2018 21st International Conference on Intelligent Transportation Systems (ITSC)
Maui, Hawaii, USA, November 4-7, 2018

Roadgraph Generation and Free-Space Estimation in Unknown Structured Environments for Autonomous Vehicle Motion Planning

Tobias Kessler[1*], Pascal Minnerup[1*], Klemens Esterle[1*],
Christian Feist[2], Florian Mickler[2], Erwin Roth[2] and Alois Knoll[3]

Abstract— **Automotive manufacturers and customers wish to have fully automated driving functionality available in a huge set of locations, scenarios, and markets. This raises the need for universally applicable scene understanding and motion planning algorithms that do not rely on highly accurate maps or excessive infrastructure communication. In this paper we introduce two novel approaches for extracting a topological roadgraph with possible intersection options from sensor data along with a geometric representation of the available maneuvering space. Also, a search and optimization-based path planning method for guiding the vehicle along a selected track in the roadgraph and within the free-space is presented. We compare the methods presented in simulation and show results of a test drive with a research vehicle. Our evaluations show the applicability in low speed maneuvering scenarios and the stability of the algorithms even for low quality input data.**

I. Introduction

The first fully automated vehicles might have access to high definition maps and a very precise localization only in a limited set of locations. Unequivocally, such maps provide a high value but are expensive to generate and maintain. Therefore, the vehicle must be able to perceive an unknown environment (not covered by a map) through its own sensors and form a topological and geometric representation. Similarly, the reference track or driving corridor to follow has to be extracted from sensor data. These are essential for the subsequent motion planning methods which have to cope with this imperfect information.

This raises the question whether a smooth and stable autonomous vehicle motion can be achieved in such non-mapped environments and incomplete scenario information.

In this work, we present two novel approaches that structure the unknown environment based on sensorial observations. We obtain a topological roadgraph containing possible driving options and a geometric representation of the free-space available for maneuvering. Our algorithms provide topologically valid results with respect to (geometric) vehicle limitations. With direction clues from a navigation system or decision making entity, the most suitable network path is extracted and passed to a path planner that generates a drivable path. Along these paths jerk optimal trajectories are

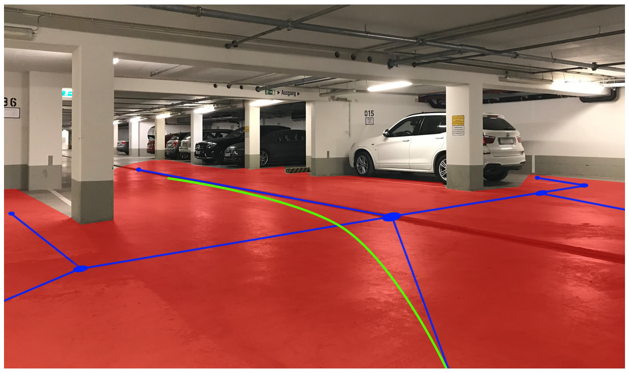


Fig. 1. Schematic illustration of the approximated free-space (red) and the roadgraph with direction options (blue). Based on the directional choice to go straight, a smooth path can be planned (green).

generated and stabilized by a controller. The algorithms are validated using a prototype autonomous vehicle equipped with a Lidar sensor and an appropriate sensor data processing pipeline.

Our main contribution is the presentation of algorithms solving the scene understanding and path planning aspects of the problem. Fig. 1 illustrates the methods in a parking garage scenario. Our work focuses on the presentation of

- two algorithms to generate a topological roadgraph and to estimate the available free-space from fused sensor information,
- a comprehensive description, comparison and evaluation of the methods,
- a two-stage path planning method based on a graph search and numerical optimization,
- the description of the full motion planning system setup from sensor information to vehicle control commands.

This paper is organized as follows. After a brief problem statement in Section II and a literature review in Section III, Section IV introduces two roadgraph and free-space creation algorithms. Based on these, the path planning method is presented in Section V followed by the analysis of an exemplary test drive Section VI. Section VII and Section VIII discuss the results and conclude this work.

