2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

# Evaluating Location Compliance Approaches for Automated Road Vehicles

Alexander Zhu, Stefanie Manzinger, and Matthias Althoff

*Abstract*— **This work presents techniques for efficiently checking location compliance of automated road vehicles. We refer to location compliance as an allowed translational and rotational positioning of a vehicle on a road network, i.e., the vehicle does not enter forbidden lanes or regions reserved for other traffic participants, such as bike lanes or dedicated bus routes. Previous work has mostly focused on efficient collision detection between traffic participants and static obstacles represented as bounded sets. We formulate location compliance as a set enclosure problem, which cannot be solved directly with collision detection; thus, different algorithms from computational geometry have to be applied. We present polygon enclosure and boundary mesh generation approaches and evaluate them using existing road geometries from the CommonRoad database. For a fair comparison, we generate thousands of random instances which are evaluated statistically.**

## I. Introduction

One of the most important constraints of autonomous vehicles is to stay in certain lanes and avoid unsafe regions. In emergency situations, regions such as open spaces and bike lanes, which are typically not allowed to be entered, can suddenly be utilized to avoid a collision. We refer to checking whether a vehicle is allowed to drive in a certain region as *location compliance*. This includes determining enclosure in a certain region or the entering of forbidden regions. While many approaches exist for checking collisions between traffic participants and static obstacles (see e.g., [1], [2]), both of which can be represented by bounded sets, very little work exists on efficient and exact algorithms for checking location compliance. This often involves determining whether unbounded areas are entered—the different nature of unbounded sets requires different computational geometry algorithms. Checking whether the vehicle under consideration (the ego vehicle) is location compliant has to occur at high frequency for all vehicle poses in each motion plan, typically with more than 10Hz. As a result, the performance of location compliance largely determines the performance of motion planners. Subsequently, we detail how this work is related to the literature on motion planning and lane departure detection.

Motion planning can be roughly categorized in graph search techniques and continuous optimization techniques [3]. All graph search techniques (e.g., rapidly exploring random trees [4], probabilistic roadmaps [5], or motion primitives [6], [7] in combination with A* [8], Anytime A* [9], ARA* [10], D* [11]) require collision checks. Concerning optimization techniques, one can consider road boundaries as constraints: For fixed Cartesian coordinate systems, those constraints are non-convex and hard to handle. For this reason, almost all publications are planning in lane-based coordinate systems and typically assume constant road width, although this is a strong simplification, see e.g., [12]–[15]. No matter what motion planning technique is used, in reality, the executed motion differs from the plan due to disturbances, sensor noise, and unmodeled dynamics. Consequently, verification techniques are necessary, which typically require collision checking with road boundaries [16], [17].

Although the survey paper [3] and all previously mentioned works on motion planning of road vehicles state the importance of location compliance, no work explicitly provides an efficient computational solution for non-discretized road boundaries. However, many works exist for discretized road boundaries, e.g., represented by occupancy maps [1], [18]. As discussed later, using box-shaped occupancy grids results in unnecessary inaccuracy and thus possibly unsound verification results. In [19], [20], the occupancy map is referred to as a *drivability map* and also suffers from the previously mentioned resolution issue. In [21], polygons instead of an occupancy map are used to formulate constraints for their continuous trajectory optimization. Generally, one can say that most works do not precisely check if road or lane boundaries are met. We have not found a single paper in the Darpa Urban Challenge book [22] or any reference in the overview paper [3], that describes beyond occupancy maps how to best check whether the lane/road boundary has been left when following a planned trajectory. Additionally, works with a stronger safety focus do not describe how the departure of lane/road bounds is detected [23], [24].

Please note that this work intentionally does not consider the problem of detecting lane and road bounds from on-board sensors [25]–[27] or fusing this information with map data [28], but assumes that the maps are already provided. To account for missing or outdated map data, an autonomous vehicle should additionally be able to perform instantaneous detection or short-term prediction of lane/road departure from sensor data as presented in [29], [30]. These do not require map data, but only work for short-term predictions and do not perform exact intersections for verifying location compliance [31], [32].

In this work, we present efficient methods for location compliance in a systematical way. Sec. II provides an overview of our developed approaches. The polygon

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

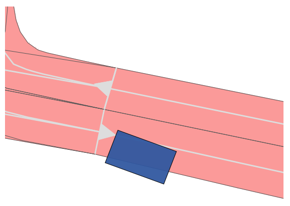
enclosure approach is presented in Sec. III and different boundary mesh approaches are introduced in Sec. IV. All developed approaches are evaluated in Sec. V using existing road geometries, followed by conclusions in Sec. VI.

