Automatic Generation of Safety-Critical Test Scenarios for Collision Avoidance of Road Vehicles

Matthias Althoff and Sebastian Lutz

***Abstract*— It is apparent that one cannot rely solely on physical test drives for ensuring the correct functionality of autonomous vehicles. Since physical test drives are costly and time consuming, it is advantageous to accompany them with computer simulations. However, since most traffic scenarios are not challenging, even simulations are often too time consuming. To address this issue, we present an approach that creates automatically critical driving situations, i.e., situations with a small solution space for avoiding a collision. Our approach combines reachability analysis for determining the size of the solution space with optimization techniques to shrink it. The solution space is reduced by shifting the initial states of traffic participants, demanding an immediate and correct action of the vehicle under test. We demonstrate our approach by automatically increasing the criticality of several initially uncritical situations recorded from real traffic.**

I. Introduction

Testing is an integral part in the development processes of the automotive industry [1]. Interfaces of many traditional automotive systems are well defined, e.g., for anti-lock braking, yaw stabilization, or adaptive cruise control, to name only a few. In autonomous driving, however, all possible environments in which a vehicle will drive are unknown at design time so that vehicles will encounter situations that have never been tested before. This is one of the main reasons why one has to test autonomous road vehicles for 440 million km to demonstrate that they have a better performance than humans with a 95% confidence level [2]. This translates to 12.5 years of test driving with a fleet of 100 test vehicles continuously driving. While this is already extremely costly, the testing effort is based on the assumption that the developed system is not changed during the testing period; changes of the system would additionally prolongate the testing phase.

While efforts with physical test fleets are steadily increased [3]–[5], it is obvious that physical tests alone are too costly and time consuming. For prototyping new ideas more quickly, scaled-down vehicles are sometimes used [6], [7]. The main direction taken by industry and academia, however, is to accompany physical tests with virtual driving tests, requiring the development of a simulation environment in which vehicles can be tested by many factors faster than in real-time, in order to expose the vehicle under test—in this work referred to as the ego vehicle—to many different situations, see e.g., [8]–[10]. Even this approach, however, can be very time consuming since dangerous or interesting situations are rare events; in 2013 in the US, for example, a human driver had to drive around 1.3 million miles in order to obtain a reported injury [2, p. 2]. To further reduce the effort in simulation-based testing, research on the formalization of traffic rules is conducted to automatically determine whether the ego vehicle caused a crash [11]. Testing efforts can be further reduced by formal methods, which exhaustively consider uncertainties from initial states, disturbances, and sensor noise; either during design time [12], [13] or runtime [14]. Nevertheless, formal methods also require interesting driving situations for validation purposes. It is thus of great importance to automatically generate interesting test cases. These test cases can either be applied to the above-mentioned simulation environments, on proving grounds [15], or in a mixed reality, where some vehicles are real and others are added virtually [16].

Automatic or semi-automatic generation of interesting test cases is a long-standing research area in software engineering [17] so that we first review works in that area before presenting applications to autonomous vehicles. Many approaches have been developed to generate test cases for discrete systems found in software engineering, see e.g., [18]–[20]. When adding continuous variables, the situation becomes much more complicated, timed systems with clocks being the simplest extension [21]. However, for automated driving, complicated vehicle dynamics must be added, requiring techniques that work with mixed discrete/continuous systems, also known as hybrid systems. The approaches in [22]–[26] have been developed for general hybrid systems with rather fixed interfaces so that they are not directly applicable for generating interesting driving situations.

Only a few techniques have been developed for automatic test case generation of hybrid dynamics addressing the needs of autonomous systems. In [27] proprietary test case generation for automated vehicles was developed based on the S-TaLiRo tool. An approach to infer the vehicle intelligence level from a finite number of tests is presented in [28]; however, this work does not consider fully automatic test case generation. While S-TaLiRo offers many optimization engines to create interesting situations, other approaches focus on machine learning techniques: Interesting scenarios are grouped by unsupervised learning to find situations where small deviations of the environment lead to great performance variations in [29]. Combination and mutation of recorded data is used in [30] to create new test cases.

Somewhat related to automatic test case generation for motion planning are the following topics: a) fault detection strategies, which help automating tests [31]; b) tools for the

All authors are with the Technische Universität München, Fakultät für Informatik, Lehrstuhl für Robotik und Echtzeitsysteme, Boltzmannstraße 3, 85748, Garching, Germany. {althoff, sebastian.lutz}@tum.de

statistical analysis of mission goals indirectly supporting the search for interesting test cases [32]; and c) testing of low-level controllers of autonomous vehicles [33]–[35].

None of the previous approaches are based on constructive algorithms, ensuring that generated situations have a small and quantifiable solution space. Our test case generation is the first that quantifies the solution space by explicitly computing the drivable area of the ego vehicle. We construct scenarios so that the drivable area is optimized towards a user-defined value, making it possible to obtain situations with a desired criticality in terms of drivable area.

The paper is organized as follows: Sec. II provides a detailed problem statement followed by Sec. III presenting an overview of our proposed solution. The optimization approach for creating critical scenarios is described in Sec. IV. We demonstrate the applicability of the optimization routine using scenarios that are originally not critical in Sec. V followed by conclusions in Sec. VI.