II. Problem Definition and System Overview

The system presented in this paper is embedded in a fully automated vehicle platform with a proprietary architecture and further black-box components. Using a human machine interface, it can also be used as an assistance system.

[1]Tobias Kessler, Pascal Minnerup and Klemens Esterle are with fortiss GmbH, An-Institut Technische Universität München, Munich, Germany. Pascal Minnerup left fortiss GmbH before submitting this work.
[2]Christian Feist, Florian Mickler and Erwin Roth are with AUDI AG, Ingolstadt, Germany
[3]Alois Knoll is with Robotics, Artificial Intelligence and Real-time Systems, Technische Universität München, Munich, Germany
[*]These authors contributed equally to this work.

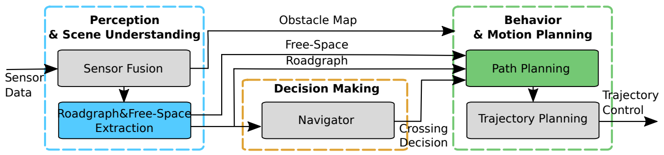


Fig. 2. Architecture overview of the developed system. We focus on the roadgraph and free-space extraction and the path planning in this work.

Within the system architecture shown in Fig. 2, we focus on the roadgraph and free-space extraction module further discussed in Section IV that delivers

- the topological roadgraph,
- the geometric free-space representation,
- the list of crossings the navigator can base crossing decisions on.

By *roadgraph* we denote the network of possible reference path options. The roadgraph covers the topology of the environment and graph edge lengths shall fit to real-world distances. The roadgraph does not contain drivable paths by the non-holonomic vehicle. *Free-space* is the obstacle-free area within the environment that is available to the vehicle for safe maneuvering. Parking lots or other small free-spaces are distinguished from broad corridors. The figures in Section IV show exemplary roadgraph and free-space representations.

The subsequent navigation relies on the integrated crossing detection algorithm. The detection needs to be robust, as falsely identified crossings lead to a less smooth path and hence decrease the comfort. Missing crossings might lead to the vehicle not reaching its desired goal.

The major components interacting with the developed system are

- a model representing the physical environment in the vehicle's sensor range consisting of obstacles,
- a navigating entity to take a direction decision at each intersection,
- the odometry data of the vehicle.

The obstacle map changes dynamically and represents the environment in a polygonal way, cf. Kubertschak *et al.* [1]. This map will be referred to as *fences* in the following. A centralized sensor data fusion system incorporates all sensor data and provides a set of fences, but is not discussed in this work. The obstacle map contains static and dynamic obstacles. With the periodic replanning strategy, the planned paths can also cope with moving obstacles.

The roadgraph including a list of crossings ahead is passed to a navigator that decides which direction to take at each intersection. This component can rely on its own knowledge of the topological structure of the surroundings and its goals, the in-vehicle navigation system, commands from a communication unit, other information sources or feedback.

The path planner receives this decision together with the roadgraph to periodically generate new drivable paths along the selected roadgraph edges. The planner is restricted to only use the calculated free-space. We describe this in more detail in Section V. It passes the physically drivable path to the trajectory planner that operates at a higher frequency and can react to small changes in the environment and to obstacles moving fast.

III. Related Work

We present two methods to generate a roadgraph and approximate the free-space. The first makes use of the well-known construction of a Voronoi diagram (see e.g. LaValle [2]), the second is based on covering the environment with a series of circles inspired by the work of Chen *et al.* [3]. The *Orientation-Aware Space Exploration Guided Heuristic Search Method* generates a drivable path for a non-holonomic vehicle using a series of circles. Applying this approach to our problem, we aim to represent the whole free-space instead of only a driving corridor. The circle-based path planning heuristic is compared to a Hybrid A* and a RRT motion planner and proves to be faster in the evaluated scenarios. The heuristic is suitable for navigating in narrow spaces and unstructured environments, which is relevant to the problems discussed in this work.

