On Time-Memory Trade-Off for Collision Detection

Albert Rizaldi, Sebastian Söntges, and Matthias Althoff

***Abstract*— Future collision avoidance systems, which are capable of fully controlling the vehicle, have to make critical decisions in a very short time. To do this, they need to check constantly if their own vehicle's occupancy collides with the other traffic participants' occupancy. Those collision checks consume a substantial amount of time and consequently, the collision avoidance systems could fail to intervene in complex scenarios. We propose a new approach to reduce the computation time for collision checks significantly. Instead of using geometric methods, we store finitely many possible collision scenarios between two objects in a table and thus collision checks become a matter of lookup table queries. To ensure that the finite number of configurations cover all possible scenarios, we use a novel abstraction technique which guarantees that every collision will be detected. The approach works for arbitrarily many traffic participants by applying the approach pairwise (own vehicle and other object) to each traffic participant. Randomly generated scenarios show that the new approach can be several times faster than geometric intersection techniques thanks to the trade-off between memory consumption and computation time.**

I. Introduction

Collision avoidance systems in fully automated road vehicles are a key technology in a future without life-threatening collisions. Once collision avoidance systems gain the required reliability, we will also see a transformation from highly automated driving towards fully automated driving. A major task of collision avoidance systems is to verify whether the planned action is indeed collision-free. This is achieved by intersecting the predicted occupancy of the ego vehicle with the predicted occupancy of other traffic participants over time.

Geometric collision detection of occupancies consumes substantial computation time when searching for collision-free paths [1], [2]. A major influence on the efficiency of collision detection algorithms is the geometric representation of static obstacles and other traffic participants. Two major approaches exist. The first approach uses sensor data, which has only been processed to a small extent and is gathered in occupancy grids, see e.g. [3] for the two-dimensional case and [4] for the three-dimensional case. In order to improve access to grid-based occupancy information, tree-based data structures are often used [5]. The alternative approach is to fuse data from multiple sensors and represent occupancies in a *model-based* object representation [6]. Since our approach falls into the latter category, the remainder of the literature review focuses on this.

Albert Rizaldi, Sebastian Söntges, and Matthias Althoff are with Technische Universität München, Fakultät für Informatik, Lehrstuhl für Robotik und Echtzeitsysteme, Boltzmannstraße 3, 85748, Garching, Germany {rizaldi, soentges, althoff}@in.tum.de

One of the most common model-based representations of objects are polyhedrons, i.e. intersections of halfspaces [7]. Special cases are rectangles as applied in [8] or trapezoids as applied in [9]. Intersection detection between polyhedra can be efficiently implemented using the Separating Axis Theorem [10]. Since collision checking between two circles and between other model representation and circles are very efficient, other traffic participants have been represented by several circles in [11]. Circles can be generalised to ellipsoids, which are often used in robotics [12]. Besides single vehicles, it is of interest to represent occupancy regions by several rectangles or similar shapes [13]–[15]. This makes it possible to consider uncertainty in sensing and uncertainties in the prediction of traffic participants.

To improve the efficiency of collision detection algorithms, a popular approach is to use a hierarchy of representations from simple to complicated representations [16], [17]. For instance, one can enclose bounded polyhedra (i.e. polytopes) or other object representations by n-spheres. Only in the event that enclosing n-spheres intersect, which is computationally cheap to detect, the more elaborate collision detection involving the corresponding polyhedra is performed.

To overcome the huge computational requirements for collision detection, we present a completely different approach. Instead of computing possible intersections with classical intersection algorithms, we pre-compute whether a collision occurs for possible relative configurations (positions and orientations). The results of a finite number of relative configurations are stored in a lookup table. Since we need to guarantee that no collision occurs for all possible (infinitely many) configurations, we abstract the collision check such that the full configuration space is covered by the finite pre-computed set. Thus, the collision check is simplified to a simple database entry retrieval, significantly speeding up the collision checking. In addition, we store the depth of penetration of objects if they intersect as an additional feedback for the planner. This computation is usually not performed in classical approaches due to the high computational costs. In our approach however, this information comes for free since we only store the penetration, where a value of 0 indicates that there is no collision. In principle, our approach is independent of the geometric representation of objects. In this work, we demonstrate the approach for rectangles. Note that more complicated shapes can be composed from several rectangles. In the broader sense, our approach trades memory for time consumption [18].

The paper is organised as follows. We give an overview of our approach in Sec. II and detail it in Sec. III and IV. Section V explains how the proposed approach is used to

construct the lookup table and Sec. VII describes how the proposed approach is evaluated. We end this paper with the conclusion and future work in Sec. VIII.

II. Overview of the Approach

Suppose we have two rectangles denoted by R_{ego} and R_{other}. Rectangle R_{ego} has width W_{e} and length L_{e}, and W_{o} and L_{o} are defined similarly for R_{other}. A rectangle is specified by the triples (x, y, θ), where the pair (x, y) is the rectangle's centre in fixed Cartesian coordinates, and variable θ is the orientation of the rectangle (see Fig. 2). The collision detection problem considered is to determine whether these two rectangles collide or not.

The objective of this paper is to obtain a *fast* collision detection method by building a lookup table, which trades time consumption for memory consumption. It is not obvious how to apply time-memory trade-off in collision detection, since collision detection has a continuous input space (positions and orientations). Thus, infinitely many lookup table entries would be required to represent all possible positions and orientations of any two rectangles.

