CommonRoad Drivability Checker: Simplifying the Development and Validation of Motion Planning Algorithms

Christian Pek, Vitaliy Rusinov, Stefanie Manzinger, Murat Can Üste, and Matthias Althoff

***Abstract*— Collision avoidance, kinematic feasibility, and road-compliance must be validated to ensure the drivability of planned motions for autonomous vehicles. Although these tasks are highly repetitive, computationally efficient toolboxes are still unavailable. The CommonRoad Drivability Checker—an open-source toolbox—unifies these mentioned checks. It is compatible with the CommonRoad benchmark suite, which additionally facilitates the development of motion planners. Our toolbox drastically reduces the effort of developing and validating motion planning algorithms. Numerical experiments show that our toolbox is real-time capable and can be used in real test vehicles.**

I. Introduction

The development and validation of motion planning algorithms for autonomous vehicles are challenging because planners should safely operate in any traffic scenario. From 2010 to 2019, the research community published over 25,000 publications[1] to solve the motion planning problem. In general, the proposed planners share a common and safety-relevant goal: generated motions must be drivable.

To ensure drivability (see Fig. 1), generated motions must be 1) collision-free, 2) respect the road boundaries, and 3) be feasible regarding a given vehicle model. Although these checks are repetitive and essential to each motion planning framework, there are still very few open-source libraries available for autonomous vehicles.

We expect that the availability of a drivability checker simplifies the development process and improves the quality of motion planning algorithms. Researchers could focus on the novel aspects of their planners while relying on a tested base of general-purpose operations for motion planning. Because our checker is open-source, it can be audited and improved by anyone to create a computationally efficient and trustworthy library. Similar software projects, such as the *Kinematics and Dynamics Library* (KDL) [1] and the computer vision library *OpenCV* [2], have demonstrated that software quality and safety of robotic systems increase when using community-based libraries.

A. Related work

One of the design goals of our toolbox is the applicability to a vast majority of motion planning approaches. Thus, in the following paragraphs, we briefly review how existing planning approaches ensure drivability.

All authors are with the Department of Informatics, Technical University of Munich, 85748 Garching, Germany. christian.pek@tum.de, ge56cug@tum.de, stefanie.manzinger@tum.de, can.ueste@tum.de, althoff@tum.de

[1]Scopus search query TITLE-ABS-KEY(”vehicle” AND ”planning”) on 04. May 2020.

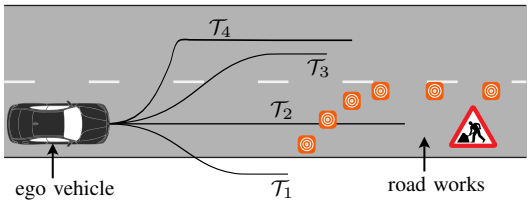


Fig. 1: Ensuring the drivability of planned motions is challenging. To ensure that trajectories are drivable, motion planners must ensure that planned trajectories do not leave the road (see trajectory $\mathcal{T}_1$), do not collide with obstacles (see trajectory $\mathcal{T}_2$), and do not violate vehicle dynamics (see trajectory $\mathcal{T}_4$). Here, only trajectory $\mathcal{T}_3$ is drivable.

a) Collision checking: Checking whether the occupancy of the autonomous vehicle, denoted as *ego vehicle* from here on, intersects with other obstacles is computationally expensive [3]–[7]. To decrease computation effort, the occupancies of obstacles and the ego vehicle are often enclosed by simple set representations [8]. For instance, one of the most common representations are convex polyhedrons [9], e.g., rectangles [10], for which the Separating Axis Theorem [11] can be used to detect pairwise collisions. Circles (and ellipsoids) are yet another way to enclose occupancies of obstacles [12], [13] or the ego vehicle [14], [15]. A set of geometric shapes can also tightly enclose complex occupied regions in the environment [16]–[18]. By enlarging occupancies of obstacles, we can consider bounded position uncertainties related to the sensing and predicting of other traffic participants.

The efficiency of collision checking for large numbers of shapes can be improved by incorporating accelerators such as hierarchies of enclosures [19], [20] (see Fig. 2). These hierarchies are used to over-approximate obstacles with simpler shapes to quickly identify candidate obstacles that might collide with the ego vehicle. Only for collision candidates, the more elaborate set representations are used for collision checking.

Our library supports many popular geometrical shapes and implements state-of-the-art hierarchical representations to speed-up collision checking. It can be easily integrated in many planning approaches and used to validate if a planned trajectory is indeed collision-free.

b) Road-compliance checking: The ego vehicle must stay within the road network, i.e., it shall not occupy walkways and bike lanes. We refer to this as road compliance. The verification of road compliance is implemented

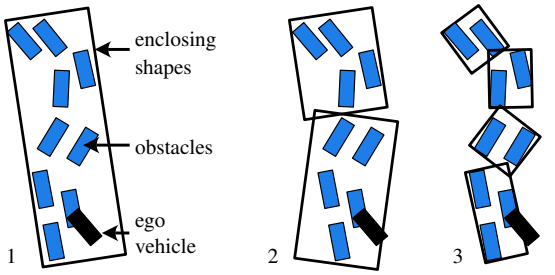


Fig. 2: Example of a boundary volume hierarchy. Obstacles are enclosed by simple shapes to identify candidate obstacles for the collision check with the ego vehicle. From step 1 to step 3, the algorithm uses oriented bounding boxes to identify the candidate obstacles.

differently depending on the utilized trajectory planning technique. Discrete trajectory planning techniques, such as rapidly exploring random trees (RRTs) [21], probabilistic roadmaps [22], or motion primitives [23], [24], usually verify road compliance using collision checks. Therefore, the space outside of the road network can be modeled as static obstacles for collision detection with the ego vehicle. Continuous trajectory planning techniques often incorporate road boundaries as constraints, e.g., by limiting the maximum and minimum allowed lateral deviation from a reference path [15], [25]–[28]. These constraints are usually represented in curvilinear or lane-based coordinate systems [29] and typically assume that roads have a constant width. However, this assumption is generally invalid for many road networks.