To generate a drivable path on the roadgraph, we use a variant of the Hybrid A* path planning algorithm [4] combined with a novel path optimization technique. Dolgov *et al.* show the implementation of an A* search algorithm with a non-holonomic heuristic ignoring obstacles combined with a holonomic heuristic taking obstacles into account. As a cost function for the necessary post-optimization, a generalized Voronoi diagram is used that generates cheap paths entering narrow corridors, which is not the case using standard potential field approaches. Topological roadgraphs guide the graph search along a lane network [5].

Similar to Tanzmeister *et al.*, we treat the goal pose as unknown. An A*-based and a RRT-based planner is combined to quickly find candidate goals and clusters of reasonable paths in a dynamically changing environment [6].

Based on nonlinear optimization, Li and Shao solve static autonomous vehicle parking problems [7]. The problem is formulated as a mathematical model and optimized for minimal time in various rather small simulated parking scenarios including maneuvering. In contrast to their work, we do not rely on a fixed obstacle map and generate a curvature-optimal motion instead of a time-optimal one.

A substantial amount of research is devoted to road and lane detection [8] with free-space approximation using cameras, Lidar, or a fusion of both. Camera images can either be processed using classical methods [9] or machine learning techniques [10]. Na *et al.* [11] show that even in complex road situations the drivable free-space can be extracted from Lidar point cloud data. Using a model-based and region-based method, the drivable space is expanded starting near the vehicle. A setup with multiple Lidar and camera sensors and an appropriate sensor data fusion can solve both, the extraction of the drivable free-space as well as a detection of possible lanes [12]. With a fusion of one Lidar and one monocular camera an unsupervised learning approach can be implemented to detect the drivable free-space. After a preprocessing step, a Markov network and belief propagation is applied to obtain robust results [13]. In our work, we use already fused and processed sensor data. This offers the flexibility to seamlessly change the sensor set but needs to cope with changing data quality.

IV. Scene Understanding

This section introduces two alternative solutions for the geometric exploration for roadgraph extraction and free-space generation based on the sensed obstacles. Section IV-A uses a circle-based heuristic whereas Section IV-B is based on the creation of a Voronoi diagram. We chose a cyclic regeneration with a frequency of $1Hz$ which proved to be sufficient in the evaluated low speed scenarios.

A. The Heuristic Circle-based Approach

1) Roadgraph Generation: The method essentially fits circles into the free-space and connects their centers. Algorithm 1 shows the implementation.

Algorithm 1: Circle-based Geometric Exploration

```
Data: VehiclePos, Map
Result: Circles, Parents
Parameter: $H$, $R_{min}$
1  Circles = {};
2  Parents = {};
3  Candidates = {LargestCircleAround(VehiclePos)};
4  while Candidates $\neq \emptyset$ do
5      Current = $argmax_{c \in Candidates}(Radius(c))$;
6      Candidates = Candidates \ Current;
7      if Center(Current) $\notin$ (Area(Circles) $\cup$ $H$) and
        Radius(Current) $\geq R_{min}$ then
8          Circles $\overset{\cup}{\leftarrow}$ Current;
9          foreach $c \in$ Perimeter(Current, N) do
10             Next = LargestCircleAround(c);
11             Parents $\overset{\cup}{\leftarrow}$ (Current,Next);
12             Candidates $\overset{\cup}{\leftarrow}$ Next;
```

The algorithm expects the current position of the ego vehicle *VehiclePos* and the obstacle map *Map* of the environment as inputs. It computes a set of circles covering the free-space in the environment and a relationship containing a parent for each circle. It starts with expanding a circle around the current position of the vehicle in line 3. The function *LargestCircleAround(VehiclePos)* computes the largest circle around the passed position *VehiclePos* that does not intersect with any obstacle in the map. Additionally to the sensed obstacles, this function assumes an additional horizon polygon H outside of which everything is considered colliding. The circle expanded in line 3 is the first element of the candidate circles. The actual exploration is performed in the following while loop. In each iteration, the largest of the candidate circles is expanded and removed from the candidate list. If this circle is smaller than the threshold R_{min} or its center is contained in the area covered by all circles in the result set, the circle is discarded and otherwise added to the result set. The circles around N equally distributed points of the perimeter of it are added to the candidate set. For each new circle, the parent circle is stored.