## II. Problem Statement

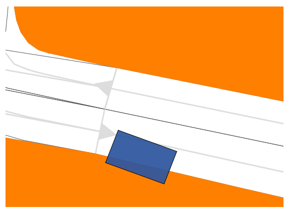
Let us first introduce some sets before formalizing the problem of location compliance. We denote the occupancy of the ego vehicle on a two-dimensional surface as $\mathcal{E} \subset \mathbb{R}^2$. As mentioned before, we assume that fused maps (maps corrected by sensor information) are already provided. We also assume that the allowed region for a vehicle, denoted by $\mathcal{A} \subset \mathbb{R}^2$, within the map, is provided as well. These regions typically originate from combining the given map information with traffic rules [33]. Checking location compliance can be done either by determining the enclosure of $\mathcal{E}$ in the allowed region, or by checking intersection with the compliment $\mathcal{A}^C$ of the allowed region:

$$\mathcal{E} \subseteq \mathcal{A} \Leftrightarrow \mathcal{E} \cap \mathcal{A}^C = \emptyset. \tag{1}$$

Both options are visualized in Fig. 1.



(a) Enclosure checking: Is the blue vehicle enclosed in the red allowed region? ($\mathcal{E} \subseteq \mathcal{A}$?)



(b) Boundary generation: Does the blue vehicle collide with the orange forbidden region? ($\mathcal{E} \cap \mathcal{A}^C = \emptyset$?)

Fig. 1: Visualization of enclosure checking and boundary generation approaches. The vehicle is depicted as a blue rectangle.

Determining $\mathcal{E} \subseteq \mathcal{A}$ is often realized with significantly different algorithms than collision detection $\bigcap_i \mathcal{O}_i = \emptyset$ between objects $\mathcal{O}_i \subset \mathbb{R}^2$ [34], [35]. In Sec. III we solve location compliance by checking $\mathcal{E} \subseteq \mathcal{A}$, which we refer to as the *polygon enclosure approach*. For instance, this can be done by checking whether the vehicle intersects with the road border. In Sec. IV we compute whether $\mathcal{E} \cap \mathcal{A}^C = \emptyset$, which we refer to as the *boundary mesh approach*. This approach uses collision detection between objects, but it requires the construction of the complement region $\mathcal{A}^C$.

In order to represent the problem in (1) by a finite representation for computational analysis, we express the ego vehicle as a polygon and the road network with lanelets, which are essentially polygons with a left and right border, to determine the driving direction.

**Definition 1 (Lanelet [36]):** A lanelet is defined by its *left* and *right bound*, where each bound is represented by an array of points (a polyline), as shown in Fig. 2. □

Due to the use of polygons, we can represent arbitrary shapes with arbitrary precision [37]. In reality, road and vehicle information might not be exact, so that a vehicle could potentially leave the allowed area without it being detected. This can be prevented by appropriately enlarging $\mathcal{E}$ or shrinking $\mathcal{A}$ as a safety margin for inaccurate map data.

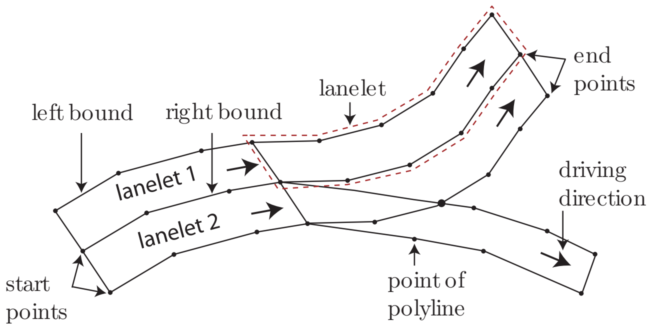


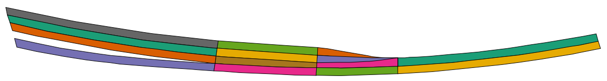
Fig. 2: Road network composed of lanelets.

The main objective of this work is to present the enclosure and the boundary mesh approaches and to evaluate their efficiency for checking the location compliance of road vehicles. Results of the evaluation are presented in Sec. V. Please note that our work focuses on the location compliance in relatively small maps. When operating on large road networks, it might be required that the map is divided into several local regions. During motion planning, different regions have to be loaded due to the motion of the ego vehicle. However, we do not analyze the loading times of map regions, because typically only a few loading operations are required, compared to thousands of location compliance checks, which thus dominate the computation time.

## III. Polygon enclosure approach

The polygon enclosure technique checks whether the ego vehicle $\mathcal{E}$ is enclosed by the allowed region $\mathcal{A}$, which is described by lanelets. Let us abstract away the driving direction of each lanelet and model the $i^{\text{th}}$ lanelet as a polygon $\mathcal{L}_i$. The task now is to check whether $\mathcal{E} \subseteq \bigcup_i \mathcal{L}_i$. Note that $\mathcal{E}$ might intersect with multiple lanelets, so it is not enough to check for $\exists i : \mathcal{E} \subseteq \mathcal{L}_i$. There are different considerations for how this can be computed efficiently. Here, we discuss the three following aspects: The choice of a basis from which the polygons are constructed, the selection method for polygons, and the computational method to check for inclusion.

