. The animations of the experiments can be found at *https://mediatum.ub.tum.de/1662399*.

A. Scenario I: Urban road

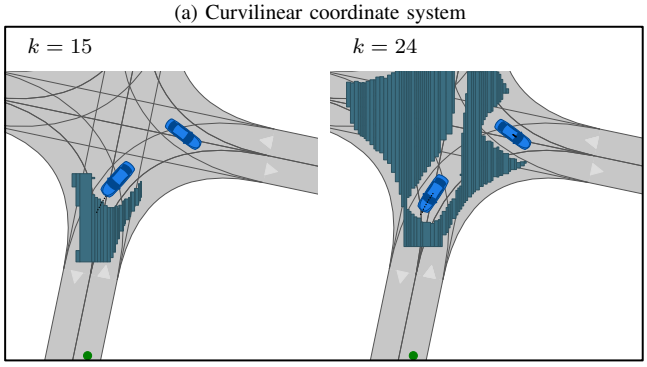
The first scenario illustrates an urban driving situation with a static obstacle (e.g., a parked vehicle) in front of the ego vehicle and another vehicle following the ego vehicle. We demonstrate the computation of reachable sets for the ego vehicle in the Cartesian coordinate system. Fig. 9 shows the collision-free drivable areas for selected steps, which consider both static and time-varying occupancies of obstacles. It can be seen that the drivable areas detect the narrow passage on the left side of the static obstacle.

B. Scenario II: Intersection

Our second scenario is a four-way intersection with the presence of two other vehicles. In Fig. 10 we visualize the collision-free drivable areas at steps $k = 15$ and $k = 24$ within the Cartesian coordinate system and a curvilinear coordinate system.



(a) Curvilinear coordinate system



(b) Cartesian coordinate system

Fig. 10: Drivable areas in scenario II at different steps within two coordinate systems. We remind that the parameters used in the computations within the two coordinate systems are different (see Sec. II-A and Tab. I).

We further use this scenario to demonstrate the extraction of driving corridors (Fig. 11). To this end, we investigate the drivable area at the final step $k_f = 30$ in Fig. 11a: We see that the drivable area is disconnected due to the turning vehicle V_1 at the intersection. Thus, the drivable area exhibits two connected sets $\mathcal{C}_{30}^{(1)}$ and $\mathcal{C}_{30}^{(2)}$, each belonging to a separate driving corridor. Starting from the two connected sets in the last step, our toolbox identifies two driving corridors $\mathcal{C}_{\text{brake}}(\cdot)$ and $\mathcal{C}_{\text{turn}}(\cdot)$ for the time interval $[0, 30]$. The extracted corridors are visualized in Fig. 11b and 11c, where the corridors and the occupancy of vehicle V_1 are stacked over time. The two driving corridors correspond to different tactical decisions: In $\mathcal{C}_{\text{brake}}(\cdot)$, the ego vehicle would brake in the middle of the intersection before vehicle V_1. In contrast, corridor $\mathcal{C}_{\text{turn}}(\cdot)$ represents a maneuver where the ego vehicle would accelerate and continue turning right through the gap between V_1 and the road boundary.

C. Scenario III: Highway

Fig. 12 shows a highway scenario that is created from the NGSIM dataset [39]. We carry out the computation in a curvilinear coordinate system using a planned route as the reference path. The results show that our toolbox robustly handles the computation of the reachable sets of the ego vehicle in scenarios with multiple dynamic obstacles and can detect narrow gaps between vehicles, see, e.g., $k = 30$ in Fig. 12.

TABLE II: Computation Time Within Cartesian Frame

Method	k_f	Unit	Avg.	Std. Dev.
Polytopic set	30	ms	378	131
Graph-based	30	ms	227	58