The resulting tree of circle centers presents a roadgraph that the vehicle can follow. However, Fig. 3 shows examples in which this graph does not correspond to intersection decision points. Therefore, some circles are dropped. The solid orange circles are discarded due to their lack of size. Circles without child circles are discarded too as those are likely to correspond to wrong crossings. These solid blue circles form false positive crossings which indicate a not-existing directional choice. Through appropriate choice of the minimum circle radius most false positive crossings can be avoided.

2) Free-space Approximation: The circles give a conservative bound on the available free-space as each circle borderline touches at least one obstacle but is collision free. The algorithm on the one hand aims to find circles as large as possible, but also aims to cover the whole free-space with circles. Therefore, the union of circles approximates the available free-space. Algorithm 2 states the generation process.

Algorithm 2: Circle-based Free-Space Approximation

```
Data: CircleTree, BoundingBox
Result: Freespace // Freespace Polygon
1  SampledCirclesList = sampleCircles(CircleTree);
2  Freespace = union(SampledCirclesList);
3  Freespace = intersection(Freespace,BoundingBox);
4  Freespace = simplify(Freespace);
```

As we aim for a polygonal representation of the free-space, we sample each circle as a polygon and join those. To reduce the complexity of the resulting polygon, we apply the Namer–Douglas–Pecker algorithm as implemented in the Boost Geometry Library [14] to simplify the polygon. Note, that after the simplification the bound is not conservative any more as obstacles might now enter the polygon due to the removal of some points. Section V-E shows how we cope with this fact. The number of circles in the circle tree as well as the chosen circle discretization interval and the simplification distance define the runtime of the algorithm.

Fig. 4 shows an example of how the free-space polygon is formed from the circles. The resulting polygon is a closed area and is used to restrict the path planning space.

B. The Voronoi Diagram-based Approach

1) Roadgraph Generation: Voronoi diagrams are well known in the literature and standard implementations exist. We use the implementation from the Boost Polygon Library [14]. Algorithm 3 sketches the implementation of the roadgraph generation based on a Voronoi diagram. First, the Voronoi diagram is created based on the environment map Map within a sufficiently large bounding box. This ensures that the Voronoi edges do not end at obstacles, but lead into the free-space as can be observed in Fig. 5. Regions that are

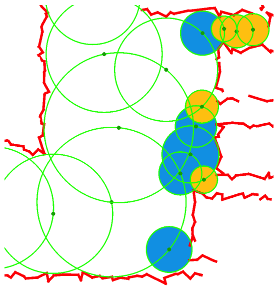


Fig. 3. Special cases of the circle-based roadgraph generation. Obstacle fences in red, circles with centers in green, deleted circles in solid colors, see Section IV-A.1 for details.

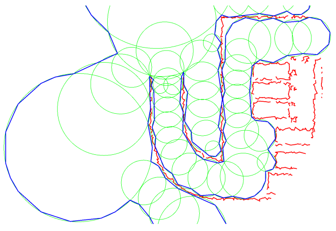


Fig. 4. Free-space approximation using the circle-based method; obstacles in red, circle tree in green, combined and simplified free-space polygon in blue.

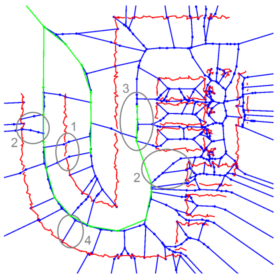


Fig. 5. Special cases and post-processing steps of the Voronoi-based exploration. Fences in red, raw Voronoi diagram in blue, final roadgraph in green. The gray numbers are referred to in Section IV-B.1.