Our library supports different methods to check whether trajectories are road-compliant based on collision checks and polygon enclosures. Thus, our methods can be used when planning trajectories with discrete approaches, but also to verify trajectories that are planned by continuous methods under simplified road geometry constraints.

c) Feasibility checking: Trajectories must be feasible regarding a vehicle model. In general, only feasible trajectories can be tracked with adequate performance by vehicle controllers [30], [31]. Depending on the planning approach, we distinguish between three different stages at which feasibility checks are performed: before, during, and after planning.

Before planning: Discrete planning approaches, e.g., using motion primitives [24], perform the feasibility check before planning. Based on a forward simulation of a given vehicle model, trajectory segments are precomputed. These segments are concatenated online to generate feasible trajectories. Because these segments are generated offline, complex and computationally expensive vehicle models can be applied.

During planning: To avoid large precomputed databases, many discrete planning approaches, e.g., RRTs [21], perform the feasibility check online for a given (often simple) vehicle model. They sample states or inputs and check if the resulting trajectories comply with the vehicle model [32], [33].

Continuous planning techniques incorporate the vehicle model as an additional constraint in the optimization. Nevertheless, many optimization-based planners can only incorporate simple vehicle models for computational efficiency. Thus, performing the feasibility check during planning leads to a trade-off between computational efficiency and accuracy of the vehicle model.

After planning: Other discrete planning approaches, e.g., state lattices [34]–[36], first generate a set of intended trajectories by sampling various goal states. However, the resulting trajectories are usually computed without considering a vehicle model, e.g., by generating polynomials or splines. Therefore, the feasibility of each trajectory must be checked afterwards by using simulation.

Our feasibility module provides functionalities to check whether trajectories comply with a given vehicle model before, during, or after planning, and thus, supports various motion planners. Moreover, interfaces enable users to integrate their own models easily.

B. Contributions

This paper presents a novel toolbox for the development and validation of motion planning algorithms for autonomous vehicles. Specifically, the CommonRoad Drivability Checker

1) provides computationally efficient and reliant techniques to verify whether given trajectories are collision-free, respect road boundaries, and are dynamically feasible;
2) is open-source and can be used, modified, and reviewed by anyone;
3) offers easy-to-use Python and C++ interfaces and tutorials for quickly getting started;
4) works out-of-the-box with all scenarios from the CommonRoad benchmark suite [37].

C. Overview of the toolbox

The CommonRoad Drivability Checker (`commonroad_dc`) consists of three core modules to test drivability. The following paragraphs briefly summarize the core functionality of each module.

a) `collision`*:* The collision checker module checks whether given geometric objects (e.g., rectangles or triangles) collide with each other. Based on the geometric representation, we provide a computationally efficient method to represent complex traffic scenarios for collision checking.

b) `boundary`*:* The road boundary module determines the road compliance of a given trajectory by either verifying whether the ego vehicle is still fully enclosed in the road network or performing collision checks with obstacles that model the boundary of the road network. Our module provides different methods (triangulation and the creation of oriented rectangles) to generate road boundary obstacles.

c) `feasibility`*:* The dynamics module builds on top of the vehicle models provided by CommonRoad. It determines the feasibility of a given trajectory by reconstructing the inputs to the corresponding (nonlinear) vehicle model. Trajectories are feasible if the obtained inputs respect the constraints of the vehicle model, e.g., limited steering rate.

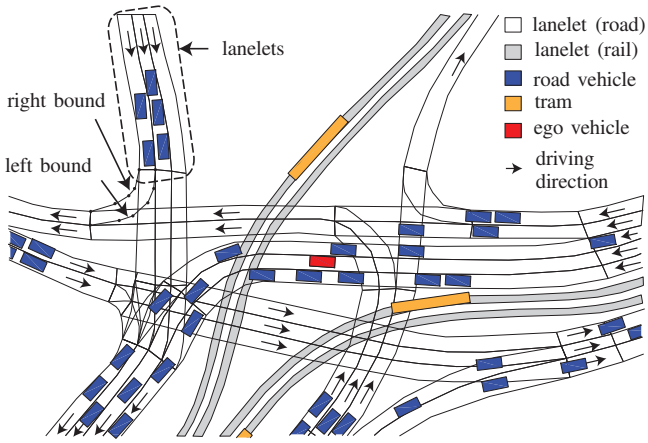


Fig. 3: Lanelets for the CommonRoad scenario DEU_Muc-1_1_T-1. Figure taken from [37].

D. Outline of the paper

The rest of this paper is organized as follows: Sec. II introduces the necessary mathematical definitions and models. Subsequently, we present our solutions for collision, road-compliance, and feasibility checking in Sec. III-V. We demonstrate the benefits of the CommonRoad Drivability Checker in Sec. VI with example scenarios from the CommonRoad benchmark suite. The paper finishes with conclusions in Sec. VII.