TABLE III: Computation Time Within Curvilinear Frame

Method	k_f	Unit	Avg.	Std. Dev.
Polytopic set	30	ms	53	16
Graph-based	30	ms	26	8

D. Computation Time

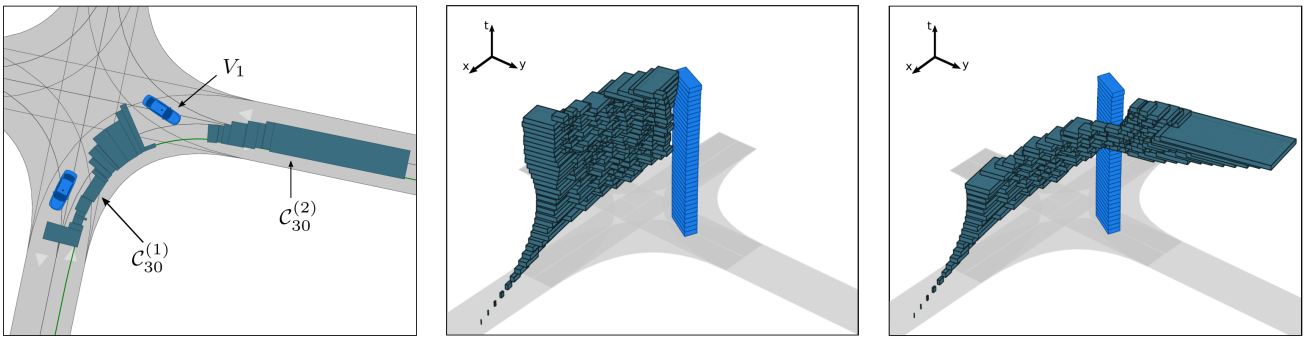
We computed the reachable sets for the planning problems given in 100 randomly chosen scenarios from the CommonRoad benchmark suite to benchmark the performance of our toolbox. All computations were executed on a laptop with an Intel Core i7-7700HQ 2.8 GHz processor. The computation times for the two methods provided in the toolbox are displayed in Tab. II for the Cartesian coordinate system and in Tab. III for curvilinear coordinate systems, respectively. For both propagation methods, computations in the curvilinear coordinate system are faster than in the Cartesian coordinate system. This is because in curvilinear coordinate systems, we used more conservative parameters in both the longitudinal and lateral directions (see Sec. II-A and Tab. I), which resulted in less nodes in the reachability graph and smaller drivable areas. As expected, the average computation times of the graph-based propagation are shorter than those of the polytopic set propagation due to the reasons described in Sec. III-B.3. The computations took a fraction of the planning horizon and render the toolbox suitable for real-time applications.

V. Conclusions

We presented CommonRoad-Reach, an open-source toolbox for computing reachable sets and extracting driving corridors for AVs. Unlike existing tools that offer general-purpose reachability algorithms, our toolbox is tailored to AV-specific applications such as motion planning in arbitrary dynamic traffic scenarios. As a result, our toolbox integrates two methods for the reachable set computation published in [2], [3]. By providing Python and C++ implementations of the algorithms, our toolbox offers both prototyping and real-time capabilities to users. From reachable sets, our toolbox further extracts collision-free driving corridors, which can be used as solution space for motion planners. We used different dynamic traffic scenarios of varied complexity to demonstrate the functionalities of our toolbox, and we benchmarked the real-time functionality against 100 scenarios from the CommonRoad benchmark suite.

Acknowledgments

This work was funded by the Huawei-TUM collaboration project *Research on Key Technologies of Safety Assurance for Autonomous Vehicles*, Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program



(a) Connected sets at $k_f = 30$. (b) Corridor $\mathcal{C}_{\text{brake}}(\cdot)$: Braking before V_1 (c) Corridor $\mathcal{C}_{\text{turn}}(\cdot)$: Turning right in front of V_1

Fig. 11: Identification of two driving corridors corresponding to different driving maneuvers. In figures (b) and (c) we ignore the occupancy of the other vehicle on the opposite lane for visualization purposes.

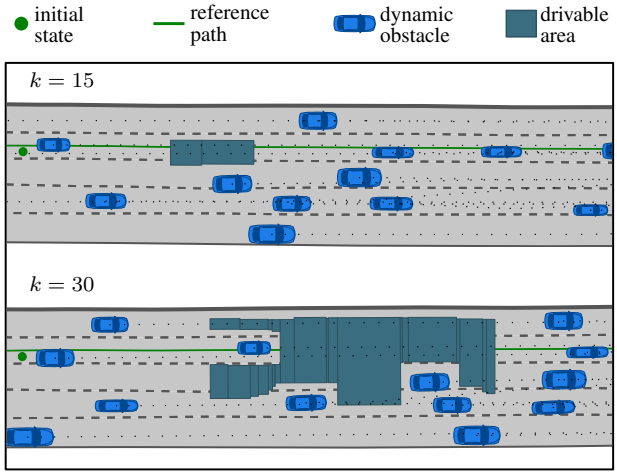


Fig. 12: Drivable areas in scenario III at different steps.