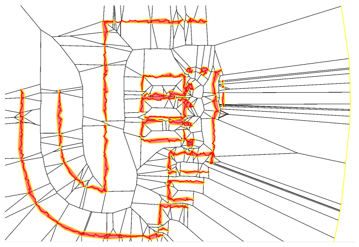


Fig. 6. Free-space approximation using Voronoi cells with the cells in black, obstacle fences in red, shape of the free-space approximation in yellow and obstacle approximations in light red.

```
Algorithm 3: Voronoi Diagram-based Exploration
  Data: VehiclePos, Map, BoundingBox
  Result: (E,V) // Edges, Vertices
  Parameter: $d_{max}$
1 Map = simplify(Map);
2 (E,V) = createVoronoiDiagram(Map ∪ BoundingBox);
3 (E,V) = removeCollidingEdges(E,V,Map);
4 (E,V) = removeShortBranches(E,V,Map);
5 (E,V) = combineSmallEdges(E,V,Map);
6 (E,V) = shrinkToFirstMeters(E,V,VehiclePos,$d_{max}$);
```

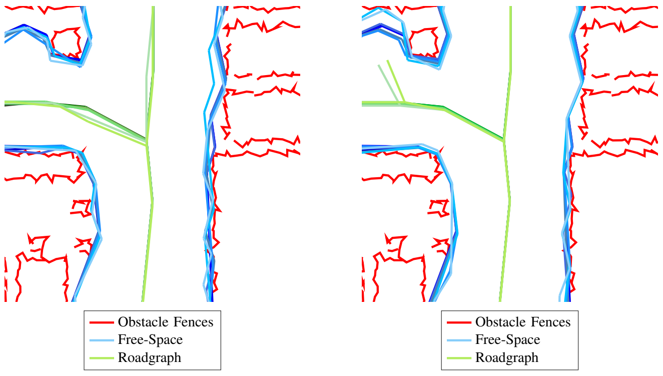
out of the range of the vehicle's sensor system form a large free-space. The resulting graph also contains edges leaving the currently explored parts of the map. The result by the off-the-shelf Voronoi algorithm has to be post processed in order to generate a valuable roadgraph.

Fig. 5 shows various cases requiring a post-processing of the Voronoi diagram. It contains edges that pass very small gaps in the fences polygon (marked as 1 in Fig. 5). Such holes are often caused by sensor inaccuracies and do not exist in reality. Even if they do exist, the vehicle is not able to drive through them. Therefore, all edges that collide with an obstacle are removed from the resulting graph in line 3 of algorithm 3. After removing the colliding edge, a branch leading to this edge might still remain part of the Voronoi diagram. This behavior is not desired, as it leads to false positive crossing detections (indicated as 2 in Fig. 5). Therefore, the next step removes all branches that are shorter than a configurable size. In some cases, subsequently removing short branches can add up to removing a long branch. In this case, the longest branch is kept in the graph (see 3 in Fig. 5). Furthermore, a non straight obstacle edge can lead to short edges starting from the main graph of the Voronoi diagram. These are also removed to avoid false positive crossing detections (marked as 4 in Fig. 5). In order to limit the number of nodes in the resulting graph, line 5 combines subsequent short edges to longer ones. Finally, line 6 explores the graph starting at the node nearest to the vehicle with a maximum depth d_{max}. It has to be set according to the range of sensor measurements. All nodes that are not explored are removed from the resulting graph. This step also removes all parts of the graph that lost connection to the graph in line 3 of the algorithm. The graph is subsampled with a fixed sampling distance afterwards.