II. Problem Statement

The goal of this paper is to automatically generate traffic situations for which it is hard to obtain a safe motion plan. We first specify the motion planning problem similarly to [36], followed by defining the drivable area of the ego vehicle. Let us denote by $f(x(t), u(t))$ the right hand side of the state space model of the ego vehicle so that

$$\dot{x}(t) = f(x(t), u(t)),$$

where $x \in \mathbb{R}^n$ is the state vector and $u \in \mathbb{R}^m$ is the input vector. We further require the initial state $x_0 \in \mathbb{R}^n$ ($x(t_0) = x_0$), the initial time t_0, and the final time t_f. The time-varying, allowed space on the road surface is denoted by $\mathcal{W}_{\texttt{free}}(t) \subset \mathbb{R}^2$. Let us introduce the power set $P()$ for the function $O(x(t)) : \mathbb{R}^n \rightarrow P(\mathbb{R}^2)$ returning the occupancy of a vehicle, which has to lie within the free space: $\forall t \in [t_0, t_f] : O(x(t)) \subseteq \mathcal{W}_{\texttt{free}}(t)$. We also require constraints $g(x(t), u(t), t) \leq 0$, such as speed limits or other traffic rules [11]. Equality constraints can be constructed from inequality constraints (e.g., $x \leq 0 \wedge -x \leq 0 \equiv x = 0$). The goal region $\mathcal{G} \subset \mathbb{R}^n$ can consist of disjoint sets and as soon as $x(t) \in \mathcal{G}$ at time $t = t_f$, a feasible solution is found. After introducing an input trajectory as $u(\cdot)$ (in contrast to a value $u(t)$ at time t) and the cost function of the obtained solution $J(x(t), u(t), t_0, t_f)$, we can finally formulate the motion planning problem as finding

$$u^*(\cdot) = \underset{u(\cdot)}{\arg\min}\ J(x(t), u(t), t_0, t_f)$$

subject to

$$\begin{aligned} &\dot{x}(t) = f(x(t), u(t)), \quad O(x(t)) \subseteq \mathcal{W}_{\texttt{free}}(t), \\ &g(x(t), u(t), t) \leq 0, \quad x(t_0) = x_0, \quad x(t_f) \in \mathcal{G}. \end{aligned} \tag{1}$$

We denote a feasible (not necessarily optimal solution) as $\chi(t; x_0, u(\cdot))$ which meets all constraints in (1). Since we are not only interested in the optimal solution, but in the space of solutions, we require the set of reachable states [37]. In particular, we use a so-called *anticipated reachable set*, which excludes states that will inevitably result in an accident. After introducing the set of initial states $\mathcal{X}_0$, the set of input trajectories $\mathcal{U}$, and the time horizon $t_h \geq t_f$, we define the anticipated reachable set as

$$\mathcal{R}(t; \mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot), t_h) = \Big\{ \chi(t; x_0, u(\cdot)) \Big| \exists u(\cdot) \in \mathcal{U}, \exists x_0 \in \mathcal{X}_0, O\big(\chi(\tau; x_0, u(\cdot))\big) \subseteq \mathcal{W}_{\texttt{free}}(\tau) \text{ for } \tau \in [t_0, t_h] \Big\}.$$

Using the previously introduced occupied region $O(x)$, the drivable area becomes

$$\mathcal{D}(t; \mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot), t_h) = \bigcup_{x \in \mathcal{R}(t; \mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot), t_h)} O(x).$$

To quantify the solution space over time, we introduce the function $\texttt{area}(\mathcal{D})$, returning the area of the drivable area. For simplicity of notation, we introduce the tuple $S = (\mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot))$ representing a scenario and write

$$A(S, t) = \texttt{area}\big(\mathcal{D}(t; \mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot), t_h)\big).$$

The goal of this work is to create a scenario S with a reference solution space $A_{ref}(t) > 0$ using a weight $w(t) > 0$:

$$\underset{S}{\arg\min} \int_0^{t_h} w(t) \big(A(S, t) - A_{ref}(t)\big)^2. \tag{2}$$

The weight $w(t) > 0$ is time-varying so that one can put more or less emphasis on achieving the reference solution space at different times.

III. Overview of the Approach

In order to solve (2), we optimize the initial states of other traffic participants including the ego vehicle until the desired size of the drivable area is obtained. This principle is illustrated in Fig. 1 for a deliberately simple static scenario, in which only the size of the leftmost static obstacle is changed. One can see that with slight changes, a rather simple motion planning problem quickly becomes hard due to the small remaining drivable area.

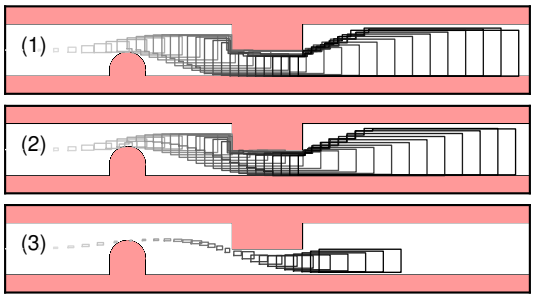


Fig. 1. Drivable area of the ego vehicle driving from left to right for different sizes of the leftmost static obstacle. Since only slower trajectories are feasible in scenario (3), the drivable area does not reach as far right.

When not only considering static obstacles, but also other traffic participants, one has to compute the effect of their changed initial states on their future occupancy. Different types of vehicle predictions are discussed in Sec. III-A.

Given the predictions of other traffic participants, we present in Sec. III-B how the drivable area of the ego vehicle is computed, which is later optimized.

A. Predictions of Other Traffic Participants

Possible future occupancies of other traffic participants have to be removed from the allowed space $\mathcal{W}_{\texttt{free}}(\cdot)$ of the ego vehicle to avoid collisions. There are basically two main options: a) One provides single trajectories of other traffic participants or b) provides a set of possible trajectories. In the first case one creates a scenario in which the motion planner has to find a safe solution when the other trajectory is known, which is typically the case in collaborative driving [38], [39]. In the second case, the future movement of other traffic participants is unknown, which is typically the case for non-communicating traffic participants, such as e.g., bicyclists.