*a) Polygon basis:* As mentioned before, $\mathcal{A}$ is described by lanelets, and these can be modeled as polygons. However, for a more efficient algorithm, we can also unify laterally adjacent lanelets into a single polygon. We call a set of laterally adjacent lanelets *lane sections*. In Fig. 3 on the following page, the lanelet and lane section representation are compared. The motivation for utilizing lane sections is that two laterally adjacent lanelets share a polyline between them, but this line is not present in the lane section polygon. This means that the total number of points can be reduced with this representation, which might lead to a more efficient algorithm. In the following, we denote the individual polygons as $\mathcal{L}_i$, regardless of their basis.


(a) Lanelets.

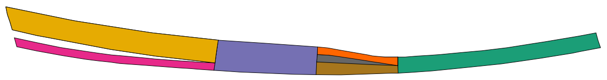

(b) Lane sections.

Fig. 3: Comparison of different polygon representations.

*b) Selection criteria:* Only some of the $\mathcal{L}_i$ potentially intersect with $\mathcal{E}$. For this reason, we observe different strategies how to select the relevant polygons:

- *Map region:* The union of all $\mathcal{L}_i$ of a map region $\mathcal{M} \supset \mathcal{E}$ is constructed. We get a single polygon $\mathcal{L}_\cup = \bigcup_i \mathcal{L}_i$ which represents $\mathcal{A}$. The benefit of this strategy is that this union can be precomputed for a certain $\mathcal{M}$. However, $\mathcal{L}_\cup$ can have as many points as the sum of the points of all $\mathcal{L}_i$, which might lead to an inefficient inclusion test.
- *Box selection:* Potentially intersecting $\mathcal{L}_i$ in $\mathcal{M}$ are identified by intersecting their axis-aligned bounding box with the bounding box of $\mathcal{E}$. This can be computed in constant time and provides a set of candidate polygons $\mathcal{L}_i^d$. This set is often much smaller than the set of all $\mathcal{L}_i$, and it is guaranteed to contain all polygons that intersect with $\mathcal{E}$. However, each $\mathcal{L}_i^d$ and their union cannot be precomputed as the selection depends on $\mathcal{E}$.

*c) Computational method:* Depending on the selection strategy, we operate on a single polygon $\mathcal{L}_\cup$ or on a set of candidate polygons $\mathcal{L}_i^d$. We can reduce all $\mathcal{L}_i^d$ to a single polygon by computing their union $\bigcup_i \mathcal{L}_i^d$. For all following discussions on time complexity in this paper, we assume that each involved polygon has N line segments. Computing the union of two general polygons has time complexity $\mathcal{O}(N^2)$ and well-known algorithms exist [38]. In particular, for union operations we use the General Polygon Clipper (GPC) library [39], which implements the algorithms by Vatti [40]. GPC can handle non-convex, self-intersecting polygons, and polygons with holes, so we can perform the union operation for arbitrary road shapes.

Now, with a single polygon $\mathcal{L} \in \{\mathcal{L}_\cup, \bigcup_i \mathcal{L}_i^d\}$, computing $\mathcal{E} \subseteq \mathcal{L}$ can be done in the following ways:

- *Segment intersection:* The enclosure test of two polygons can be performed in $\mathcal{O}(N \log N)$ time [41, Theorem 5] by checking whether any of their line segments intersect. If they do, $\mathcal{E} \nsubseteq \mathcal{L}$. Otherwise, if their line segements do not intersect, $\mathcal{E} \subseteq \mathcal{L}$ iff any point of $\mathcal{E}$ is located in $\mathcal{L}$.
- *Difference:* As polygon clipping operations are already implemented for computing the union, we also examine using polygon difference:

$$\mathcal{E} \subseteq \mathcal{L} \iff \mathcal{E} \setminus \mathcal{L} = \emptyset, \quad (2)$$

where the difference operation is a clipping operation, and therefore has a complexity of $\mathcal{O}(N^2)$, and checking for an empty polygon takes constant time. While this enclosure test has a higher worst-case bound than segment intersection, both methods have an overall complexity of $\mathcal{O}(N^2)$ if we operate on a polygon set, as this requires us to compute its union.
- *Iterative difference:* If we operate on the polygons $\mathcal{L}_i^d$, instead of computing their union, we can also perform iterative differences:

$$\mathcal{E}_{remain,i+1} = \mathcal{E}_{remain,i} \setminus \mathcal{L}_i^d, \quad (3)$$

where $\mathcal{E}_{remain,0} = \mathcal{E}$. The benefit of this method is that at each step $i$, we can check whether $\mathcal{E}_{remain,i+1} = \emptyset$. If this is the case, it can be concluded that $\mathcal{E} \subseteq \mathcal{A}$, without necessarily iterating over all $\mathcal{L}_i^d$.