II. Preliminaries

Let us introduce the state space $\mathcal{X} \subset \mathbb{R}^n$ as the possible set of states x and $\mathcal{U} \subset \mathbb{R}^m$ as the set of admissible control inputs u of the ego vehicle whose motion is governed by the differential equation

$$\dot{x}(t) = f\big(x(t), u(t)\big). \tag{1}$$

Without loss of generality, we assume that the initial time is $t_0 = 0$, and we adhere to the notation $u\big([t_0, t_h]\big)$ to describe an input trajectory for the time interval $[t_0, t_h], t_0 < t_h$. Furthermore, $\chi\big(t; x(t_0), u\big([t_0, t_h]\big)\big)$ denotes the solution of (1) at time $t \in [t_0, t_h]$ subject to the initial state $x(t_0) = x_0$ and the input trajectory $u\big([t_0, t_h]\big)$. We assume that state trajectories $x\big([t_0, t_h]\big)$ are provided to our toolbox.

We use the following operations between two sets $\mathcal{X}_1$ and $\mathcal{X}_2$: $\mathcal{X}_1 \cup \mathcal{X}_2$ denotes the union of sets, $\mathcal{X}_1 \cap \mathcal{X}_2$ is the intersection of sets, and $\mathcal{X}_1 \setminus \mathcal{X}_2 := \{x_1 \,|\, x_1 \in \mathcal{X}_1 \wedge x_1 \notin \mathcal{X}_2\}$ denotes the set difference. Moreover, the notation $\mathcal{X}_1^{\complement} := \mathbb{R}^n \setminus \mathcal{X}_1$ denotes the complement of a set $\mathcal{X}_1$.

Lanes are represented as lanelets [38], which are "atomic, interconnected, and drivable road segments" [37] (see Fig. 3). Lanelets are defined by their left and right bounds that are modeled as polylines $(p_0, \ldots, p_i, \ldots), i \in \mathbb{N}, p_i \in \mathbb{R}^2$. We form polygons $\mathcal{L}_j$ by connecting the left and right bounds of each lanelet $j \in \{1, \ldots, N_{\text{lanes}}\}$, where N_{lanes} describes the number of lanelets in the road network. The set $\mathcal{D} = \bigcup_{j \in \{1,\ldots,N_{\text{lanes}}\}} \mathcal{L}_j \subset \mathbb{R}^2$ describes the drivable lanes for the autonomous vehicle.

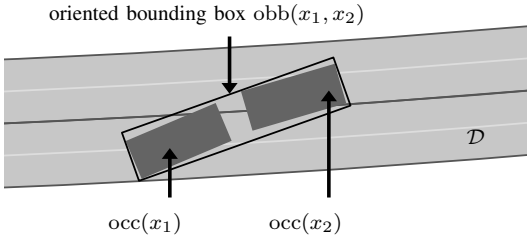


Fig. 4: Oriented bounding box hull around the occupancies $\mathrm{occ}(x_1)$ and $\mathrm{occ}(x_2)$.

Next, we formalize the occupancy of traffic participants. The operator $\mathrm{occ}(x) : \mathcal{X} \to \mathrm{P}\big(\mathbb{R}^2\big)$, where $\mathrm{P}(\mathbb{R}^2)$ is the power set of $\mathbb{R}^2$, relates the state vector x of a traffic participant to the set of points occupied by it. Given a set $\mathcal{X}$, we define $\mathrm{occ}(\mathcal{X}) := \{\mathrm{occ}(x) \,|\, x \in \mathcal{X}\}$. We use the operator $\mathrm{obb}(x_1, x_2)$ to construct an oriented bounding box around the occupancies corresponding to the states x_1 and x_2 as illustrated in Fig. 4.

The occupancy set of all (static and dynamic) obstacles at a given point in time is denoted by $\mathcal{O}(t) \subseteq \mathbb{R}^2$. For the time interval $[t_1, t_2]$, we define $\mathcal{O}\big([t_1, t_2]\big) = \bigcup_{t_1 \le t \le t_2} \mathcal{O}(t)$. For dynamic obstacles, we assume the existence of a prediction that accounts for the future positions of those obstacles over time.

III. Computationally Efficient Collision Detection

Our collision checker library builds upon existing libraries, e.g., FCL [39] and Box2D [40], and supports various collision queries, e.g., binary collision checks or finding all colliding obstacles. In contrast to the existing libraries, we have adapted our library to the domain of autonomous vehicles, i.e., we use specialized structures for representing time-variant traffic scenarios. The interface of our library is designed modularly so that new functionalities can be easily integrated, e.g., from other open-source libraries.

Our collision checker supports intersection tests between basic geometric primitives, such as axis-aligned rectangles, oriented rectangles, triangles, circles, and polygons. Polygons can be non-convex; therefore, we triangulate the interior of polygons and use the resulting mesh for collision detection. Moreover, shape primitives can be grouped into shape-groups. Alongside static obstacles, the collision checker also provides the functionality to consider time-variant obstacles.

It becomes impractical to perform brute-force collision checks [5] as the number of obstacles grows. To reduce the number of checks, the collision checker and shape-groups process collision queries by filtering candidate obstacles for faster collision detection. Our library supports various accelerators (see [5] for a general overview of accelerator structures); in our evaluations (see Sec. VI), dynamic AABB trees (using the implementation from Box2D) and uniform grids (our implementation) performed the best.