While single trajectories are obtained either from recordings or by using simulation models [36], the computation of a set of behaviors is not straightforward. For the latter case we use the tool SPOT [40]. An example of computing the drivable area in an intersection is presented in Fig. 2 for a specific time interval. Please note that SPOT returns sets for each consecutive time interval. After denoting the free space obtained from road boundaries and static obstacles by $\mathcal{W}_{\texttt{static}}$, the drivable area of other traffic participants by $\mathcal{D}_{other}(\cdot)$, and introducing the set difference between sets $\mathcal{A}$ and $\mathcal{B}$ as $\mathcal{A} \setminus \mathcal{B}$, we obtain $\mathcal{W}_{\texttt{free}}(\cdot) = \mathcal{W}_{\texttt{static}} \setminus \mathcal{D}_{other}(\cdot)$. The obtained free space is used subsequently to obtain the drivable area of the ego vehicle, which additionally considers constraints due to the vehicle dynamics so that it is a subset of $\mathcal{W}_{\texttt{free}}(\cdot)$.

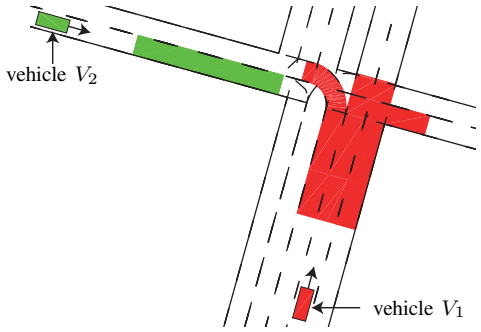


Fig. 2. Drivable areas of vehicles in an intersection within the time interval [1.5,2] s. A detailed scenario description can be found in [40, Sec. V].

B. Computation of the Drivable Area

We compute the drivable area $\mathcal{D}(t; \mathcal{X}_0, \mathcal{W}_{\texttt{free}}(\cdot), t_h)$ according to the approach presented in [41]. To efficiently compute the drivable area, it is represented by the union of Cartesian products of convex polytopes. Each polytope represents a set of position/velocity pairs, which can be reached in the x- and y-direction. Polytopes are chosen because they are closed under intersection, which is required for removing regions from the drivable area possibly occupied by another traffic participant or static obstacle. An example of a drivable area is presented in Fig. 3 of an overtaking scenario for different time steps. For a more detailed description, the reader is referred to [41]. The drivable area is used in the optimization routine below to generate scenarios which have a minimum weighted deviation to a desired drivable area according to (2).

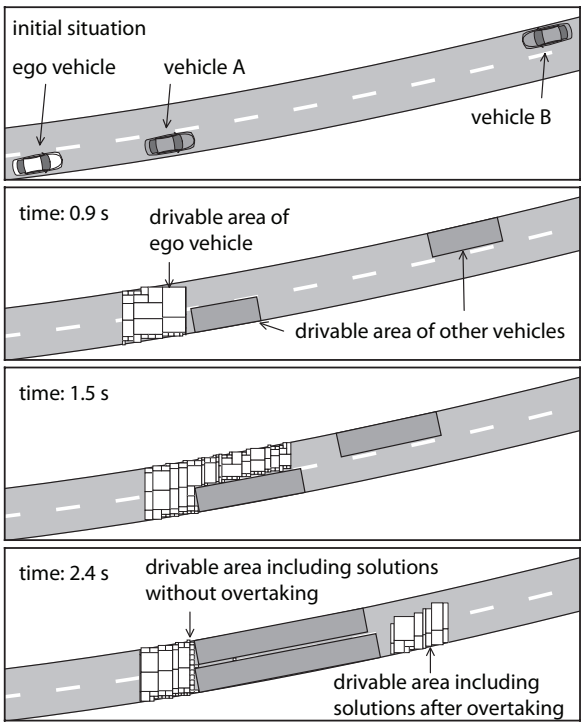


Fig. 3. Drivable area at different time steps for an overtaking scenario. The ego vehicle can either stay behind car A or barely overtake it. Due to illustration reasons, only the positions and not the velocities are shown.

IV. Optimization Routine

In order to solve a finite optimization problem, we discretize the time equidistantly to $t_k = \Delta t \cdot k$, $\Delta t \in \mathbb{R}^+$ being the step size and $k \in \mathbb{N}$ the time step. Thus, the reference area over time can be represented as a q-dimensional vector with entries $a_{ref,k} = A_{ref}(t_k)$, where $q = \lceil (t_h - t_0)/\Delta t \rceil$ and $\lceil \alpha \rceil$ returns the smallest integer greater than or equal to $\alpha \in \mathbb{R}$. Additionally, we introduce the operator $\gamma(S)$ returning the area profile, i.e., the development of the drivable area over discrete times t_k:

$$\gamma(S) := \left[a_1, a_2, \ldots, a_q\right], \quad a_k = A(S, t_k).$$

Thus, the discrete-time approximation of the optimization problem in (2) can be written in matrix notation as

$$\arg\min_{S} (\gamma(S) - a_{ref})^T W (\gamma(S) - a_{ref}), \tag{3}$$

where $W = \texttt{diag}([w(t_1), \ldots, w(t_q)])$ and $\texttt{diag}(\xi)$ returns the diagonal matrix of a vector ξ. Since motion planning problems in two dimensions can be invariant with respect to position and orientation (i.e., when the vehicle is in an unstructured environment (e.g., parking lot), all vehicles can be translated and rotated without changing the problem), we fix the position and orientation of the ego vehicle for

solving (3). As a consequence, only remaining states of the ego vehicle, such as velocity, are optimized, while for all other vehicles all initial states are optimized. In the future, we will automatically detect whether the problem is invariant with respect to position and orientation and remove the above restriction depending on the situation. Also, please note that in this work, we only optimize the initial states of vehicles, while the optimization of their individual movement is the subject of future work. We first rewrite (3) as a quadratic programming problem and later extend it using binary search.