## IV. Boundary Mesh Approaches

In contrast to the polygon enclosure approach, the boundary mesh approach checks whether $\mathcal{E} \cap \mathcal{A}^C = \emptyset$; see (1). Computing whether two polygons intersect can be done in $\mathcal{O}(N \log N)$ time [41, Theorem 5]. It remains to compute $\mathcal{A}^C$ whose representation directly affects the computation times. Computing $\mathcal{A}^C$ can however be done preemptively for a map region $\mathcal{M}$. With this, we can determine the location compliance in $\mathcal{O}(N \log N)$ compared to $\mathcal{O}(N^2)$ for polygon enclosure of two polygons. Nonetheless, the average number of polygons in $\mathcal{A}^C$ to be checked might be larger than the average number of polygons for the enclosure approach. Also, the average computation time for operations involving two polygons might be different from the worst-case time complexity. Both aspects are evaluated in Sec. V.

The goal for generating a boundary mesh that models $\mathcal{A}^C$ is twofold: First, it should be composed from simple objects for which efficient collision checks exist, and second, it should be composed of as few objects as possible. In the following, we give an overview of three investigated approaches.

### A. Quadtree

Quadtrees (see e.g., Ericson [42]) can be used to fill $\mathcal{A}^C$ with axis-aligned rectangles. In short, a quadtree recursively splits a rectangle into four equal parts if it intersects with $\mathcal{A}$. With this procedure, after several iterations a rather tight boundary mesh is obtained, as shown in Fig. 4 on the next page. The method can fill large spaces with only a few boxes, while a large number of boxes are constructed close to the edge of the road. While there are small gaps between the mesh and the road (since the road is generally not axis-aligned), these can be reduced by increasing the maximum depth, but at the cost of creating more boxes.

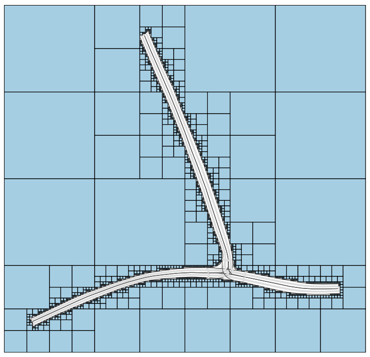


Fig. 4: The boundary mesh (in blue) generated by the basic quadtree approach. Scenario map: GER_Ffb_2 from [43].

### B. Shell

The main disadvantage of the quadtree approach is that many boxes are required at the border of the road to fill non-axis-aligned regions with axis-aligned boxes. This disadvantage is addressed by our so-called *shell approach*. The method constructs boundary rectangles along the polylines of the road border. The rectangles have a user-defined width $w$. To handle areas between lanelets where a rectangle is too wide to fit and it intersects with $\mathcal{A}$, a recursive subdivision algorithm is used, just like for quadtrees. The result of the shell approach is shown in Fig. 5.

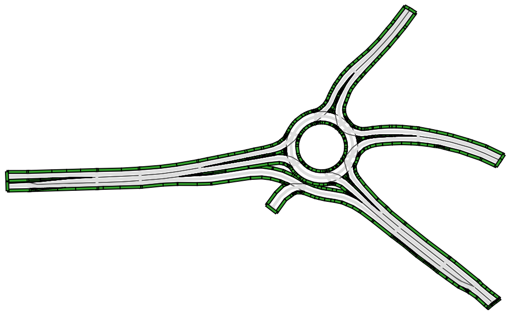
In contrast to the quadtree approach, $\mathcal{A}^C$ is not filled completely; only the immediate area around $\mathcal{A}$ contains rectangles. When the ego vehicle occupancy $\mathcal{E}$ is computed for consecutive time intervals, as e.g., presented in [16], crossing the shell is detected irrespectively of the width $w$ (a minimum distance regarding floating point errors is required, of course). When collision checks are performed at discrete points in time with a time increment $\Delta t$, the width should be chosen as $w \geq v_{\mathtt{max}}\Delta t$, where $v_{\mathtt{max}}$ is the maximum velocity of the ego vehicle. To keep this overview concise, we present the implementation of this newly developed approach in the Appendix.

### C. Triangulation

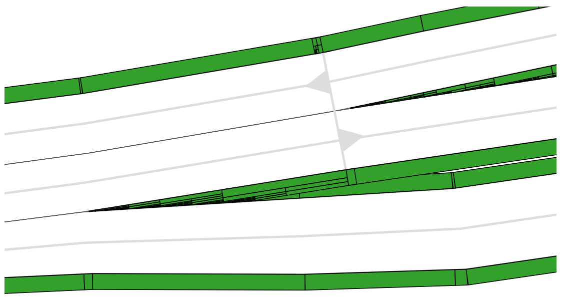
Finally, triangulation is proposed, which is an exact method to create the boundary mesh. We perform a Delaunay Triangulation between the road border and the border of the map region $\mathcal{M}$. Our implementation uses the *Triangle* library by Shewchuk [44]. Fig. 6 on the next page shows the triangulation of an example map.