We tackle this issue as follows. Firstly, we *discretise* the input space to make the lookup table entries finite. Secondly, due to the finite entries, we *quantise* the position and orientation of the rectangle when we want to query the lookup table. Thirdly, we *enlarge* both rectangles when constructing the lookup table to compensate for the quantisation error.

We argue that due to the enlargement, the lookup table is *complete* i.e. it asserts a collision when in reality there is a collision. This enlargement also implies that false positives are now possible, i.e. the lookup table asserts a collision while in reality there is none. However, this is no issue in practice, since manoeuvres that almost result in a collision are undesirable, too. This is because a certain amount of robustness of a collision-free manoeuvre is required to compensate for unmodelled errors.

III. Discretisation and Quantisation

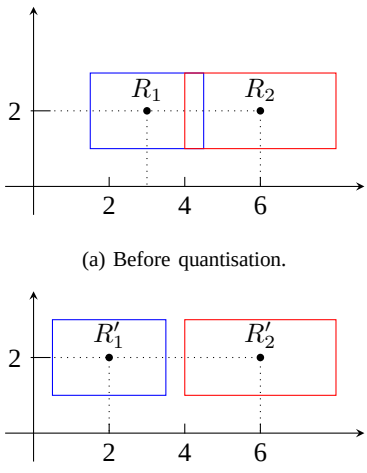
There are six dimensions in the input space to discretise:

$$(x_{\text{e}}, y_{\text{e}}, \theta_{\text{e}}) \in X_{\text{e}} \times Y_{\text{e}} \times \Theta_{\text{e}} \text{ and}$$
$$(x_{\text{o}}, y_{\text{o}}, \theta_{\text{o}}) \in X_{\text{o}} \times Y_{\text{o}} \times \Theta_{\text{o}},$$

where all sets are subsets of $\mathbb{R}$ and bounded. Any element lying outside these bounds is considered to be collision-free. Each dimension is discretised by fixing two constants: the sampling interval $\Delta_Z \in \mathbb{R}$ and the number of samples $N_Z \in \mathbb{N}$. The result of the discretisation for each dimension is the following finite set:

$$I_Z \;=\; \{i \cdot \Delta_Z \mid i \in \mathbb{N} \;\wedge\; 0 \leq i < N\} \tag{1}$$

Since the original set is infinite and the resulting set I above is finite, we have to define the mapping between these two sets, which is called quantisation [19]. Suppose that we have a set Z in the input space, its discretised set I, and its sampling interval Δ_Z, and $\lfloor \cdot \rfloor$ as floor function. We then



(a) Before quantisation.

(b) After quantisation.

Fig. 1. False negative due to quantisation error. The top figure is the actual scenario, while the bottom figure is the scenario after the quantisation. The step value for dimension X_{e} and X_{o} is 2 units of length. Rectangle R_1 and R'_1 have the width and length of 2 and 3 units, respectively. Meanwhile, rectangle R_2 and R'_2 have the width and length of 2 and 4 units.

quantise the dimension by defining a function $f_Z : Z \to I$ with the following formula:

$$f_Z(z) = \left\lfloor \frac{z}{\Delta_z} + 0.5 \right\rfloor \cdot \Delta_z \tag{2}$$

The main issue with the quantisation in (2) is the *quantisation error*. This error is the difference between the value which index i represents and the exact value z, that is $|z - i \cdot \Delta_z|$. This error[1] can make our collision detection method incomplete. That is, it can assert *no* collision while in reality there *is* a collision (false negative), which can possibly result in a crash.

This false negative situation happens when the overlapping region between two rectangles is smaller than half of the sampling interval (see Fig. 1). Figure (1b) shows that if we query the lookup table with the new coordinate, it will not assert a collision. However, Fig. 1a shows that both rectangles actually collide and, therefore, it shows that the current lookup table is incomplete. To achieve a complete collision detection method, we have to enlarge the original rectangle to compensate for this error.

IV. Rectangle Enlargement

We start this section by explaining the principle of rectangle enlargement, and later use it to compute the amount of enlargement for each dimension.

A. General Principle of Rectangle Enlargement

A rectangle R is enlarged to a rectangle R' such that R' contains all other rectangles mapped to R using (2). For example, in Fig. 1, since R_1 is mapped to R'_1, the enlarged version of R'_1 will contain R_1.

[1]From now, we will simply use the term 'error' when we mean 'quantisation error'.

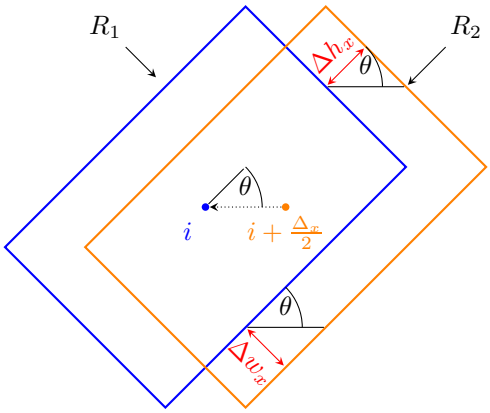


Fig. 2. Amount of enlargement due to error in dimension X. The blue and orange rectangle have the position and orientation of (i, y, θ) and $(i + \frac{\Delta_x}{2}, y, \theta)$, respectively.