A. Formulation as a Quadratic Programming Problem

The influence of the scenario on the drivable area cannot be described explicitly, which would be required when using numerical optimization routines. Instead, we construct a local model by first estimating the influence of each traffic participant on the drivable area individually. Let us define a variation of the i^{th} component of the initial state of vehicle V_j by $\Delta x_{0,i}^{(j)}$. Please note that one of the vehicles $V_1, \ldots, V_p$ is the ego vehicle and that we introduce p as the total number of considered traffic participants. Next, we denote the initial scenario by S_0 and define the operator $S(\langle i, j\rangle, \Delta\tilde{x}_{0,i}^{(j)})$ returning an updated scenario after the i^{th} initial state of vehicle V_j has changed to $x_{0,i}^{(j)} := x_{0,i}^{(j)} + \Delta\tilde{x}_{0,i}^{(j)}$. The tilde on $\Delta\tilde{x}_{0,i}^{(j)}$ is used to distinguish this change from the change $\Delta x_{0,i}^{(j)}$ actually realized later during optimization. We define the change in the area profile as

$$b_i^{(j)} := \frac{\gamma\big(S\big(\langle i, j\rangle, \Delta\tilde{x}_{0,i}^{(j)}\big)\big) - \gamma(S_0)}{\Delta\tilde{x}_{0,i}^{(j)}}, \tag{4}$$

where $b_i^{(j)} \in \mathbb{R}^q$. Assuming a linear relation between the change of the initial state and the obtained area profile in the vicinity of the original scenario S_0, we obtain for $\Delta x_{0,i}^{(j)}$ the new area profile

$$\gamma\big(S(\langle i, j\rangle, \Delta x_{0,i}^{(j)})\big) \approx \gamma(S_0) + b_i^{(j)} \Delta x_{0,i}^{(j)}. \tag{5}$$

After introducing $n(j)$ as the number of state variables of the j^{th} traffic participant, we further simplify the notation by stacking all $b_i^{(j)}$ values in a $q \times r$ matrix ($r = \sum_{j=1}^{p} n(j)$) to obtain

$$B = [b_1^{(1)}, b_2^{(1)}, \ldots, b_{n(1)}^{(1)}, b_1^{(2)}, \ldots, b_{n(2)}^{(2)}, \ldots, b_{n(p)}^{(p)}] \in \mathbb{R}^{q \times r} \tag{6}$$

and all variations of initial states analogously in

$$\Delta x_0 = [\Delta x_{0,1}^{(1)}, \Delta x_{0,2}^{(1)}, \ldots, \Delta x_{0,n(1)}^{(1)}, \ldots, \Delta x_{0,n(p)}^{(p)}] \in \mathbb{R}^r. \tag{7}$$

We write $S(\Delta x_0)$ to represent a new scenario when the change is represented by the vector Δx_0, which is approximated by the sum of individual changes from (5):

$$\gamma(S(\Delta x_0)) \approx \gamma(S_0) + \sum_{j=1}^{p} \sum_{i=1}^{n(j)} b_i^{(j)} \Delta x_{0,i}^{(j)}. \tag{8}$$

Using (6) and (7), we can write (8) as

$$\gamma(S(\Delta x_0)) \approx \gamma(S_0) + B\Delta x_0. \tag{9}$$

Inserting (9) into (3) and introducing $\Delta a_0 := \gamma(S_0) - a_{ref}$ results in the following quadratic programming problem:

$$\begin{aligned}
&\arg\min_{\Delta x_0} (\gamma(S_0) + B\Delta x_0 - a_{ref})^T W (\gamma(S_0) + B\Delta x_0 - a_{ref}) \\
&= \arg\min_{\Delta x_0} (\Delta a_0 + B\Delta x_0)^T W (\Delta a_0 + B\Delta x_0) \\
&= \arg\min_{\Delta x_0} (B\Delta x_0)^T W (B\Delta x_0) + \Delta a_0^T W (B\Delta x_0) \\
&\quad + (B\Delta x_0)^T W \Delta a_0 + \Delta a_0^T W \Delta a_0 \\
&= \arg\min_{\Delta x_0} \Delta x_0^T \tilde{W} \Delta x_0 + c^T \Delta x_0,
\end{aligned} \tag{10}$$

where

$$\begin{aligned}
\tilde{W} &= B^T W B, \\
c &= \Delta a_0^T (WB + W^T B).
\end{aligned} \tag{11}$$

Please note that $\Delta a_0^T W \Delta a_0$ can be removed in (10) since it only shifts the value of the optimization problem: We are only interested in the Δx_0 minimizing the area and not in the minimum cost value, so that this offset can be removed. As a result, we obtain an optimization problem over the shift of the initial state Δx_0 instead of an optimization problem of abstract scenarios S as originally formulated in (3).

So far, we have not considered constraints, but clearly we have to restrict the velocity of each vehicle to be within $[0, v_{max}]$, where v_{max} is the maximum allowed velocity on the road or a maximum velocity that considers a certain amount of overspeeding. We also add the constraint that the area profile should always be positive; although the true area is always positive, negative values can be computed due to the assumption on linearity in (5). Let $\texttt{velInd}(j)$ return the index of the state $x^{(j)}$ representing the velocity of the j^{th} vehicle. We can now formulate the following constraints:

$$\begin{aligned}
&\forall j : x_{0,\texttt{velInd}(j)}^{(j)} + \Delta x_{0,\texttt{velInd}(j)}^{(j)} \in [0, v_{max}] \qquad \text{(velocity)}, \\
&\forall i \in [0, q] : \gamma(S_0)_i + (B\Delta x_0)_i \geq 0 \qquad \text{(area)}.
\end{aligned} \tag{12}$$

If the model of a vehicle does not have velocity as a state (e.g., when it is an input), we make the obvious changes to (12). The minimization of (10) together with the constraints in (12) results in a quadratic program, for which efficient solvers exist [42]. The combination with binary search is addressed in the next subsection.

B. Binary Search

To address the problem that $b_i^{(j)}$ in (4) is undefined when the drivable area becomes empty, we switch to binary search, which does not require computing $b_i^{(j)}$. When a scalar target value lies in between two values, binary search iteratively splits the half where the target lies [43].