Some maps contain polygons with excessively many points potentially creating many thin triangles; this has two disadvantages: a) increased computational load for location compliance testing, and b) potentially wrong computations due to floating point rounding errors. We are using the following measures to counteract this issue:

- We resample the lanelet polylines and remove unnecessary points using the Douglas-Peucker point reduction algorithm [45].



(a) Full view.



(b) Detailed view.

Fig. 5: An example of the shell approach with rectangles (dark green) surrounding the road. Scenario map: GER_B471.

- We apply a constrained Delaunay triangulation, where a minimal angle for each triangle is enforced [46].

All measures are applied during the construction of the mesh and thus do not influence the online computation times. The effects of the countermeasures are presented in Fig. 7 on the following page.

## V. Evaluation

In this section we evaluate all presented approaches. We use three road networks from the CommonRoad database [43]: GER_Ffb_2, GER_Muc_1a, NGSIM_US_101, and an additional map GER_B471 that we have created for this work. All computations are performed on a Core i7 2600K CPU with 8 GB of DDR3-RAM. To account for fluctuations in computational times, each computation was repeated 20 times, and the minimum time of each run[1] is presented. Each evaluation is performed by sampling 10,000 random poses of vehicles with uniformly distributed position and orientation. Further details are presented in the Appendix.

In this measurement, we only focus on the runtime of the location compliance check. The memory requirements

[1]as recommended in the documentation for time measurements in Python, see: `https://docs.python.org/3/library/timeit.html#python-interface` (Accessed date 12.01.18)

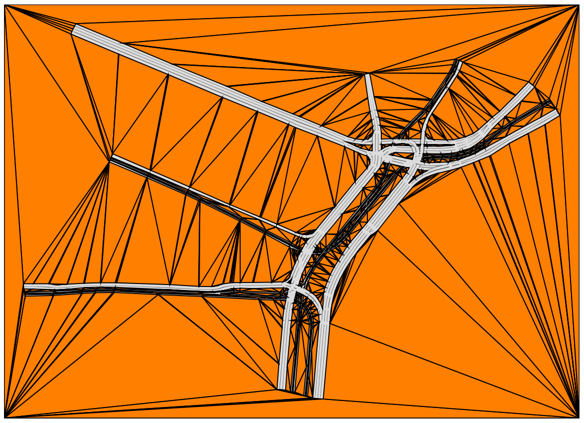
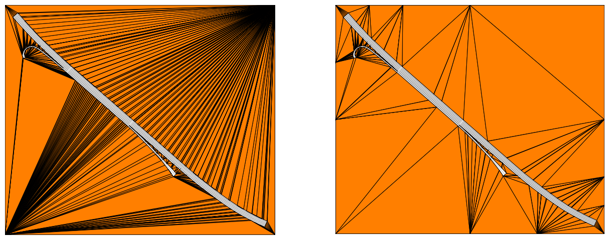


Fig. 6: Triangulation result for scenario map GER_Muc_1a from [43].



(a) Before countermeasures. (b) After countermeasures.

Fig. 7: Triangulation before and after countermeasures. Scenario map: NGSIM_US_101 from [43].

are negligible here because the maps only contain a few thousand vertices. However, for a large road network, where different maps have to be loaded dynamically, the memory consumption becomes a valid concern.

*a) Polygon enclosure approaches:* Tab. I on the next page shows the time measurements for the enclosure approach. We can see that the box selection is more efficient than operating on the union of all lanelets or lane sections in the map. This is because the polygon that represents the entire map has too many vertices, and therefore the difference operation with $\mathcal{O}(N^2)$ for $N$ vertices takes more time than for the smaller polygons.

Furthermore, we can see that for checking the polygon inclusion, the segment intersection performs far worse than the polygon difference. This is suprising, because as mentioned in Sec. III on page 2, the worst case runtime of segment intersection is in $\mathcal{O}(N \log N)$ compared to that of polygon clipping $\mathcal{O}(N^2)$. As the approach is non-competitive, we have not computed the segment intersection for the map selection strategy and for the lane section basis.

When comparing the lanelet and lane section approaches, the lane sections tend to be faster, but the relative difference between them varies between maps. For instance, the lane difference is 4.3 times faster than the lanelets difference for NGSIM_US_101, but only 1.1 times faster for GER_Muc_1a. This is because the maps have different numbers of adjacent lanelets. If more of the lanelets are neighboring, the lane sections are larger. The lane polygons have only slightly more vertices than a single lanelet, as they do not contain the edges between the neighbors. Hence, the lane section representation leads to fewer polygons, and overall to fewer vertices that have to be computed.