The amount by which a rectangle is enlarged depends on the sampling interval Δ_z in (2). Suppose that we decide on a fixed value $i \in I$ in (1), then the bounds are the infimum and supremum of the set

$$\{z \in Z \mid f_Z(z) = i\}. \tag{3}$$

By using the property $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$, we obtain the infimum and supremum of the set in (3) as $i - \frac{\Delta}{2}$ and $i + \frac{\Delta}{2}$, respectively.

Figure 2 illustrates how one of the bounds (supremum) contributes to the required enlargement. From its position and the set (3), rectangle R_2 is mapped to R_1 and, therefore, R_1 is enlarged to include R_2. As the figure shows, we achieve this by increasing the width and the length of R_1 by Δw_x and Δh_x, respectively.

B. Amount of Rectangle Enlargement for Each Dimension

1) Dimension X: We calculate the enlargement for dimension X as follows. From Fig. 2, we can see that the values for Δh_x and Δw_x are

$$\Delta h_x = \frac{\Delta_x}{2} \cdot |\cos(\theta)| \quad \text{and} \quad \Delta w_x = \frac{\Delta_x}{2} \cdot |\sin(\theta)|. \tag{4}$$

These values, however, come from the contribution of the supremum only. By similar reasoning, we can also see that the infimum contributes the same values as in (4). Therefore, the total enlargement is the sum of the contributions from the supremum and the infimum:

$$\Delta H_x = 2 \cdot \Delta h_x = \Delta_x \cdot |\cos(\theta)| \tag{5}$$
$$\Delta W_x = 2 \cdot \Delta w_x = \Delta_x \cdot |\sin(\theta)| \tag{6}$$

2) Dimension Y: The amount of enlargement for dimension Y can be calculated similarly to (5) and (6):

$$\Delta H_y = 2 \cdot \Delta h_y = \Delta_y \cdot |\sin(\theta)| \tag{7}$$
$$\Delta W_y = 2 \cdot \Delta w_y = \Delta_y \cdot |\cos(\theta)| \tag{8}$$

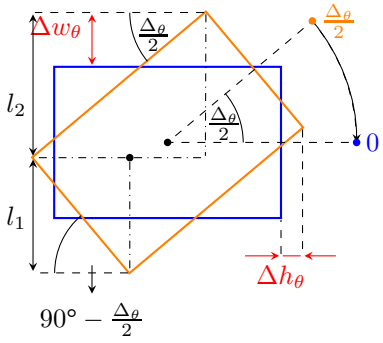


Fig. 3. Enlargement to compensate error from dimension Θ.

3) Dimension Θ: The principle discussed previously still applies to dimension Θ, but the amount of enlargement is different from the previous two dimensions. Figure 3 illustrates the enlargement required to compensate the error in dimension Θ. We derive the formula for ΔW_θ, which is twice of Δw_θ, as follows.

$$\begin{aligned}
& \Delta W_\theta \\
= \quad & \langle \text{by graphical reasoning from Fig. 3} \rangle \\
& |W - (l_1 + l_2)| \\
= \quad & \langle \text{by } l_1 = W \cdot \sin\left(90° - \tfrac{\Delta_\theta}{2}\right) \text{ and } l_2 = H \cdot \sin\left(\tfrac{\Delta_\theta}{2}\right) \rangle \\
& \left|W - W \cdot \sin\left(90° - \tfrac{\Delta_\theta}{2}\right) - H \cdot \sin\left(\tfrac{\Delta_\theta}{2}\right)\right| \\
= \quad & \langle \text{by factorisation and trigonometric identity} \rangle \\
& \left|\left(1 - \cos\left(\tfrac{\Delta_\theta}{2}\right)\right) W - \sin\left(\tfrac{\Delta_\theta}{2}\right) H\right|
\end{aligned}$$

The formula for ΔH_θ can be derived similarly and we formalise it as follows.

$$\Delta H_\theta = 2 \cdot \Delta h_\theta = \left|\left(1 - \cos\left(\frac{\Delta_\theta}{2}\right)\right) H - \sin\left(\frac{\Delta_\theta}{2}\right) W\right|$$

The enlargement discussed here is based on the superposition principle. We analyse the enlargement from each dimension independently and add them correspondingly. It is justified to use the superposition principle since rotation and translation are linear mappings.

V. Lookup Table Construction

This section details how the approach of discretisation, quantisation, and enlargement is used to construct the lookup table. First, we analyse the number of dimensions required for the lookup table. Then, we decide which information to store in the lookup table and how to obtain this information. Lastly, we present an algorithm to construct this lookup table and briefly report on its implementation.

A. Number of Dimensions

The collision detection lookup table has four dimensions. The first two dimensions are the relative distances between the centres of rectangles R_{other} and R_{ego} in dimension X and Y, denoted by $\overline{X}$ and $\overline{Y}$. The third and fourth dimension in the lookup table are Θ_{e} and Θ_{o}, the orientation of rectangles R_{ego} and R_{other}, respectively.