SPP 1835 *Cooperative Interacting Automobiles* under grant No. AL 1185/4-2, the German Federal Ministry of Education and Research (BMBF) within the *Munich Cluster for the Future of Mobility in Metropolitan Regions (MCube)* under grant 03ZU1105AA, and the Central Innovation Programme of the German Federal Government under grant ZF4086013GR9.

References

[1] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annu. Rev. Control Rob. Auton. Syst.*, vol. 4, no. 1, pp. 369–395, 2021.

[2] S. Söntges and M. Althoff, "Computing the drivable area of autonomous road vehicles in dynamic road scenes," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 6, pp. 1855–1866, 2018.

[3] M. Klischat and M. Althoff, "A multi-step approach to accelerate the computation of reachable sets for road vehicles," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2020, pp. 2306–2312.

[4] N. Kochdumper, P. Gassert, and M. Althoff, "Verification of collision avoidance for CommonRoad traffic scenarios," in *Int. Workshop Appl. Verif. Contin. and Hybrid Syst.*, 2021, pp. 184–194.

[5] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Trans. Rob.*, vol. 30, no. 4, pp. 903–918, 2014.

[6] S. Söntges, M. Koschi, and M. Althoff, "Worst-case analysis of the time-to-react using reachable sets," in *Proc. of the IEEE Intell. Veh. Symp.*, 2018, pp. 1891–1897.

[7] A. Lawitzky, A. Nicklas, D. Wollherr, and M. Buss, "Determining states of inevitable collision using reachability analysis," in *Proc. of the IEEE Int. Conf. Intell. Robot. Syst.*, 2014, pp. 4142–4147.

[8] M. Koschi and M. Althoff, "Set-based prediction of traffic participants considering occlusions and traffic rules," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 249–265, 2020.

[9] M. Naumann, H. Konigshof, M. Lauer, and C. Stiller, "Safe but not overcautious motion planning under occlusions and limited sensor range," in *Proc. of the IEEE Intell. Veh. Symp.*, 2019, pp. 140–145.

[10] P. F. Orzechowski, A. Meyer, and M. Lauer, "Tackling occlusions and limited sensor range with set-based safety verification," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2018, pp. 1729–1736.

[11] T. Stahl and F. Diermeyer, "Online verification enabling approval of driving functions – Implementation for a planner of an autonomous race vehicle," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 97–110, 2021.

[12] Y. S. Shao, C. Chen, S. Kousik, and R. Vasudevan, "Reachability-based trajectory safeguard: A safe and fast reinforcement learning safety layer for continuous control," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3663–3670, 2021.

[13] C. Pek and M. Althoff, "Fail-safe motion planning for online verification of autonomous vehicles using convex optimization," *IEEE Trans. Rob.*, vol. 37, no. 3, pp. 798–814, 2020.

[14] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2020, pp. 1–7.

[15] S. Vaskov, H. Larson, S. Kousik, M. Johnson-Roberson, and R. Vasudevan, "Not-at-fault driving in traffic: A reachability-based approach," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2019, pp. 2785–2790.

[16] S. Manzinger, C. Pek, and M. Althoff, "Using reachable sets for trajectory planning of automated vehicles," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 232–248, 2021.

[17] S. Söntges and M. Althoff, "Computing possible driving corridors for automated vehicles," in *Proc. of the IEEE Intell. Veh. Symp.*, 2017, pp. 160–166.

[18] T. Gu, J. M. Dolan, and J.-W. Lee, "Automated tactical maneuver discovery, reasoning and trajectory planning for autonomous driving," in *Proc. of the IEEE Int. Conf. Intell. Robot. Syst.*, 2016, pp. 5474–5480.