Finally, we compare the difference and the iterative difference approaches. Computing the iterative difference tends to be faster for lanelets and for lane sections. This makes sense, as for the iterative differences, the remaining polygon is checked for emptiness after each difference operation. This means that this approach can terminate preemptively. Additionally, the iterative method requires one less clipping operation, as the other approach first builds the union of polygons and then performs an inclusion check by set difference. However, this does not always seem to be more efficient; for the map GER_Ffb_2, the non-iterative algorithm is slightly faster. Overall, the measurement shows that computing the iterative differences for lane sections is the most efficient method for the polygon enclosure approach.

*b) Boundary mesh approaches:* Tab. II on the next page shows the evaluation for the runtime and mesh size of the boundary collision approaches. The quadtree is clearly non-competitive, because of the large number of rectangles that it generates. The triangulation tends to be faster than the other methods, mostly because the resulting meshes contain fewer collision objects. Another reason is that collision detection performs better with triangles than with oriented rectangles, which we can see in the case of the map GER_Ffb_2, where the number of objects is almost equal between the shell and triangulation methods, but the latter is much faster. However, there is an outlier with NGSIM_US_101, where the shell is slighly faster than the triangulation, because it generates roughly half as many objects. Nonetheless, the triangulation seems to be the best boundary approach in most cases, particularly because it represents an exact solution, while the quadtree and shell are approximations. We chose a high recursion depth for the quadtree and shell approaches so that their approximation errors are relatively low. Instead of using them as a stand-alone method for testing the location compliance, it might be beneficial to apply them with a low depth value for fast, preemptive checks.

*c) Overall assessment:* When we compare the best results of the polygon and boundary approaches, the iterative lane difference approach is slightly faster than the triangulation in all cases. However, the measurements are highly dependent on the implementation of the methods. An area for future improvement is to ensure that the frameworks for polygon enclosure and collision detection are optimized.

Overall, in the current implementation, the polygon and boundary approaches have a comparable runtime. This suggests that both are similarly efficient methods to check for location compliance.

To better understand the performance of our approaches, it is necessary to compare them to collision detection on occupancy grids, which to date is the primarily-used method for checking location compliance. While we have not evalu-

TABLE I: Polygon enclosure runtime measurement. Time measured in seconds.

| **Polygon basis** | **Selection criteria** | **Computational method** | **GER B471** | **GER Ffb 2** | **GER Muc 1a** | **NGSIM US 101** |
|---|---|---|---|---|---|---|
| Lanelets | Map | Difference | 0.6901 | 0.2929 | 6.4191 | 0.6881 |
| | Box | Difference | 0.327 | 0.1748 | 1.696 | 0.6159 |
| | | Segment Intersection | 6.1982 | 3.1636 | 21.9658 | 5.6783 |
| | | Iterative Difference | 0.2905 | 0.1451 | 1.0426 | 0.362 |
| Lane sections | Map | Difference | 0.6928 | 0.2959 | 6.4199 | 0.6875 |
| | Box | Difference | 0.2568 | **0.0787** | 1.5852 | 0.1423 |
| | | Iterative Difference | **0.2275** | 0.0803 | **0.7317** | **0.1084** |

TABLE II: Boundary approach measurement. The number of objects (nr) in the boundary meshes are listed and the runtime is measured in seconds.

| | **Quadtree** | | **Shell** | | **Triangulation** | |
|---|---|---|---|---|---|---|
| | time | nr | time | nr | time | nr |
| **GER B471** | 1.8049 | 7098 | 1.2352 | 3040 | **0.3143** | **2213** |
| **GER Ffb 2** | 0.7515 | 4021 | 0.2947 | 685 | **0.0974** | **653** |
| **GER Muc 1a** | 4.519 | 16813 | 5.3756 | 12893 | **0.8544** | **6457** |
| **NGSIM US 101** | 1.1346 | 4705 | **0.1927** | **431** | 0.2297 | 916 |

ated occupancy maps directly, a quadtree is essentially a non uniform grid that contains discrete occupancy information. The advantage of our proposed algorithms is that there is no inherent inaccuracy in the obstacle representation. In a grid, a large amount of cells is necessary to be able to describe road boundaries with a precision of a few centimeters. Therefore, our approaches are better suited if this level of accuracy is required. The disadvantage of our approaches is that they are currently not as efficient as state-of-the-art occupancy grid methods, since collision detection within a uniform grid can be reduced to querying distinct cells in the grid [1]. However, our results show that much more compact representations can be used for storing obstacle information compared to a grid, such as a triangle mesh or polygon representation and we also show that checking location compliance can be performed relatively efficient with them. It remains to optimize our collision detection and polygon intersection tests, with the goal that our approaches reach a similar performance as occupancy grids.

## VI. Conclusions

To conclude, we propose multiple methods to check for location compliance of vehicles with an arbitrary polygon shape. The algorithms operate on a map region $\mathcal{M}$, which is part of a road network that is represented by lanelets. Our approaches include both approximate and exact methods, but from our results, we strongly encourage the use of exact location compliance detection.