Due to position invariance, it is sufficient to consider relative values for the position. For example, two rectangles

Algorithm 1 CONSTRUCT-LOOKUP-TABLE

Input: $W_\mathrm{e}, L_\mathrm{e}, W_\mathrm{o}, L_\mathrm{o}$: dimensions of both rectangles
Output: $table[N_{\overline{X}}, N_{\overline{Y}}, N_{\Theta_\mathrm{e}}, N_{\Theta_\mathrm{o}}]$: lookup table

1: $ego \leftarrow$ RECTANGLE$(W_\mathrm{e}, L_\mathrm{e})$
2: $other \leftarrow$ RECTANGLE$(W_\mathrm{o}, L_\mathrm{o})$
3: **for all** $(i, j, k, l) \in N_{\overline{X}} \times N_{\overline{Y}} \times N_{\Theta_\mathrm{e}} \times N_{\Theta_\mathrm{o}}$ **do**
4: $\quad ego$.UPDATE_AND_ENLARGE$(0, 0, k)$
5: $\quad other$.UPDATE_AND_ENLARGE(i, j, l)
6: $\quad table[i_j, i_k, i_l, i_m] \leftarrow$ MEASURE_PEN$(ego, other)$
7: **end for**

with triples $(x_\mathrm{e}, y_\mathrm{e}, \theta_\mathrm{e})$ and $(x_\mathrm{o}, y_\mathrm{o}, \theta_\mathrm{o})$ will have the same collision status with two other rectangles with triples $(x_\mathrm{e} + \delta_x,\, y_\mathrm{e} + \delta_y,\, \theta_\mathrm{e})$ and $(x_\mathrm{o} + \delta_x,\, y_\mathrm{o} + \delta_y,\, \theta_\mathrm{o})$. This reduces the memory complexity of $\mathcal{O}(n^6)$ to $\mathcal{O}(n^4)$, where n is the maximum number of samples in each dimension.

Nevertheless, we avoid using relative orientation in the lookup table. This is because it requires a sine or cosine function to compute the relative index, which takes a considerable amount of time. This is not the case for relative position, however, since it only needs a subtraction operation.

B. Measure of Penetration as Collision Status

Rather than only storing the information about the collision status, we also add the information about the *measure of penetration* (cf. [20], [21]) between rectangles R_ego and R_other when they collide. If the measure has the value of zero, then we interpret it such that both rectangles do not collide; otherwise, they collide and R_other penetrates R_ego by the amount indicated by the measure.

The measure is useful as additional feedback to the manoeuvre planner. If it falls below a predetermined value, we could recheck the collision status with a more accurate but probably less time efficient method. In our case, it helps to address the issue of false positives in our approach.

C. Algorithm for Lookup Table Construction

Algorithm 1 illustrates the lookup table construction. The inputs to this algorithm are the width and length of both rectangles, and the output is a four-dimensional table. The algorithm mainly operates with the object *rectangle*, which we assume to contain all the sampling interval constants of all the sets to discretise.

The function UPDATE_AND_ENLARGE (line 4 and 5) updates the rectangle position and orientation according to the current entry (i, j, k, l), and enlarges the dimension according to the method described in Sec. IV. Meanwhile, the function MEASURE_PEN measures the penetration of R_other towards R_ego.

We implemented this algorithm in MATLAB 2014a with parameters listed in Table I. The time required to build the lookup table is $3839\,\mathrm{s}$ with an Intel i5-4330M $2.80\,\mathrm{GHz}$ processor and $12\,\mathrm{GB}$ of RAM. The lookup table consumes $23.37\,\mathrm{MB}$ of memory.

TABLE I
PARAMETERS FOR LOOKUP TABLE CONSTRUCTION

Parameter	$\overline{X}$	$\overline{Y}$	Θ_e	Θ_o
N	40	40	72	72
Δ	0.1482	0.1482	0.0885	0.0885

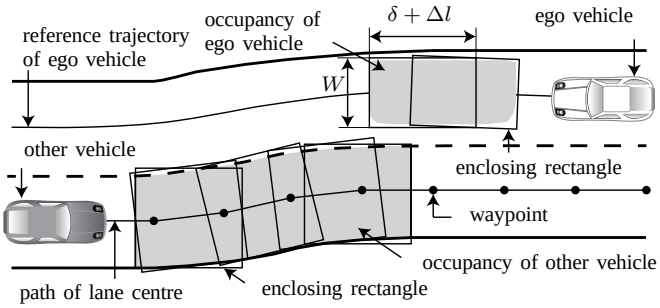


Fig. 4. Traffic scenario at time $[t_{k-1}, t_k]$.

VI. VERIFYING A PLANNED PATH

Previous sections focus on checking the collision between two rectangles. This section develops the idea further by showing how to verify a planned path. Firstly, we provide the general approach for verifying a planned path and show how it relates to collision checking of rectangles. The next subsection explains how the occupancy during a time interval is represented with a series of rectangles, and the last subsection provides the general algorithm for verifying a planned path.

A. General Approach for Verifying a Planned Path

We assume that the plan generated is represented by a series of equal-length and connected segments. The length of each segment is denoted by a constant $0 < \delta$, and for any two adjacent segments, the difference between their orientations is assumed to be at most ϕ_max, where $0 \leq \phi_\mathrm{max} < \frac{\pi}{2}$ (see Fig. 6).