2) Free-space Approximation: From auxiliary data from the Voronoi diagram creation a straight forward free-space estimation can be achieved as sketched in algorithm 4.

```
Algorithm 4: Voronoi Cell-based Free-Space Approximation
  Data: VoronoiDiagram, ObstacleEdges, CarPosition
  Result: Freespace // Freespace Polygon
1 VoronoiCells = getAllVoronoiCells(VoronoiDiagram);
2 ObstacleEdges = inflate(ObstacleEdges);
3 ObstacleEdges = simplify(ObstacleEdges);
4 FreespaceDecomposition =
    difference(VoronoiCells,ObstacleEdges);
5 FreespaceCandidates = union(FreespaceDecomposition);
6 Freespace = findContainingPolygon
    (CarPosition,FreespaceDecomposition);
```

The set of points that are nearest to one Voronoi edge or point form a polygon called a Voronoi cell. These cells are generated during the computation of the Voronoi diagram. Voronoi cells can serve as a natural basis for an approximation of the free-space. As the cells can intersect with Voronoi edges (obstacles), the cells are not collision free and have to be augmented. Algorithm 4 states the necessary steps and Fig. 6 illustrates the process. As edges of Voronoi cells can lie on obstacles, all obstacles are inflated by a constant distance (of few centimeters) which also adds some conservatism. We use the Ramer–Douglas–Peucker algorithm to simplify the obstacle edges, which reduces the complexity of the border of the freespace. This step significantly improves the runtime of the algorithm. The resulting union of all Voronoi cells are not necessarily a cohesive area. The one containing the current vehicle position is picked.



(a) Circle method (b) Voronoi method

Fig. 7. Comparison of the extracted free-space and the generated roadgraph for varying sensor uncertainty resulting in noisy fences. Only the worst case (most noisy) fences are displayed in red. Varying color values from dark to light for the free-space border (blue) and reference path (green) indicate the transition from ideal to worst-case fences.

C. *Comparison of Circle and Voronoi Exploration Method*

The two introduced geometric exploration methods have different strengths and weaknesses.

The circle-based exploration method is independent of the obstacle map representation and requires only little post processing effort. Further it requires low effort for additional customizations or cost terms. For example, the shape of the resulting paths can be influenced by limiting the circle size or by rewarding to drive straight in large free areas. Also additional sensor information like detected lanes or warning markers can easily be incorporated.

For the Voronoi-based exploration method, the geometric shape of the resulting path is well defined leading to a predictable and stable vehicle behavior. Furthermore the creation of a Voronoi diagram and the geometric post-processing methods can be used from a optimized standard library implementation improving runtime and maintainability. A disadvantage is the necessary post processing of the geometric extraction results.

As both of our free-space and roadgraph generation is based on already processed sensor data, it has to be stable for bad (noisy, inaccurate) results of the preprocessing. Fig. 7 shows an example of the circle-based and the Voronoi-based method coping with unfavorable fences input data. The circle-based method detects the T-crossing on the roadgraph very stable even for bad input data. Parts of the free-space are cut off but in a acceptable way. The Voronoi-based method is more likely to detect undesired crossings that could not be equalized in the post processing steps. On the other hand, the output of the free-space generator using Voronoi is very stable, even when dealing with highly noisy fences data. In the experiment the vehicle position is fixed. A moving vehicle yields varying sensor data and hence comparable results.

V. Motion Planning

The path planner is executed in an anytime manner. As soon as the environment, the roadgraph or the goal location change, a replanning is triggered. As a starting point, we choose the current position of the vehicle.

We implement two major components for path planning, a graph-search-based A* planner and an optimization-based planner. We briefly describe the functionality of both and how both methods are applied to plan a drivable path based on the reference path from the roadgraph.

A. *Navigation*

The navigation can either be performed automatically or on-the-fly while driving. In automatic navigation mode, a goal position is picked on a topologically correct map. The automatic navigator then determines the sequence of crossing decisions in order to reach the goal position. As the crossing decisions do not depend on correct lengths, the map does not need to be geometrically correct. An accurate localization within the map is not necessary either. In the latter case, a new direction decision is requested from the navigator at each crossing.

The result of the navigator is a list of direction decisions for each crossing to be passed on the current roadgraph. The edges to follow are extracted from the roadgraph in order to generate a reference path. We cut off the end of this reference path and limit its length to $20m$ as the latter part of the path is likely to change with future updated sensor information.