In general, planned trajectories are considered collision-free if the ego vehicle does not collide with any (static or

dynamic) obstacle. Given the occupancy set of all obstacles $\mathcal{O}$, we formally define collision-free trajectories as:

Definition 1 (Collision-free Trajectory)
A state trajectory $x\big([t_0, t_h]\big)$ is collision-free with respect to a set of obstacles $\mathcal{O}$ if $\forall t \in [t_0, t_h] : \mathrm{occ}(x(t)) \cap \mathcal{O}(t) = \emptyset$.

As stated in Def. 1, trajectories must be collision-free for all continuous times t. Thus, collision avoidance also must be ensured between two subsequent states $x_1 = x(t_1)$ and $x_2 = x(t_2)$ with $t_1 \leq t_2$. However, only collision checks in discretized time are currently available. We therefore provide the functionality for time-variant objects, i.e., obstacles and the ego vehicle, to construct an oriented bounding box around the occupancies for two consecutive time steps using the operator $\mathrm{obb}(x_1, x_2)$ (see Fig. 4).

IV. Ensuring Road-Compliance

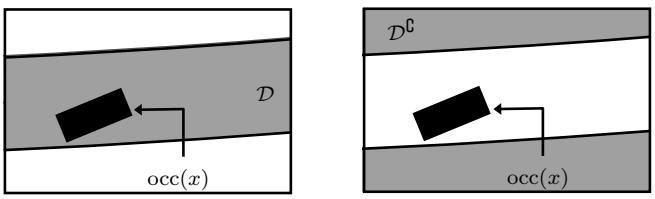
The ego vehicle is not allowed to leave the drivable road network $\mathcal{D}$. To ensure that trajectories do not violate this rule, we define road-compliant trajectories.

Definition 2 (Road-compliant Trajectory)
A state trajectory $x\big([t_0, t_h]\big)$ complies to the road network if $\forall t \in [t_0, t_h] : \mathrm{occ}(x(t)) \subset \mathcal{D}$.

It should be noted that the check in Def. 2 is equivalent to $\mathrm{occ}(x) \cap \mathcal{D}^{\complement} = \emptyset$ (see Fig. 5). Our toolbox implements methods for both checks, which are explained subsequently.

A. Occupancy inclusion

The occupancy inclusion approach (see Fig. 5a) checks whether the occupancy of the ego vehicle along the trajectory $x\big([t_0, t_h]\big)$ is always enclosed in $\mathcal{D}$. We perform the occupancy inclusion check for the two states x_1 and x_2 by computing the set difference between $\mathrm{obb}(x_1, x_2)$ and $\mathcal{D}$, i.e., $\mathrm{obb}(x_1, x_2) \subset \mathcal{D} \Leftrightarrow \mathrm{obb}(x_1, x_2) \setminus \mathcal{D} = \emptyset$. To increase computational efficiency, we partition $\mathcal{D}$ into smaller subsets $\mathcal{D}_j = \mathcal{D} \cap \mathcal{G}_j,\ j \in \mathbb{N}$, using a uniform rectangular grid with cells $\mathcal{G}_j$ (see Fig. 6a). Using collision detection, we identify candidate subsets $\mathcal{D}_j$ by determining grid cells $\mathcal{G}_j$ intersecting $\mathrm{obb}(x_1, x_2)$. Let $c_i \in \mathbb{N}, i \in \{0, 1, 2, \ldots\}$, denote the indices of the candidate subsets. We then perform the check $\mathrm{obb}(x_1, x_2) \setminus \mathcal{D} = \emptyset$ iteratively as $\mathcal{E}_{i+1} := \mathcal{E}_i \setminus \mathcal{D}_{c_i}$, where $\mathcal{E}_1 = \mathrm{obb}(x_1, x_2)$. As soon as we detect $\mathcal{E}_{i+1} = \emptyset$, we abort our algorithm and return that the trajectory respects the road boundaries.



(a) Occupancy inclusion check $\mathrm{occ}(x) \subset \mathcal{D}$.

(b) Collision check $\mathrm{occ}(x) \cap \mathcal{D}^{\complement} = \emptyset$.

Fig. 5: The two general approaches to check road-compliance are complementary to each other: $\mathrm{occ}(x) \subset \mathcal{D} \Leftrightarrow \mathrm{occ}(x) \cap \mathcal{D}^{\complement} = \emptyset$.

The occupancy inclusion approach works well in many scenarios with simple road geometries. However, for complex road networks, the polygons $\mathcal{D}_j$ are represented by large numbers of vertices, increasing computation times.

B. Road-boundary collision check

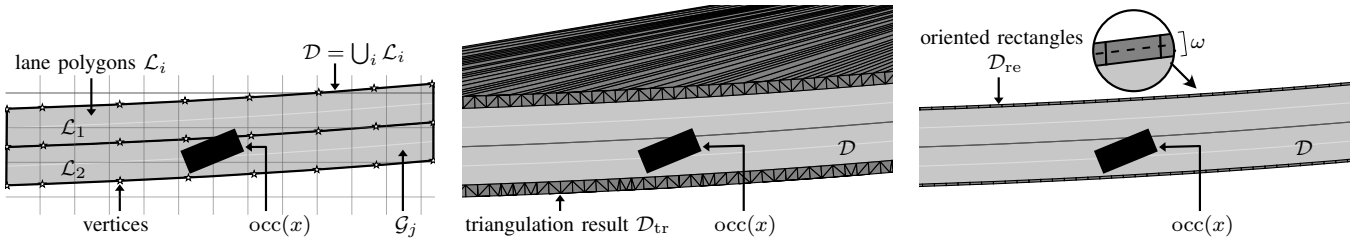
In the road-boundary obstacles approach, we exploit the fact that $\mathrm{obb}(x_1, x_2)$ is road-compliant if the ego vehicle is not located outside of the road network, i.e., $\mathrm{obb}(x_1, x_2) \cap \mathcal{D}^{\complement} = \emptyset$. Since $\mathcal{D}^{\complement}$ is usually unbounded, we compute a finite approximation of $\mathcal{D}^{\complement}$, which often requires checking whether a trajectory leaves $\mathcal{D}$ and enters the finite approximation of $\mathcal{D}^{\complement}$. Otherwise, trajectories starting in $\mathcal{D}^{\complement}$ but outside of our finite approximation could be classified as road-compliant. However, the assumption that the initial state of the trajectory to be verified is road-compliant is valid for most applications.