Suppose that we have a time horizon $[t_0, t_f]$ which is divided into several smaller time intervals $[t_{k-1}, t_k]$ for $k = 1, \ldots, f$ (see Fig. 4). We also assume that there are two paths: one planned path for the ego vehicle, and the centreline of the lane which the other vehicle occupies. We verify the safety of the planned path relative to the predicted path by repeating the following steps until the time horizon t_f is reached [15]:

1) Compute the reachable set and the occupancies of the ego vehicles and the other traffic participant according to their dynamics and uncertainties at time interval $[t_{k-1}, t_k]$,
2) Cover the occupancy of the ego vehicle and the other traffic participants with rectangles,
3) Check the collision between any two combination of rectangles from the occupancies of the ego and the other vehicle.

If there is no collision for any of these time intervals, we conclude that the planned path is safe.

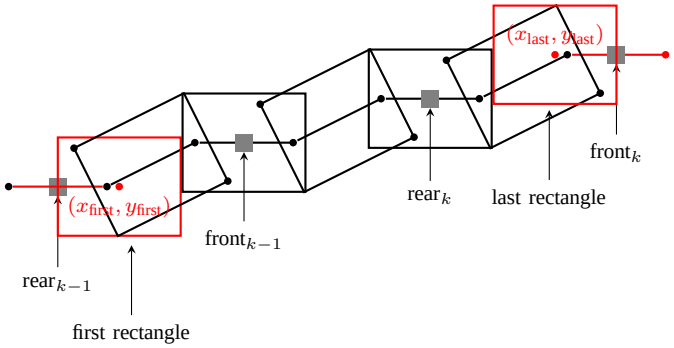


Fig. 5. Covering the occupancy at time $[t_{k-1}, t_k]$ with rectangles.

B. Covering Occupancies of Vehicles with Rectangles

Suppose that we have a time interval $[t_{k-1}, t_k]$ and a plan as illustrated in Fig. 5. The vehicle is assumed to occupy the region from rear_{k-1} to front_{k-1} at time t_{k-1} and the region from rear_k and front_k at time t_k. The region which needs to be covered with rectangles, therefore, ranges from rear_{k-1} to front_k.

Each rectangle is placed such that it has the same orientation as each segment in the path. The width of the rectangle covers the width of the vehicle and additional uncertainty: for the ego vehicle, the sources of uncertainty come from sensor noise, disturbance, and uncertain initial states; for other traffic participant, the source of uncertainty mainly comes from the model input (changing lane, accelerating or decelerating) [15]. Usually, the width of the R_{ego} is a slight enlargement of the width of the ego vehicle, and the width for the other rectangle is a slight enlargement of the lane width.

In order to determine the length of each rectangle, consider first the two adjacent segments in Fig. 6 which differ in orientation by $\phi_{\max}$. If we cover both segments with a rectangle of length δ, there is a gap between them which is shown as the quadrilateral $ABCD$. To cover this gap, we enlarge the length of the rectangle by

$$\Delta l \;=\; W \cdot \tan\left(\frac{\phi_{\max}}{2}\right). \tag{9}$$

Thus, the length of each rectangle is equal to the sum of the length of each segment and the enlargement above, i.e. $\delta + \Delta l$.

Since any two adjacent segments cannot differ in orientation by more than $\phi_{\max}$, we can safely enlarge every rectangle with the amount as in (9). This is mainly due to the fact that the tangent function is strictly increasing. Thus, any difference of orientations $|\phi| \leq \phi_{\max}$ requires an enlargement smaller than the value in (9). By doing so, we ensure that any potential gap in the path is always covered.

Except for the rectangles covering the first and the last segment, each rectangle is positioned at the midpoint between the two endpoints of each segment. Suppose that the length of the path from the beginning until $\mathit{rear}_{k-1} = (x_{k-1}, y_{k-1})$ is s_{k-1} and the orientation of the first segment is θ_{k-1}. Then

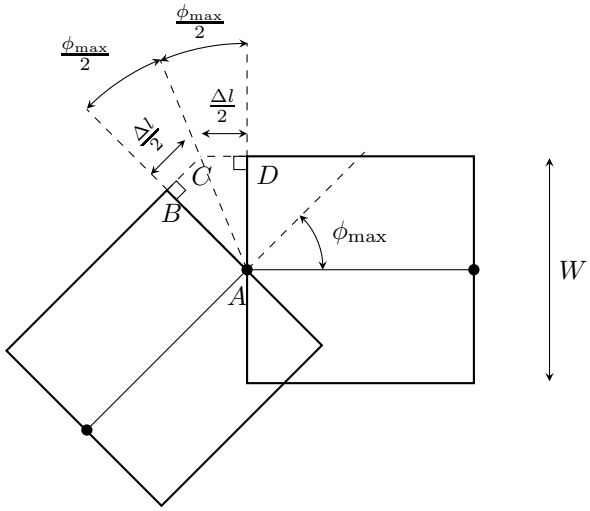


Fig. 6. Computing enlargement of the covering rectangle.