In our work, binary search acts as a repair mechanism when the last Δx_0 has caused the solution space to become empty. We denote the initial state before the last quadratic programming update as $x_{0,before}$ and the initial state after the update as $x_{0,after}$. Our binary search implementation in Alg. 1 is terminated once the solution space is re-established

Algorithm 1 `binarySearch`$(x_{0,before}, x_{0,after}, \mu)$

Require: Initial state $x^{(j)}_{0,before,i}$ and $x^{(j)}_{0,after,i}$ before and after last quadratic programming update, iteration limit μ

Ensure: $x^{(j)}_{0,i}$ so that solution space is not empty

1: $b_{max} \leftarrow 0$
2: **for** $j = 1 \dots p$ **do**
3: **for** $i = 1 \dots n(j)$ **do**
4: $b^{(j)}_{abs,i} \leftarrow \left| \frac{\gamma(S(\langle i,j \rangle, \Delta\tilde{x}^{(j)}_{0,i})) - \gamma(S_0)}{\Delta\tilde{x}^{(j)}_{0,i}} \right|$
5: **end for**
6: **end for**
7: $b_{sorted} \leftarrow$ `sort`(b_{abs})
8: **while not** `empty`(b_{sorted}) **do**
9: $vI, sI \leftarrow$ `pop`(b_{sorted})
10: **for** $\theta = 1 \dots \mu$ **do**
11: $x^{(vI)}_{0,curr,sI} \leftarrow 0.5(x^{(vI)}_{0,before,sI} + x^{(vI)}_{0,after,sI})$
12: $a_{curr} \leftarrow \gamma(S(\langle sI, vI \rangle, \Delta x^{(vI)}_{0,curr,sI}))$
13: **if** $\forall l: \quad a_{curr,l} > 0$ **then**
14: **return** $x^{(vI)}_{0,curr,sI}$
15: **else**
16: $x^{(vI)}_{0,after,sI} \leftarrow x^{(vI)}_{0,curr,sI}$
17: **end if**
18: **end for**
19: **end while**
20: **return** $x^{(vI)}_{0,before,sI}$

or after a user-defined limit of μ iterations has been reached. To select the variable subject to binary search, we choose as a heuristic the one with the largest sensitivity, i.e., the variable associated to the largest $b^{(j)}_i$ value in (4) as shown in lines 1-7 of Alg. 1. Please note that the command `sort`(b_{abs}) returns a sorted list of sensitivities. If the binary search on this variable would exceed the iteration limit μ (line 10), the variable with the next lowest sensitivity is changed using binary search (line 9), where `pop`(b_{sorted}) returns the first vehicle index vI and state index sI in the list b_{sorted} and removes them from the list. If all variables of all vehicles exceed the iteration limit, the result before the last quadratic programming update is returned, so that Alg. 1 always re-establishes a solution space.

Please keep in mind that after one variable has been successfully modified via binary search, previous variables which could not be changed anymore can possibly be changed again due to the modified traffic situation. For this reason, after a solution space is regained, the quadratic programing problem in (10) is re-iterated until the difference in the cost function

$$\kappa = (\gamma(S) - a_{ref})^T W (\gamma(S) - a_{ref})$$

between updates is smaller than a user-defined value ϵ. We also set a limit it_{max} for switching between quadratic programming and binary search. The overall algorithm combining quadratic programming and binary search is presented in Alg. 2. There, we use the function `quadProg`(solve (10)), which solves one iteration of the quadratic programming problem and returns the next initial state $x_{0,curr}$, the updated scenario S, and the Boolean variable `success` whether $b^{(j)}_i$ existed for all initial state variables and vehicles. Furthermore, `binarySearch` calls Alg. 1, and `updateScenario`$(S, x^{(j)}_{0,i})$ returns an updated scenario given the previous one S and the new initial state $x^{(j)}_{0,i}$.

Algorithm 2 `optimizedScenario`$(x_0, \epsilon, it_{max}, \mu, W, a_{ref})$

Require: Initial state x_0, threshold ϵ, iteration limit it_{max}, binary search iteration limit μ, weighting matrix W, reference area profile a_{ref}

Ensure: critical scenario S

1: $\kappa_{new} \leftarrow 0, \quad \kappa_{old} \leftarrow -\infty, \quad it \leftarrow 0, \quad x_{0,curr} \leftarrow x_0$
2: **while** $|\kappa_{new} - \kappa_{old}| \geq \epsilon$ **and** $it < it_{max}$ **do**
3: `success` $\leftarrow$ `true`
4: **while** $|\kappa_{new} - \kappa_{old}| \geq \epsilon$ **and** `success` $=$ `true` **do**
5: $\kappa_{old} \leftarrow \kappa_{new}, \quad x_{0,old} \leftarrow x_{0,curr}$
6: $x_{0,curr}, S,$ `success` $\leftarrow$ `quadProg`(solve (10))
7: $\kappa_{new} \leftarrow (\gamma(S) - a_{ref})^T W (\gamma(S) - a_{ref})$
8: **end while**
9: $x^{(v)}_{0,s} \leftarrow$ `binarySearch`$(x_{0,old}, x_{0,curr}, \mu)$
10: $S \leftarrow$ `updateScenario`$(S, x^{(v)}_{0,s})$
11: $\kappa_{new} \leftarrow (\gamma(S) - a_{ref})^T W (\gamma(S) - a_{ref})$
12: $it \leftarrow it + 1$
13: **end while**

V. Numerical Examples

Before we present results, we introduce the representation of the road network. For our benchmarks we use *lanelets* [44] as atomic, interconnected, and drivable road segments to represent the road network. A lanelet is defined by its *left* and *right bound*, where each bound is represented by an array of points. An example of a complicated intersection constructed from lanelets including tram lines is shown in Fig. 4. We have enhanced lanelets by *traffic regulations*, e.g., the speed limit, as presented in [36].