We approximate $\mathcal{D}^{\complement}$ by obstacles of simple shapes for efficient collision checks [41]. Moreover, the number of collision checks should be small. To achieve both goals, we approximate $\mathcal{D}^{\complement}$ with triangles or oriented rectangles, which we describe in the following paragraphs.

a) Triangulation: We represent $\mathcal{D}^{\complement}$ by a set $\mathcal{D}_{\mathrm{tr}}$ of triangles (see boundary obstacles in Fig. 6b) using Delaunay triangulation [42]. Since some road networks might have roads with complex geometries, triangulations could result in a large number of thin triangles or floating-point errors. We counteract both issues by resampling the lanelet polygons and removing unnecessary vertices by applying the Douglas-Peucker reduction algorithm [43]. Furthermore, we enforce a minimal angle for each triangle (i.e., all angles within a created triangle must be equal or larger than the minimum angle) in the triangulation process to reduce the number of thin triangles. We do not fill $\mathcal{D}^{\complement}$ completely but use a bounding box over-approximating the bounded set $\mathcal{D}$ instead. Using the presented collision checker (see Sec. III), we efficiently determine whether $\mathrm{obb}(x_1, x_2)$ collides with the set $\mathcal{D}_{\mathrm{tr}}$ of triangles.

Fig. 6b shows an example of a given state x. Even though we added countermeasures, some scenarios might still require the creation of a large number of triangles (see triangulation result in Fig. 6b). Therefore, we developed another road-compliance approach as described in the following paragraph.

b) Oriented rectangles: In contrast to triangulation, this method creates a set $\mathcal{D}_{\mathrm{re}}$ of oriented rectangles that separates $\mathcal{D}$ and its complement $\mathcal{D}^{\complement}$. To compute $\mathcal{D}_{\mathrm{re}}$, we first compute the union of all lanelets $\mathcal{L}_j$ in the road network and then extract the inner and outer contours of the resulting polygon. Afterwards, we create an oriented rectangle (see Fig. 6c) for each line segment of the inner and outer contours. The oriented rectangles symmetrically overapproximate each line segment (see dashed line in Fig. 6c). In this way, we can reduce the false positive rate at road forks and merges. The width ω of each box is set to a small user-defined value, e.g., we use $\omega = 1 \times 10^{-5}$ to reduce the rate of false positives. Fig. 6c shows an example of the road-compliance check with boundary obstacles.



(a) Occupancy inclusion. (b) Boundary collision check with triangles. (c) Boundary collision check with rectangles.

Fig. 6: ZAM_Over-1_1:2018b. Road-compliance checks.

C. Comparison of methods

The occupancy inclusion approach (see Sec. IV-A) does not require users to create additional boundary obstacles for each map before the road compliance check. However, for maps with complex road geometries, this approach might require higher computation times. The oriented rectangles approach (see Sec. IV-B) achieves the best computational performance in our experiments (see Sec. VI). However, it under-approximates the drivable space of the road network and thus could result in false-positives in contrast to the accurate results of the occupancy inclusion and triangulation approaches.

V. Feasibility Checking

The ego vehicle can only be precisely controlled if the vehicle model is respected during planning. Vehicle models range from simple models, such as the point-mass model, to complex models, such as a multi-body model [37, Sec. III].

Definition 3 (Feasible Trajectory)
A state trajectory is feasible with respect to a vehicle model (1) *if* $\exists u\big([t_0, t_h]\big) \forall t \in [t_0, t_h] : \chi\big(t; x(t_0), u\big([t_0, t_h]\big)\big) \in \mathcal{X} \wedge u(t) \in \mathcal{U}$.

Our toolbox provides functions to simulate vehicle models and to check whether a given state trajectory is feasible. It currently supports the point-mass, kinematic single track, single track, and multi-body models. However, users can easily add their own vehicle models through pre-defined interfaces.

Let us determine the input u that steers the autonomous vehicle from a state vector $x(t_1)$ at time-step t_1 to a new state $x(t_2)$ with $t_1 < t_2$. We assume that the input $u(t)$ is kept constant for a single time step and construct a constrained two-point boundary value problem:

$$\begin{aligned} &\underset{u}{\operatorname{argmin}}\ J\big(x(t_2), \chi(t_2; x(t_1), u)\big), \\ &\qquad \text{s.t. } \chi(t_2; x(t_1), u) \in \mathcal{X}, u \in \mathcal{U}, \end{aligned} \tag{2}$$

where J is a user-defined cost function. It should be noted that the optimization problem in (2) forms a sequential least squares quadratic program (SLSQP). Even though (2) is in general non-convex, we achieve highly accurate and robust results in our experiments with the Python package SciPy and the algorithm presented in [44]. Our toolbox provides a cost function J that penalizes the position and orientation errors between the optimized and provided states. We accept the solution u if the computed error J is below a certain (user-defined) threshold $e_{\max}$.