Algorithm 2 VERIFYING PLAN

```
collision ← false
for all [t_i, t_{i+1}) ∈ time do
    (s_i, s_{i+1}) ← FIND_EGO_ENDPOINTS(t_i, t_{i+1})
    rects_ego ← COVER_PATH_EGO(s_i, s_{i+1})
    (s'_i, s'_{i+1}) ← FIND_OTHER_ENDPOINTS(t_i, t_{i+1})
    rects_other ← COVER_PATH_OTHER(s'_i, s'_{i+1})
    for all r_e ∈ rects_ego and not collision do
        for all r_o ∈ rects_other and not collision do
            collision ← CHECK_COLLISION(r_e, r_o)
        end for
    end for
end for
```

the centre of the first rectangle has the coordinate

$$\begin{aligned} x_{\text{first}} &= x_{k-1} + \alpha_{k-1} \cdot \cos\left(\theta_{k-1}\right), \\ y_{\text{first}} &= y_{k-1} + \alpha_{k-1} \cdot \sin\left(\theta_{k-1}\right), \end{aligned}$$

where α_{k-1} is defined as follows:

$$\alpha_{k-1} = \left(\frac{\delta + \Delta l}{2}\right) - \left(\left\lceil \frac{s_{k-1}}{\delta} \right\rceil \cdot \delta - s_{k-1}\right)$$

Similarly, if s_k represents the length of the path from the beginning until $\mathit{front}_k = (x_k, y_k)$ and the orientation of the last rectangle is θ_k, then the centre of the last rectangle has the coordinate

$$\begin{aligned} x_{\text{last}} &= x_k - \beta_k \cdot \cos(\theta_k), \\ y_{\text{last}} &= x_k - \beta_k \cdot \sin(\theta_k), \end{aligned}$$

where β_k is defined as follows:

$$\beta_k = \left(\frac{\delta + \Delta l}{2}\right) - \left(s_k - \left\lfloor \frac{s_k}{\delta} \right\rfloor \cdot \delta\right)$$

C. Detailed Procedure for Verifying a Planned Path

Algorithm 2 details the procedure for verifying a plan. It consists of three loops of which the outermost loop

TABLE II

RANGE FOR EACH CLASS OF RANDOM CONFIGURATIONS

	X_o, X_e	Y_o, Y_e	Θ_o, Θ_e
Range	$[-5, 5]$	$[-5, 5]$	$[-\pi, \pi]$

traverses all the time intervals. At each time interval, function FIND_EGO_ENDPOINTS (line 3) finds s_i and s_{i+1}, the start and end points respectively, which the ego vehicle occupies at the current time interval (see Fig. 5). Function COVER_PATH_EGO (line 4) then covers the path with rectangles ranging from s_i to s_{i+1} as described in the previous subsection. These two steps are performed similarly for the other vehicle (line 5 and 6 of Alg. 2).

The two innermost loops traverse all combinations of rectangles which cover the occupancy of the ego and the other vehicle. For each combination, the function CHECK_COLLISION checks whether a collision has occurred using the proposed lookup table or the Separating Axis Theorem. We equip the guard conditions in line 7 and 8 with a test of whether a collision has occurred. Thus, as soon as status *collision* evaluates to true, the algorithm halts immediately and prevents the loop from checking the remaining combinations.

VII. EVALUATION

The purpose of this experiment is to evaluate the timing performance of the proposed approach against the approach based on the Separating Axis Theorem [2]. To do this, we divide the evaluation into two parts. The first part evaluates the timing performances when checking the collision of random rectangles, and the second part evaluates the timing performances when they are used for verifying a planned path.

A. Evaluating Random Rectangles

As a rule of thumb, a timing measurement is valid only if the execution time is at least $100\,\text{to}\,1000\,\text{times}$ its timer overhead [22]. Therefore, we decide to measure the timing performance in batch mode (i.e. several trials per test). We generate six classes of random data, and each class represents a dimension in the inputs. Table II defines the range of values for each of these classes. We generate $N_{\text{test}} = 100$ tests and each test has $N_{\text{trial}} = 550$ trials, and hence, $55\,000\,\text{random}$ configurations in total for each class.

Algorithm 3 illustrates how we measure the execution time for a single test of the proposed approach in batch mode. Function FIND_RELATIVE_POS computes the position (x, y) of R_{other} relative to R_{ego} by assuming $\theta_e = \theta_o = 0$. Meanwhile, the function COMPUTE_INDICES finds the indices (i, j, k, l) in the lookup table for each dimension in the argument list. Each index is computed with the formula f_Z/Δ_z, where the function f_Z is defined in (2) and the identifier Z ranges over all dimensions in the argument

[2]Note that we are evaluating our approach against Separating Axis Theorem only and not the whole OBB framework.

Algorithm 3 MEASURING PROPOSED APPROACH

1: $start \leftarrow$ GET_CURRENT_TIME()
2: **for all** $trial \in test$ **do**
3: $(x, y) \leftarrow$ FIND_RELATIVE_POS($trial$)
4: $(i, j, k, l) \leftarrow$ COMPUTE_INDICES(x, y, θ_e, θ_o)
5: $dist \leftarrow table[i, j, k, l]$
6: $status \leftarrow (dist = 0)$
7: **end for**
8: $end \leftarrow$ GET_CURRENT_TIME()
9: $elapsed \leftarrow (end - start)/N_{\text{trial}}$

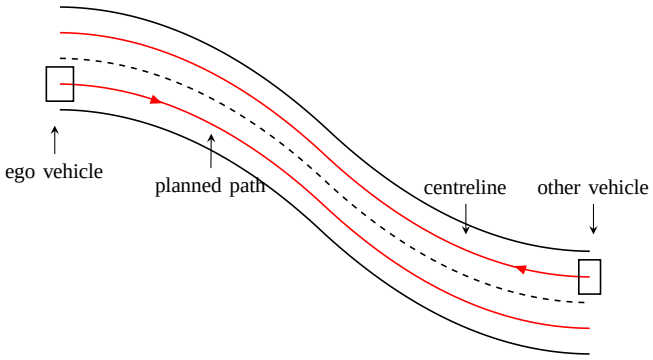


Fig. 7. First Scenario for measuring timing performance of planned path verification.

list. We use an algorithm similar to Alg. 3 to measure the timing performance of the approach based on the Separating Axis Theorem. The only difference is that lines 3–6 are replaced with specific instructions for the approach based on Separating Axis Theorem (see [10]).