The polygon enclosure approach determines whether the vehicle occupancy $\mathcal{E}$ is covered by the allowed area $\mathcal{A}$ ($\mathcal{E} \subseteq \mathcal{A}$). This is done with polygon clipping operations. In particular, our measurements suggest that the most efficient method is to iteratively compute the polygon difference with the polygons that represent $\mathcal{A}$.

On the other hand, the boundary mesh approach checks with collision detection whether $\mathcal{E} \cap \mathcal{A}^C = \emptyset$. We present quadtree, shell, and triangulation approaches to construct the mesh that represents the forbidden area $\mathcal{A}^C$. Our measurements show that Delaunay triangulation is the most promising approach for creating $\mathcal{A}^C$.

Both the polygon and boundary mesh approaches have comparable performance in our evaluation. Further work on comparing them should be done; in particular, their implementation should be extended so that both methods utilize state-of-the-art frameworks. Another area for improvement is to examine the performance on more and, in particular, larger maps.

## Acknowledgements

We gratefully acknowledge partial financial support by the Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program SPP 1835 Cooperative Interacting Automobiles (grant number: AL 1185/4-1) and the European Commission project interACT under grant number 723395.

## Appendix

*a) Shell approach:* The goal of the shell approach is to create a boundary mesh that tightly fits around $\mathcal{A}$. This is done by slightly offsetting the polylines that describe the road border and then creating rectangles along these lines. The resulting objects might intersect $\mathcal{A}$, and to solve these cases in a simple way, rectangles that intersect are split into four parts, like in a quadtree. We require three parameters for the algorithm:

1) the **width**, which determines how far an initial rectangle at depth 0 extends to the sides;
2) the **margin**, which adds a small space between the road boundary and the shell, as all rectangles would otherwise intersect with the road at their edge;
3) the **max_depth**, which specifies the maximum recursion depth for rectangle subdivision;

The implementation of the shell approach is shown in algorithm 1 on the following page. In line 3, the lane boundaries are offset. The distance is calculated so that the margin between the rectangles and the road is ensured. Then, in line 8, the recursive function BUILD_RECTANGLE is called for each two points of all the road border polylines. It is defined in the following line 11. The parameters of the function are two points $v_1$ and $v_2$, the width of its rectangle $local\_width$, and the recursion $depth$. First, in line 13, a rectangle is created, which has the line segment ($v_1$, $v_2$) as a central axis. If the rectangle does not collide with $\mathcal{A}$, it is added to the output. Otherwise, it is recursively split into four parts. However, the orientation has to be considered when calculating the coordinates for the four sub-rectangles. Summarizing line 18 to line 24: The line segment ($v_1$, $v_2$) is constructed, as well as the $normal$, which is perpendicular to the segment. Additionally, the $middle$ point of the segment is computed. With these vectors, four sub-rectangles can be calculated.

**Algorithm 1** Algorithm of the shell approach. The recursive function BUILD_RECTANGLE is defined and called in a loop.

```
1:  constant $width, margin, max\_depth$
2:  $offset \leftarrow margin + width/2$
3:  $bounds \leftarrow$ GENERATE_OFFSET_BOUNDS($offset$)
4:  $rectangles \leftarrow \{\}$
5:  $\mathcal{A} \leftarrow$ GENERATE_ROAD_REPRESENTATION( )

6:  for $bound$ in $bounds$ do
7:      for $v_1, v_2$ in $bound$ do
8:          BUILD_RECTANGLE($v_1, v_2, width, 0$)
9:      end for
10: end for

11: function BUILD_RECTANGLE($v_1, v_2, local\_width, depth$)

12:     $rectangle \leftarrow$ CREATE_RECTANGLE_ALONG_LINE
13:                 ($v_1, v_2, local\_width$)
14:     if not $rectangle$.COLLIDES_WITH($\mathcal{A}$) then
15:         $rectangles$.ADD($rectangle$)
16:     else
17:         if not $depth \geq max\_depth$ then
18:             $depth \leftarrow depth + 1$
19:             $local\_width \leftarrow local\_width/2$
20:             $line \leftarrow v_1 - v_2$
21:             $normal \leftarrow$ ROTATE($line, 90$)
22:             $normal \leftarrow$ NORMALIZED($normal$)
23:             $normal \leftarrow (local\_width/2) \cdot normal$
24:             $middle \leftarrow (line/2) + v_2$
25:             BUILD_RECTANGLE($v_1 + normal, middle + normal, local\_width, depth$)
26:             BUILD_RECTANGLE($middle + normal, v_2 + normal, local\_width, depth$)
27:             BUILD_RECTANGLE($v_1 - normal, middle - normal, local\_width, depth$)
28:             BUILD_RECTANGLE($middle - normal, v_2 - normal, local\_width, depth$)
29:         end if
30:     end if
31: end function
```