Only if we can compute a series of inputs that allows us to closely match the given state trajectory, we accept the trajectory, i.e., $\forall i \in \{0, \dots, N_h - 1\} : J\big(x(t_{i+1}), \chi(t_{i+1}; x(t_i), u_i)\big) \leq e_{\max}$, where u_i is the optimized input between states $x(t_i)$ and $x(t_{i+1})$, and N_h is the number of inputs to reconstruct the state trajectory. Otherwise, we reject the given trajectory.

VI. Numerical Experiments

The presented approaches are integrated into the *CommonRoad Drivability Checker* available at https://CommonRoad.in.tum.de. It serves as a core component of the CommonRoad benchmark system and is used to evaluate the correctness of submitted solutions. We have created multiple tutorials that help users to quickly get an overview of the main functionalities. To the best of the authors' knowledge, there is a lack of available toolboxes with a similar set of features to benchmark our toolbox. Therefore, we highlight the advantages on example scenarios from the CommonRoad scenario database. The presented computation times have been obtained on a single core on a machine with an Intel Xeon Gold 6136 3.00 GHz processor and 128 GB of DDR4 2666 MHz memory.

A. Collision checks

We evaluate our collision checker on 79 different scenarios from the CommonRoad benchmark suite. For that, we generate 1000 trajectories for each scenario using a discrete motion planner involving motion primitives. The trajectories consist of 20 states and have a time discretization (e.g., 0.2 s) as specified in the corresponding CommonRoad scenario. Fig. 7 illustrates some of the generated trajectories for an example highway scenario. We measure the time to check all of the 1000 trajectories for collisions. Over all scenarios, the collision check for 1000 trajectories takes 8.9 ms on average with a standard deviation of 2.3 ms. Tab. I shows individual computation times for selected scenarios. Since our toolbox implements state-of-the-art accelerator structures, we can significantly reduce the required number of checks. In this evaluation, we make use of AABB trees from the library Box2D.

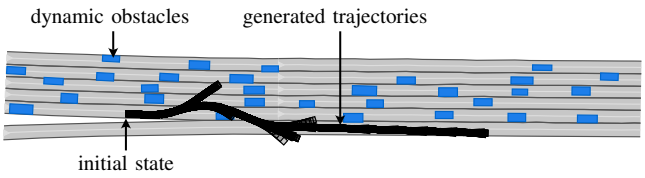


Fig. 7: Collision checking example for generated trajectories in the CommonRoad scenario USA_US101-21_1_T-1.

TABLE I: Selected results of checking 1000 generated trajectories (each 20 time steps) for collisions in scenarios with different numbers of dynamic obstacles.

Scenario	Obs.	Comp. Time
RUS_Bicycle-1_2_T-1	10	10.8 ms
USA_Peach-2_1_T-1	16	5.7 ms
USA_US101-21_1_T-1	30	6.5 ms
CHN_Sha-13_2_T-1	31	6.5 ms
USA_Lanker-2_11_T-1	55	9.4 ms
USA_Lanker-2_12_T-1	83	9.1 ms

TABLE II: Results (average and standard deviation denoted as $\varnothing$ and σ, respectively) of checking 1000 generated trajectories for road compliance.

Data	Inclusion	Triangles	Rectangles
$\varnothing$ time for checking	86.6 ms	18.8 ms	8.1 ms
σ time for checking	39.0 ms	10.6 ms	1.8 ms
$\varnothing$ no. of created shapes	31.5	517.0	149.2
σ no. of created shapes	60.0	665.6	205.0
$\varnothing$ time for creation	17.5 ms	82.2 ms	3.5 ms
σ time for for creation	64.6 ms	193.4 ms	18.3 ms

TABLE III: Results for checking the feasibility of randomly created trajectories. The tests were repeated with 1000 trajectories.

Description	Value
Maximum error e	2×10^{-2}
Precision goal solver	1×10^{-5}
Maximum iterations	100
Mean computation time	1.95 ms
Standard deviation	0.06 ms

B. Road-compliance checks

To benchmark road compliance checks, we use the scenarios and generated trajectories from our previous experiment (see Sec. VI-A). Tab. II summarizes the results of the road compliance checks. To identify candidate objects, e.g., relevant lanelets and oriented rectangles, we use a uniform grid (see Sec. I-A). In our experiments, the oriented rectangle approach achieves the best computation times, which is on average two times faster compared to using triangulation and ten times faster compared to using occupancy inclusion. Fig. 8 illustrates the generated shapes for each of the presented methods in a complex intersection scenario. The number of generated geometric shapes significantly differs among the approaches. On average, our checker has to create 4 times more rectangles and 16 times more triangles than shapes for the occupancy inclusion approach.

C. Feasibility checks

To test our implementation, we created 1.000 input trajectories by randomly sampling different inputs for the supported vehicle models. We tested our approach for all vehicle models, but for brevity, we only present the results for the kinematic single-track model (inputs are steering angle velocity and acceleration). To validate our feasibility check (2), we generate state trajectories by the forward simulation of the kinematic single-track model using known inputs. Fig. 9 shows an example trajectory and the state trajectory that was obtained using our toolbox (denoted as *traced* in Fig. 9). On average, the feasibility check takes 1.95 ms per trajectory on our hardware (see Tab. III).