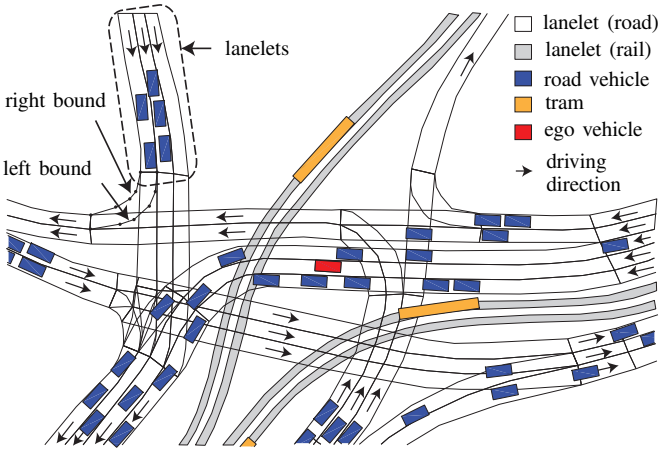


Fig. 4. Lanelets of a complex intersection in the city center of Munich (ID GER_Muc_1a of [36]). Besides roads, tram rails are also modeled.

We investigate three different scenarios S_A, S_B, and S_C, with varying numbers of other traffic participants to illustrate our approach. S_A is a deliberately simple scenario to illustrate the approach, and S_B, S_C are taken from CommonRoad, which is a publicly available set of benchmark scenarios [36]. For all subsequent scenarios we set $\forall t : a_{ref}(t) = 1$ and $w(t) = 1$. To better illustrate the results, we only show the drivable area of the ego vehicle and not of other traffic participants. All optimizations are executed on an Intel Core i7-6500U with 4 cores, which run at 2.5 GHz and the computation times are summarized in Tab. I. The quadratic programming problems $\texttt{quadProg}(\cdot)$ are solved using ECOS [45].

TABLE I
COMPUTATION TIMES

Scenario	$\mathbf{S_A}$	$\mathbf{S_B}$	$\mathbf{S_C}$
Iterations for quadratic programming	1	1	7
Computation times [s]	0.36	5.56	35.40

A. Example Scenario S_A

Scenario S_A is a deliberately simple scenario to illustrate the approach, where the ego vehicle follows a straight lane blocked by a static obstacle. The ego vehicle starts with a velocity of $v_0 = 20\frac{m}{s}$. Since we do not change the position of the ego vehicle due to a possible position invariance as mentioned in Sec. IV, it only remains to optimize the velocity. After only one iteration of quadratic programming and the subsequent binary search (see Alg. 2), the initial velocity of $20\frac{m}{s}$ is increased to $44.8\frac{m}{s}$, to the point where the scenario is almost impossible to complete without accident.

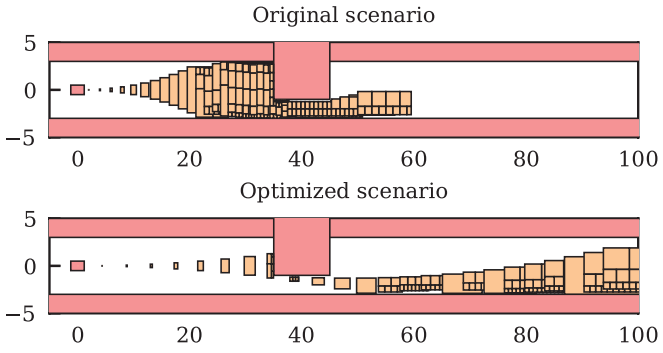


Fig. 5. Scenario S_A before and after the optimization.

B. Example Scenario S_B

The next scenario S_B is on an urban road in Munich with two lanes, with traffic facing in the same direction (ID=GER_Muc_2 of [36]). The speed limit is $18\frac{m}{s}$. The ego vehicle starts on the left lane with a velocity of $14\frac{m}{s}$, and the other four traffic participants are on the right or behind the ego vehicle, driving at 13-14$\frac{m}{s}$. Directly in front of the ego vehicle is a parked car, modeled by a static obstacle.

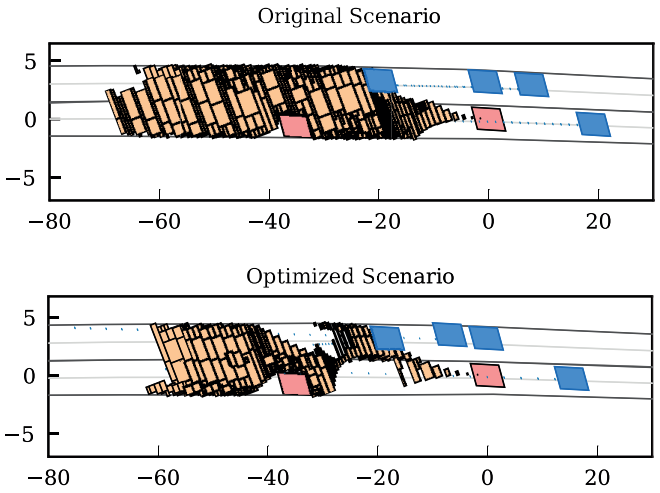
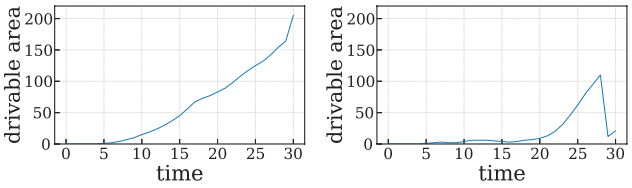


Fig. 6. Scenario S_B before and after the optimization. For clarity, we do not plot the drivable area of other traffic participants.

The velocities are changed to $13.3\frac{m}{s}$, $14.0\frac{m}{s}$ and $12.5\frac{m}{s}$ from front to rear on the right lane, and $15.4\frac{m}{s}$ for the vehicle on the left lane. Also, the other traffic participants on the right lane were pushed closer together, in an attempt to wall off the right lane. However, the ego vehicle still finds a spot to slip in and pass the obstacle. Because of the higher velocity of $17.9\frac{m}{s}$ compared to the vehicle ahead, another evasive maneuver directly after the static obstacle is required, making the scenario very dangerous as shown in Fig. 7.