We measure the execution time for both approaches by reading the current clock value in the hardware timer with WINDOWS' API QUERYPERFORMANCECOUNTER (QPC). The timer has a clock resolution of $333\,\text{ns}$ and access time of $30\,\text{ns}$ [3]. Each algorithm is implemented in C++ and compiled with MICROSOFT VISUAL C++ 2013 (optimisation level *O2*). As for the hardware, we use the machine specified in Sec. V to run this experiment.

B. Evaluating the Verification of a Planned Path

We devise two scenarios for evaluating the planned path verification. Figure 7 and 8 illustrate the first and second scenario, respectively. The first scenario depicts the situation where both the ego and the other vehicle follow the centreline of a curved lane. The second scenario shows the condition where the ego vehicle must avoid the static obstacle by crossing to the opposite lane, and also return quickly to prevent collision with the other vehicle.

1) Scenario 1: For the first scenario, we generate the tests by randomly creating the road model. To do this, we first generate the centreline for the lower lane. The centreline of the upper lane is then obtained by translating this line by the width of the lane[4]. From these two lines, we generate

[3]`http://msdn.microsoft.com/en-us/library/windows/desktop/dn553408%28v=vs.85%29.aspx`

[4]We assume that the lane width for both lanes are the same.

TABLE III

TIMING PERFORMANCE OF THE PROPOSED APPROACH AND THE SEPARATING AXIS THEOREM

	Random Rectangles ($N_{\text{test}} = 550$)		Scenario One ($N_{\text{test}} = 100$)		Scenario Two ($N_{\text{test}} = 100$)	
	Average	Std. Deviation	Average	Std. Deviation	Average	Std. Deviation
Lookup Table	6.66×10^{-9} s	4.20×10^{-9} s	2.73×10^{-5} s	1.79×10^{-6} s	2.78×10^{-5} s	2.29×10^{-6} s
Sep. Axis Thm.	1.06×10^{-7} s	1.42×10^{-8} s	2.22×10^{-4} s	1.88×10^{-5} s	2.69×10^{-4} s	2.65×10^{-5} s
Speedup	16.04		8.11		9.69	

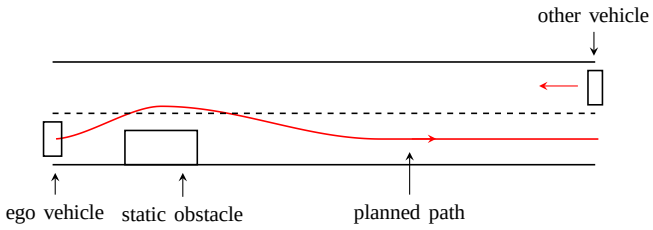


Fig. 8. Second scenario for measuring timing performance of planned path verification.

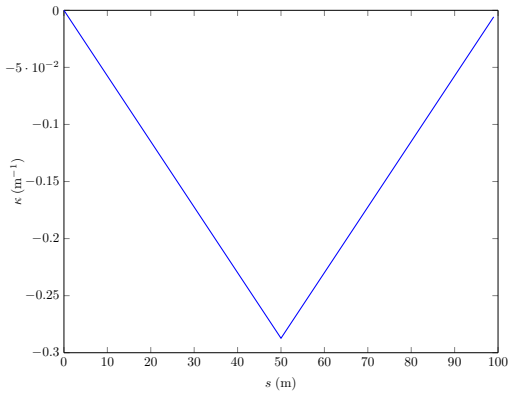


Fig. 9. Curvature profile for creating the lower lane of scenario 1.

the upper and lower border line and the line separating these two lanes.

We can generate centrelines for the lane by firstly creating random curvature profiles. For example, the centre line in the lower lane of Fig. 7 is generated by the curvature profile $\kappa(s)$ shown in Fig. 9. The curvature profiles for scenario 1 are obtained by randomising the values of the curvature at the initial, the middle, and the end of road parameter s, and connecting these values with straight lines. Lastly, the centrelines are obtained by solving the integral equations given by Dickmanns and Mysliwetz [23].

2) Scenario 2: Since scenario 2 has a fixed road model, the tests for this evaluation consist of the planned path for avoiding an obstacle and another vehicle. Instead of using the curvature profile, as in the previous scenario, we use Bézier curves of degree four to model the planned path [24] as shown in Fig. 10. After we obtain the Bézier curve, we

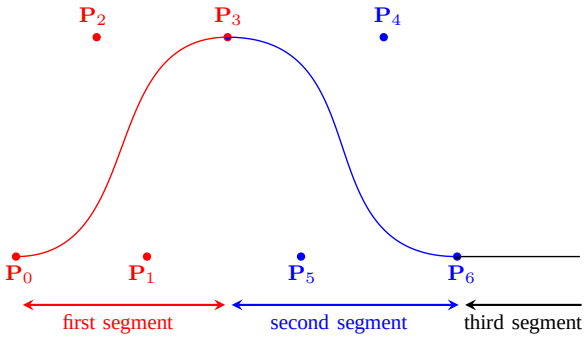


Fig. 10. An example of planned path for scenario two.