[19] P. Bender, O. S. Tas, J. Ziegler, and C. Stiller, "The combinatorial aspect of motion planning: Maneuver variants in structured environments," in *Proc. of the IEEE Intell. Veh. Symp.*, 2015, pp. 1386–1392.

[20] G. Würsching and M. Althoff, "Sampling-based optimal trajectory generation for autonomous vehicles using reachable sets," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2021, pp. 828–835.

[21] E. Irani Liu and M. Althoff, "Computing specification-compliant reachable sets for motion planning of automated vehicles," in *Proc. of the IEEE Intell. Veh. Symp.*, 2021, pp. 1037–1044.

[22] S. Manzinger and M. Althoff, "Tactical decision making for cooperative vehicles using reachable sets," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2018, pp. 444–451.

[23] X. Chen, S. Sankaranarayanan, and E. Abraham, "Flow* 1.2: More effective to play with hybrid systems," in *Int. Workshop Appl. Verif. Contin. and Hybrid Syst.*, 2015, pp. 152–159.

[24] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proc. of the Int. Conf. Comput. Aided Verif.*, 2011, pp. 379–395.

[25] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop Appl. Verif. Contin. and Hybrid Syst.*, 2015, pp. 120–151.

[26] S. Bogomolov, M. Forets, G. Frehse, F. Viry, A. Podelski, and C. Schilling, "Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices," in *Proc. of the Int. Conf. Hybrid Syst.: Comput. and Control*, 2018, pp. 41–50.

[27] S. Bak and P. S. Duggirala, "HyLAA: A tool for computing simulation-equivalent reachability for linear systems," in *Proc. of the Int. Conf. Hybrid Syst.: Comput. and Control*, 2017, pp. 173–178.

[28] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2E2: A verification tool for stateflow models," in *Int. Conf. Tools Algorithms Const. Anal. Syst.*, 2015, pp. 68–82.

[29] I. Xausa, R. Baier, O. Bokanowski, and M. Gerdts, "Computation of avoidance regions for driver assistance systems by using a Hamilton-Jacobi approach," *Optim. Control. Appl. Methods*, vol. 41, no. 2, pp. 668–689, 2020.

[30] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proc. of the Int. Conf. Hybrid Syst.: Comput. and Control*, 2015, pp. 11–20.

[31] K. Margellos and J. Lygeros, "Hamilton-Jacobi formulation for reach-avoid problems with an application to air traffic management," in *Proc. of the Am. Control Conf.*, 2010, pp. 3045–3050.

[32] O. Bokanowski, N. Forcadel, and H. Zidani, "Reachability and minimal times for state constrained nonlinear problems without any controllability assumption," *SIAM J. Control Optim.*, vol. 48, no. 7, pp. 4292–4316, 2010.

[33] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intell. Veh. Symp.*, 2017, pp. 719–726.

[34] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intell. Veh. Symp.*, 2014, pp. 420–425.

[35] B. Schürmann, D. Heß, J. Eilbrecht, O. Stursberg, F. Köster, and M. Althoff, "Ensuring drivability of planned motions using formal methods," in *Proc. of the IEEE Int. Conf. Intell. Transp. Syst.*, 2017, pp. 1–8.

[36] M. Althoff and J. M. Dolan, "Reachability computation of low-order models for the safety verification of high-order road vehicle models," in *Proc. of the Am. Control Conf.*, 2012, pp. 3559–3566.

[37] J. Eilbrecht and O. Stursberg, "Challenges of trajectory planning with integrator models on curved roads," in *Proc. of the IFAC World Congr.*, 2020, pp. 15 588–15 595.

[38] C. Pek, V. Rusinov, S. Manzinger, M. C. Üste, and M. Althoff, "CommonRoad Drivability Checker: Simplifying the development and validation of motion planning algorithms," in *Proc. of the IEEE Intell. Veh. Symp.*, 2020, pp. 1013–1020.

[39] V. Alexiadis, J. Colyar, J. Halkias, R. Hranac, and G. McHale, "The next generation simulation program," *Inst. Transp. Eng. ITE J.*, vol. 74, no. 8, p. 22, 2004.