(a) Original drivable area. (b) Optimized drivable area.

Fig. 7. Drivable area of scenario S_B before and after the optimization.

C. Example Scenario S_C

Scenario S_C is a rural road with two lanes, one for each direction (ID=GER_B471_1a). The speed limit is $28\frac{m}{s}$. Ahead of the ego-vehicle is a long obstacle, simulating a construction. It is blocking the whole lane, effectively constricting the road to only one lane for about 60 meters. On the other lane, two vehicles are coming towards the ego vehicle: First, a very broad truck moving at $8.3\frac{m}{s}$, followed by a normal car with $28\frac{m}{s}$. The ego-vehicle starts at $18\frac{m}{s}$.

The velocity of the ego-vehicle is increased to $27.5\frac{m}{s}$. The starting position of the truck is moved into the construction area, and its velocity is adjusted to $13.5\frac{m}{s}$. The car in the back is pushed back even further, while changing its velocity to $14\frac{m}{s}$. The resulting scenario can be described as follows: The ego vehicle arrives at the construction zone, but has

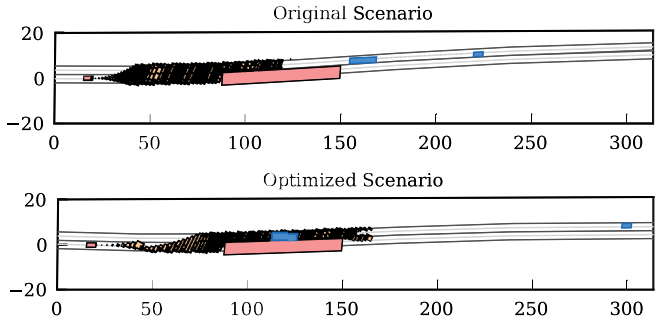
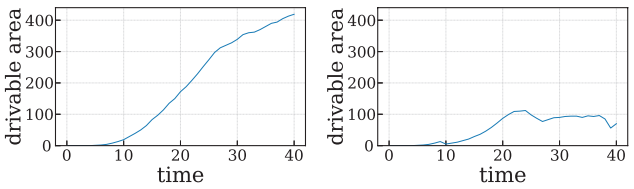


Fig. 8. Scenario S_C before and after the optimization. For clarity, we do not plot the drivable area of other traffic participants.

to conduct an emergency break to let the oncoming truck pass (situation at the top of Fig. 10). Once the way is clear, it speeds up to pass through that area (situation in center of Fig. 10) before the second car arrives, barely missing it (situation at the bottom of Fig. 10).



(a) Original drivable area. (b) Optimized drivable area.

Fig. 9. Drivable area of scenario S_C before and after the optimization.

VI. Conclusions

We have presented an approach to automatically alter traffic situations so that they become more critical in the sense that the solution space of the ego vehicle is reduced. Our approach is the first that can create situations optimized with respect to the drivable area of the ego vehicle, making it possible to realize a desired criticality. Since the presented approach does not require any user interaction, it can be used to adjust large sets of traffic situations. While we do not guarantee that we find the most critical situation, most returned situations are significantly more critical with respect to the solution space. Although the computation of critical scenarios consumes computation time, our experiments have shown that it only takes a few seconds to obtain a critical situation. This would otherwise require several thousands of kilometers to be driven in a simulator.

Acknowledgements

We gratefully acknowledge partial financial support by the project *interACT* within the EU Horizon 2020 programme under grant agreement No 723395.

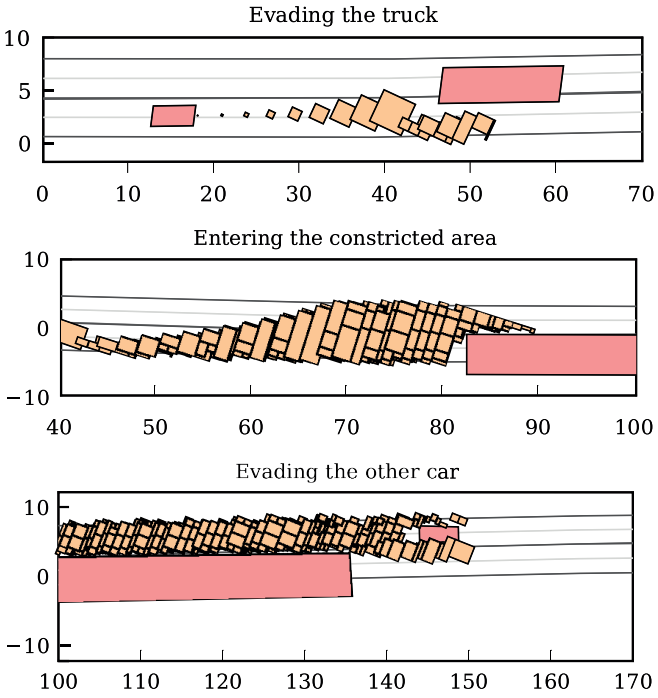


Fig. 10. Points of interest in Scenario S_C from top to bottom: Evading the truck, entering the constricted area, and evading the car.

References

[1] M. Broy, I. H. Krüger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[2] N. Kalra and S. M. Paddock, "How many miles of driving would it take to demonstrate autonomous vehicle reliability?" RAND Corporation, Santa Monica, CA, Tech. Rep., 2016. [Online]. Available: http://www.rand.org/pubs/research_reports/RR1478.html

[3] A. Broggi, M. Buzzoni, S. Debattisti, P. Grisleri, M. C. Laghi, P. Medici, and P. Versari, "Extensive tests of autonomous driving technologies," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1403–1415, 2013.