TABLE IV

NUMBER OF FALSE POSITIVES FOR LOOKUP TABLE APPROACH

	Random Rect.	Scenario 1	Scenario 2
false positives	4252	0	16
rate	7.73 %	0 %	16 %

reparameterise the curve as a function of arc length s [5] to obtain a set of equidistant points on our path.

We generate $N_{\text{test}} = 100$ tests for each scenario and they are generated from MATLAB code. The evaluation, however, is performed with Alg. 2 and it is implemented in C++. We use the same compiler, optimisation level, timer API, and hardware specification as in the previous subsection.

C. Results and Discussion

Table III summarises the timing performance of the proposed approach and the approach based on the Separating Axis Theorem. As can be seen from the table, our proposed approach is *faster* by a factor of 16.04 during collision detection of random rectangles. Additionally, it is *faster* on average by a factor of 8.9 when we verify the planned path.

Table IV reports on the number of false positives. For the random rectangles evaluation, a false positive occurs when our approach detects a collision, but there is actually no collision. For scenario 1 and 2, it is defined as a plan considered unsafe by our approach but it is actually safe.

[5]Our code is based on the following work. David Eberly. Moving Along a Curve with Specified Speed. Geometric Tools, LLC. 2007. `http://www.geometrictools.com/Documentation/MovingAlongCurveSpecifiedSpeed.pdf`

By looking at Tab. III, we can see that the execution time for the proposed approach is indeed smaller than the access time provided in Sec.VII. If we divide the execution time per test by the access time,

$$100 \;\leq\; \frac{550 \;\times\; 6.66 \times 10^{-9}\,\mathrm{s}}{3 \times 10^{-8}\,\mathrm{s}} = 122.1 \;\leq\; 1000$$

we see that it is in the range of valid measurement. Therefore, our choice of $N_{\text{trial}} = 550$ is also justified.

Table III also shows the timing performance for evaluating a single planned path with the approach based on the Separating Axis Theorem, which is in the order of $10^{-4}\,\mathrm{s}$. In a real-world implementation, a path planner could generate 10^4 planned paths and leading to verification times in the order of seconds. This is unacceptably long, and our approach helps to decrease verification times to fractions of seconds.

VIII. Conclusion and Future Work

We have described how we can trade memory for time in collision checking between rectangles, and how the safety verification of a planned path can be reduced to a series of collision checks between rectangles. We have also evaluated the proposed approach, and the results show that the timing performance improves on average by a factor of 8.9. As for future work, we intend to use real data from an autonomous vehicle to evaluate the proposed approach further.

Acknowledgement

The authors gratefully acknowledge financial support by the German Research Foundation (DFG) Graduiertenkolleg 1480 (PUMA) and Grant AL 1185/3-1.

References

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

[2] 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.

[3] 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.

[4] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified GPU voxel collision detection for mobile manipulation planning," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4154–4160.

[5] M. R. Schmid, M. Maehlisch, J. Dickmann, and H.-J. Wuensche, "Dynamic level of detail 3D occupancy grids for automotive use," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2010, pp. 269 – 274.

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

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

[8] 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.

[9] 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.

[10] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," *Computer Graphics*, vol. 30, pp. 171–180, 1996.

[11] 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.

[12] Y.-K. Choi, W. Wang, Y. Liu, and M.-S. Kim, "Continuous collision detection for two moving elliptic disks," *IEEE Transactions on Robotics*, vol. 22, no. 2, pp. 213–224, 2006.

[13] 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.

[14] T. Weiherer, S. Bouzouraa, and U. Hofmann, "An interval based representation of occupancy information for driver assistance systems," in *Proc. of the 16th International IEEE Conference on Intelligent Transportation Systems*, 2013, pp. 21 – 27.

[15] 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.

[16] 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.

[17] B. Martínez-Salvador, A. P. del Pobil, and M. Pérez-Francisco, "A hierarchy of detail for fast collision detection," in *Proc. of the IEEE/RSl International Conference on Intelligent Robots and Systems*, 2000, pp. 745 – 750.

[18] J. E. Savage, *Models of Computation: Exploring the Power of Computing*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[19] H. Kwakernaak, R. Sivan, and R. C. W. Strijbos, *Modern Signals and Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[20] S. Cameron and R. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3, Apr 1986, pp. 591–596.

[21] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri, "Computing the intersection-depth of polyhedra," *Algorithmica*, vol. 9, pp. 518–533, 1993.

[22] *Measuring Computer Performance: A Practitioner's Guide*. New York, NY, USA: Cambridge University Press, 2000.

[23] E. Dickmanns and B. Mysliwetz, "Recursive 3-d road and relative ego-state recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 199–213, Feb 1992.

[24] J. wung Choi, R. Curry, and G. Elkaim, "Path planning based on bézier curve for autonomous ground vehicles," in *World Congress on Engineering and Computer Science 2008, WCECS '08. Advances in Electrical and Electronics Engineering - IAENG Special Edition of the*, Oct 2008, pp. 158–166.