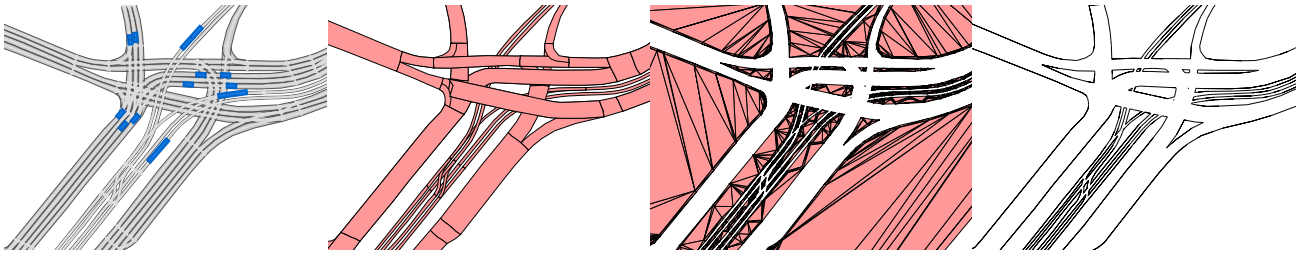
Furthermore, we have evaluated the implementation of our feasibility check with trajectories (and different vehicle models) that were submitted to the CommonRoad benchmark system. Fig. 10a shows the result of an intersection scenario. Although the intended trajectory of the ego vehicle might look feasible, our toolbox detects that the acceleration constraints must be violated to reach the desired position. Our toolbox automatically returns the feasible part of the trajectory so that users can debug their planner. Fig. 10b shows the reconstructed acceleration for the example trajectory and the maximum acceleration constraint for the chosen vehicle model.

VII. Conclusions

We presented the CommonRoad Drivability Checker–an open-source toolbox to check the drivability of trajectories for autonomous vehicles in real-time. Our collision checking module supports many geometric shapes and implements advanced algorithms to quickly determine candidate objects for collision detection. The road-compliance module provides users with three different methods, which obtain accurate results and can be adjusted to the considered road geometries. Lastly, our feasibility module offers the possibility to check whether trajectories are feasible for a certain vehicle model. This model can be chosen from a library of implemented models, or the user can implement it. Since our toolbox is fully compatible with the CommonRoad benchmark suite, users can use a large set of realistic traffic scenarios.

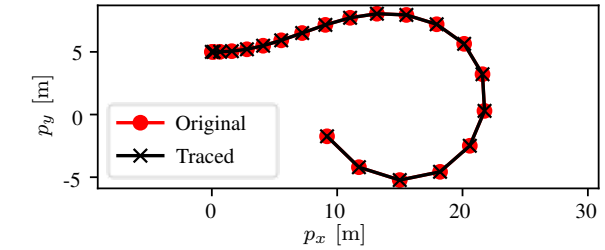
Acknowledgments

The authors gratefully acknowledge partial financial support by the project justITSELF funded by the European Research Council (ERC) under grant agreement No817629, the project *CoInCiDE* within the priority program Cooperative Interacting Automobiles funded by the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant agreement AL 1185/4-1, and the project *interACT* within the EU Horizon 2020 program under grant agreement No 723395.

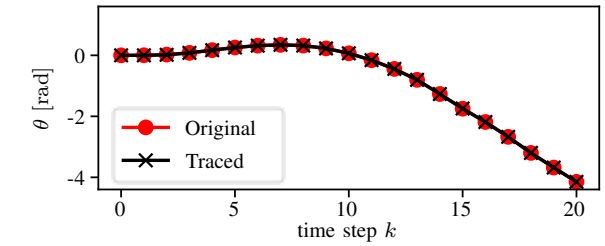


(a) Original scenario. (b) Polygons ($N_{\text{pol}} = 88$). (c) Triangles ($N_{\text{tri}} = 3862$). (d) Rectangles ($N_{\text{rec}} = 1326$).

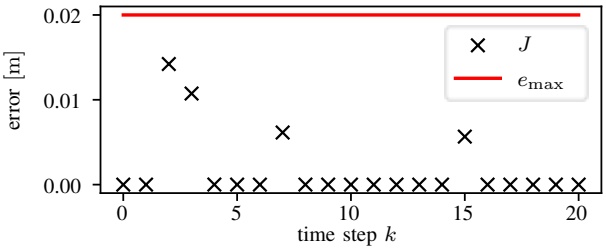
Fig. 8: Created geometric shapes in the different road-compliance checks for the CommonRoad scenario DEU_Muc-1_1_T-1.



(a) Position.

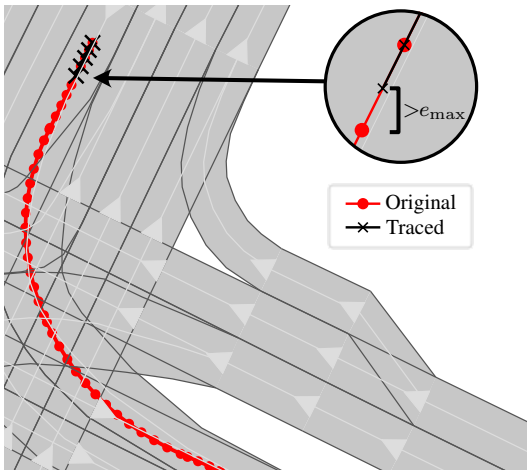


(b) Orientation.