[4] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knöppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making Bertha drive – an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[5] A. Broggi, P. Cerri, S. Debattisti, M. C. Laghi, P. Medici, D. Molinari, M. Panciroli, and A. Prioletti, "PROUD—public road urban driverless-car test," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3508 – 3519, 2015.

[6] G. Özbilgin, A. Kurt, and U. Özguner, "Using scaled down testing to improve full scale intelligent transportation," in *Proc. of IEEE Intelligent Vehicles Symposium*, 2014, pp. 655–660.

[7] H. Diab, I. B. Makhlouf, and S. Kowalewski, "A platoon of vehicles approaching an intersection: A testing platform for safe intersections," in *Proc. of the 15th International IEEE Conference on Intelligent Transportation Systems*, 2012, pp. 1918–1923.

[8] B. Kim, Y. Kashiba, S. Dai, and S. Shiraishi, "Testing autonomous vehicle software in the virtual prototyping environment," *IEEE Embedded Systems Letters*, vol. 9, no. 1, pp. 5–8, 2017.

[9] M. R. Zofka, S. Klemm, F. Kuhnt, T. Schamm, and J. M. Zöllner, "Testing and validating high level components for automated driving: Simulation framework for traffic scenarios," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2016, pp. 144–150.

[10] R. Math, A. Mahr, M. M. Moniri, and C. Müller, "OpenDS: A new open-source driving simulator for research," in *Proc. of Automotive meets Electronics*, 2013.

[11] A. Rizaldi and M. Althoff, "Formalising traffic rules for accountability of autonomous vehicles," in *Proc. of the 18th IEEE International*

Conference on Intelligent Transportation Systems, 2015, pp. 1658–1665.

[12] M. O'Kelly, H. Abbas, S. Gao, S. Shiraishi, and S. Kato, "APEX: Autonomous vehicle plan verification and execution," in *Proc. of SAE World Congress*, 2016, pp. 1–12.

[13] M. OKelly, H. Abbas, and R. Mangharam, "Computer-aided design for safe autonomous vehicles," in *Proc. of Resilience Week*, 2017, pp. 90–96.

[14] 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.

[15] H. Weisser, P. J. Schulenberg, H. Gollinger, and T. Michler, "Autonomous driving on vehicle test tracks: overview, implementation and vehicle diagnosis," in *Proc. of IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, 1999, pp. 62–67.

[16] M. Quinlan, T.-C. Au, J. Zhu, N. Stiurca, and P. Stone, "Bringing simulation to life: A mixed reality autonomous intersection," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 6083–6088.

[17] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., *Model-based Testing of Reactive Systems: Advanced Lectures*. Springer, 2005.

[18] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proc. of the 21st International Conference on Software Engineering*, 1999, pp. 285–294.

[19] J. Tretmans, *Formal Methods and Testing*. Springer, 2008, ch. Model Based Testing with Labelled Transition Systems, pp. 1–38.

[20] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[21] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou, *Testing Real-Time Systems Using UPPAAL*. Springer, 2008, ch. Formal Methods and Testing, pp. 77–117.

[22] A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas, "Robust test generation and coverage for hybrid systems." in *Hybrid Systems: Computation and Control*, ser. LNCS 4416. Springer, 2007, pp. 329–342.

[23] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivančić, A.Gupta, and G. J. Pappas, "Monte-Carlo techniques for falsification of temporal properties of non-linear hybrid systems," in *Hybrid Systems: Computation and Control*, 2010, pp. 211–220.

[24] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *Proc. of the 36th Annual Conference of IEEE Industrial Electronics*, 2010, pp. 91–96.

[25] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, 2011, pp. 254–257.

[26] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivačić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 1–30, 2013.

[27] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles," in *Proc. of the IEEE 19th International Conference on Intelligent Transportation Systems*, 2016, pp. 1470–1475.

[28] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, 2016.

[29] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2017, pp. 1443–1450.

[30] V. de Oliveira Neves, M. E. Delamaro, and P. C. Masiero, "An environment to support structural testing of autonomous vehicles," in *Proc. of the Brazilian Symposium on Computing Systems Engineering*, 2014, pp. 19–24.

[31] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, 2008.

[32] M. OBrien, R. C. Arkin, D. Harrington, D. Lyons, and S. Jiang, "Automatic verification of autonomous robot missions," in *Proc. of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2014, pp. 462–473.

[33] G. Tagne, R. Talj, and A. Charara, "Design and comparison of robust nonlinear controllers for the lateral dynamics of intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 3, pp. 796–809, 2016.

[34] D. Heß, M. Althoff, and T. Sattel, "Comparison of trajectory tracking controllers for emergency situations," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2013, pp. 163–170.

[35] D. Calzolari, B. Schürmann, and M. Althoff, "Comparison of trajectory tracking controllers for autonomous vehicles," in *Proc. of the 20th IEEE International Conference on Intelligent Transportation Systems*, 2017.

[36] 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.

[37] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Dissertation, Technische Universität München, 2010, http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20100715-963752-1-4.

[38] A. Geiger, M. Lauer, F. Moosmann, B. Ranft, H. Rapp, C. Stiller, and J. Ziegler, "Team AnnieWAY's entry to the 2011 Grand Cooperative Driving Challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1008–1017, 2012.

[39] S. Manzinger, M. Leibold, and M. Althoff, "Driving strategy selection for cooperative vehicles using maneuver templates," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 647–654.

[40] M. Koschi and M. Althoff, "SPOT: A tool for set-based prediction of traffic participants," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 1686–1693.

[41] S. Söntges and M. Althoff, "Computing possible driving corridors for automated vehicles," in *Prof. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 160–166.

[42] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[43] D. E. Knuth, "Optimum binary search trees," *Acta Informatica*, vol. 1, pp. 14–25, 1971.

[44] 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.

[45] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proc. of the European Control Conference*, 2013, pp. 3071–3076.