*b) Evaluation details:* Each of the 10,000 random vehicles has a width of 1.9 and a length of 5 meters, in order to roughly model a real world automobile. The polylines of all maps are resampled and reduced (with Douglas-Peucker, epsilon = 0.1) before the evaluation to make the results more consistent and to solve cases where adjacent lanelets would not share a polyline (this creates discrepancies between the lanelet and lane section polygons). Some of the evaluated algorithms require parameters; these are listed in the following:

- *Quadtree:* The recursion is limited with a maxium depth of 10.
- *Shell:* Width = 0.2, margin = 0.02, maximum depth = 5.
- *Triangulation:* Triangles are constrained to have minimal angles of 20 degrees, but the number of vertices may at most double to fullfill this.
- *Iterative Differences:* Unique to the polygon enclosure approaches, iterative differences introduce floating point inaccuracies. This is because the difference operation computes new points, which is not the case for polygon unions. As a result, if a vehicle intersects with multiple polygons, checking for the emptiness of the remaining polygon might not give the correct result. In practice, we set a small threshold of 1e-7 for the minimal area of the polygon.

## References

[1] 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.

[2] 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.

[3] 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.

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, 1999, pp. 473 – 479.

[5] 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.

[6] 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.

[7] D. Heß, M. Althoff, and T. Sattel, "Formal verification of maneuver automata for parameterized motion primitives," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1474–1481.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions of Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] E. A. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[10] M. Likhachev, G. Gordon, and S. Thrun, "ARA* : Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and P. B. Schölkopf, Eds. MIT Press, 2004, pp. 767–774.

[11] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652–1659.

[12] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2010, pp. 987–993.

[13] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *International Journal of Robotic Research*, vol. 31, no. 3, pp. 346–359, 2012.

[14] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2012, pp. 2061–2067.

[15] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 458–464.

[16] 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.

[17] 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 20th IEEE International Conference on Intelligent Transportation Systems*, 2017, pp. 1–8.

[18] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proc. of the IEEE International Conference on Robotics and Automation*, 1985, pp. 116–121.

[19] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[20] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, *A Perception-Driven Autonomous Urban Vehicle*. Springer, 2009, pp. 163–230.

[21] 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.

[22] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The DARPA Urban Challenge*. Springer, 2009.

[23] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proc. of the Conference on Intelligent Robots and Systems*, 2005, pp. 2210–2215.

[24] R. Parthasarathi and T. Fraichard, "An inevitable collision state-checker for a car-like vehicle," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2007, pp. 3068–3073.

[25] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications*, vol. 25, no. 3, pp. 727–745, 2014.

[26] J. Han, D. Kim, M. Lee, and M. Sunwoo, "Enhanced road boundary and obstacle detection using a downward-looking LIDAR sensor," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 3, pp. 971–985, 2012.

[27] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2010, pp. 1006–1013.

[28] M. Darms, M. Komar, and S. Lueke, "Map based road boundary estimation," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2010, pp. 609–614.

[29] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 20–37, 2006.

[30] D. J. LeBlanc, G. E. Johnson, P. J. T. Venhovens, G. Gerber, R. DeSonia, R. D. Ervin, C.-F. Lin, A. G. Ulsoy, and T. E. Pilutti, "CAPC: A road-departure prevention system," *IEEE Control Systems*, vol. 16, no. 6, pp. 61–71, 1996.

[31] P. Angkititrakul, R. Terashima, and T. Wakita, "On the use of stochastic driver behavior model in lane departure warning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 174–183, 2011.

[32] S. Mammar, S. Glaser, and M. Netto, "Time to line crossing for lane departure avoidance: A theoretical study and an experimental setting," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 2, pp. 226–241, 2006.

[33] 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.

[34] M. K. Agoston, *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer, 2005.

[35] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer graphics: principles and practice*. Addison-Wesley Professional, 2013.

[36] 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.

[37] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *Proc. of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, 1996, pp. 313–324.

[38] M. van Kreveld, "On fat partitioning, fat covering and the union size of polygons," *Computational Geometry*, vol. 9, no. 4, pp. 197–210, 1998.

[39] A. Murta, "A General Polygon Clipping Library," 2014.

[40] B. R. Vatti, "A generic solution to polygon clipping," *Commun. ACM*, vol. 35, no. 7, pp. 56–63, Jul. 1992.

[41] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *Proc. of the 17th Annual Symposium on Foundations of Computer Science*, 1976, pp. 208–215.

[42] C. Ericson, *Real-Time Collision Detection*. CRC Press, Inc., 2004.

[43] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.

[44] 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, vol. 1148, 1996, pp. 203–222.

[45] 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: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[46] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," in *Computational Geometry: Theory and Applications*, vol. 22, no. 1–3, 2002, pp. 21–74.