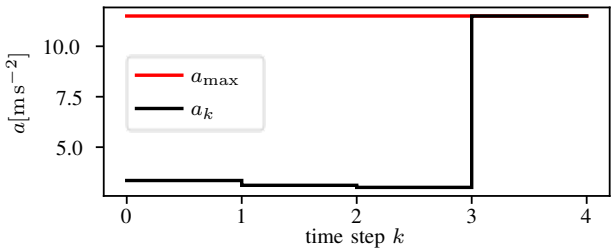


(c) Error in long. position.

Fig. 9: Feasibility check of an example trajectory with positions.



(a) Scenario with trajectory.



(b) Reconstructed acceleration.

Fig. 10: Result of the feasibility check for a planned trajectory in the CommonRoad scenario USA_Lanker-1_2_T-1-2018b. (a) The scenario and the planned and traced trajectory of the ego vehicle. (b) The reconstructed acceleration a_k for different time steps k of the trajectory. The transition to the fourth time step is infeasible, since the maximum acceleration must be violated to reach the target position (see closeup in a).

References

[1] R. Smits, "KDL: Kinematics and Dynamics Library," http://www.orocos.org/kdl.

[2] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[3] P. Jiménez, F. Thomas, and C. Torras, *Robot Motion Planning and Control*. Springer, 1998, ch. Collision Detection Algorithms for Motion Planning, pp. 305–343.

[4] G. Tanzmeister, M. Friedl, D. Wollherr, and M. Buss, "Efficient evaluation of collisions and costs on grid maps for autonomous vehicle motion planning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 2249–2260, 2014.

[5] C. Ericson, *Real-Time Collision Detection*, ser. Morgan Kaufmann series in interactive 3D technology. Morgan Kaufmann, 2005.

[6] A. Rizaldi, S. Söntges, and M. Althoff, "On time-memory trade-off for collision detection," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2015, pp. 1173–1180.

[7] B. C. Heinrich, D. Fassbender, and H.-J. Wuensche, "Faster collision checks for car-like robot motion planning," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 2018, pp. 7533–7538.

[8] S. Thrun, *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann, 2002, ch. Robotic Mapping: A Survey, pp. 1–35.

[9] B. Grünbaum, *Convex Polytopes*, S. Axler, F. W. Gehring, and K. A. Ribet, Eds. Springer, 2003.

[10] D. Ferguson, M. Darms, C. Urmson, and S. Kolski, "Detection, prediction, and avoidance of dynamic obstacles in urban environments," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2008, pp. 1149–1154.

[11] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proc. of the Annual Conf. on Computer Graphics and Interactive Techniques*, 1996, pp. 171–180.

[12] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2010, pp. 518–522.

[13] Y.-K. Choi, W. Wang, Y. Liu, and M.-S. Kim, "Continuous colli-

sion detection for two moving elliptic disks," *IEEE Transactions on Robotics*, vol. 22, no. 2, pp. 213–224, 2006.
[14] B. Gutjahr, C. Pek, L. Gröll, and M. Werling, "Efficient trajectory optimization for vehicles using quadratic programming," *at - Automatisierungstechnik*, vol. 64, no. 10, pp. 786–794, 2016.
[15] C. Pek and M. Althoff, "Computationally efficient fail-safe trajectory planning for self-driving vehicles using convex optimization," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 1447–1454.
[16] B. Vanholme, D. Gruyer, B. Lusetti, S. Glaser, and S. Mammar, "Highly automated driving on highways based on legal safety," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 333–347, 2013.
[17] T. Weiherer, S. Bouzouraa, and U. Hofmann, "An interval based representation of occupancy information for driver assistance systems," in *Proc. of the Int. Conf. on Intelligent Transportation Systems*, 2013, pp. 21 – 27.
[18] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.
[19] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.
[20] B. Martínez-Salvador, A. P. del Pobil, and M. Pérez-Francisco, "A hierarchy of detail for fast collision detection," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 2000, pp. 745 – 750.
[21] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 473 – 479.
[22] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.
[23] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.
[24] D. Heß, M. Althoff, and T. Sattel, "Formal verification of maneuver automata for parameterized motion primitives," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 2014, pp. 1474–1481.
[25] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha – A local, continuous method," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 450–457.
[26] B. Gutjahr, L. Gröll, and M. Werling, "Lateral vehicle trajectory optimization using constrained linear time-varying MPC," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1586–1595, 2017.
[27] J. Nilsson, M. Brännström, J. Fredriksson, and E. Coelingh, "Longitudinal and lateral control for automated yielding maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 5, pp. 1404–1414, 2016.
[28] W. Zhan, J. Chen, C.-Y. Chan, C. Liu, and M. Tomizuka, "Spatially-partitioned environmental representation and planning architecture for on-road autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 632–639.
[29] E. Héry, S. Masi, P. Xu, and P. Bonnifait, "Map-based curvilinear coordinates for autonomous vehicles," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2017, pp. 1–7.
[30] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.
[31] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.
[32] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 2011, pp. 2172–2179.
[33] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
[34] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *The Int. Journal of Robotics Research*, vol. 31, no. 3, pp. 346–359, 2012.
[35] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Int. Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, Sep. 2008.
[36] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, 2009, pp. 1879–1884.
[37] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.
[38] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 420–425.
[39] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2012, pp. 3859–3866.
[40] E. Catto, "Box2D: A 2D physics engine for games," https://box2d.org.
[41] A. Zhu, S. Manzinger, and M. Althoff, "Evaluating location compliance approaches for automated road vehicles," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2018, pp. 642–649.
[42] J. R. Shewchuk, "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, 1996, vol. 1148, pp. 203–222.
[43] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.
[44] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt*, 1988.