B. *Optimization-based Path Planning*

The computed reference path generated by the free-space generation module is not drivable by the vehicle yet. We aim to stick to the roadgraph path, denoted as reference path (x_{ref}, y_{ref}) as close as possible to not lose any information and formulate a nonlinear optimization problem to compute a smooth and collision free path. The initial $3m$ are cut off and replaced by a straight connection to the current vehicle pose as starting point. The reference path is re-sampled in an equidistant manner to contain n points.

We plan inside the free-space polygon denoted by $\mathcal{F}$ constructed in Section IV. Starting from the reference path as initial solution, we formulate an optimization problem to find a drivable path inside a closed polygon that is optimal with respect to an objective function. We use the state vector

$$\underline{x} = \begin{pmatrix} x & y & \theta \end{pmatrix} \tag{1}$$

and the discretized form of the vehicle model

$$\dot{\underline{x}} = \begin{pmatrix} \cos\theta & \sin\theta & \kappa \end{pmatrix} \tag{2}$$

with the vehicle pose x, y, orientation θ and steering curvature κ to describe the vehicle motion along the path with n points. As optimization vector we chose

$$\underline{u} = \begin{pmatrix} \kappa_1 & \dots & \kappa_n & h \end{pmatrix} \tag{3}$$

with the model integration step-size h. The optimization problem is formulated as follows

$$\min\ J = j_{offset} + j_{curvature} + j_{length} \tag{4}$$

$$\text{subject to}\quad dist(\mathcal{V}, \mathcal{F}) > 0 \tag{5}$$

$$\kappa_{min} \leq \kappa_i \leq \kappa_{max}\ \forall\ 1 \leq i \leq n \tag{6}$$

$$h_{min} < h < h_{max} \tag{7}$$

with the cost terms

$$j_{offset} = w_{offset} \sum_{i=1}^{n} \left((x_i - x_{i,ref})^2 + (y_i - y_{i,ref})^2\right) \tag{8}$$

$$j_{curvature} = w_{curvature} \sum_{i=2}^{n} (\kappa_i - \kappa_{i-1})^2 \tag{9}$$

$$j_{length} = w_{length} h \tag{10}$$

using appropriate scaling factors w and upper and lower curvature and step-size bounds. By $\mathcal{V}$ we denote the approximation of the vehicle shape with a series of circles, cf. Lenz *et al.* [15]. We use the Euclidean distance as distance function in (5) and calculate the necessary derivatives for the optimization solver. This nonlinear optimization problem is solved using the SLSQP algorithm drawn from the NLOpt library [16]. In the analyzed scenarios, the reference path proves to be a good initialization, the optimization converges after a few iterations.

C. Hybrid A* Path Planning

The reference path can also contain sections where it is not possible to follow the path without maneuvering, e.g. sharp corners. Using the graph-search-based Hybrid A* approach [4], we plan a path from the current vehicle position to the end point of the reference path. This way, we may introduce reverse pulling steps if not possible otherwise. As a heuristic, the well-known Reeds-Shepp curves are used (see e.g. [2]).

To gain a curvature continuous path the result of the Hybrid A* planner is post optimized segment-wise using the algorithm introduced in Section V-B.

D. Two Stage Execution of both Planning Algorithms

The optimization-based planning tends to stick close the reference path whereas the Hybrid A* finds the shortest connection from start to goal. Often, the former behavior is desirable. Therefore, we first trigger an optimization-based planning. If it fails, we use the Hybrid A* planner to introduce maneuvering or moves with huge deviation from the reference. If no path can be found at all, we stop the vehicle as the goal cannot be reached within the given free-space. Note that due to a probably exhaustive tree search, the Hybrid A* algorithm is computationally more expensive than solving the rather small optimization problem. With the short reference path the optimization problems are solved in a few hundred milliseconds at worst. For the Hybrid A* the short start to goal distance also mostly yields moderate evaluation times clearly below one second, but complex obstacle configurations can yield high evaluation times of some seconds. If a new reference path is received and the path planning is still ongoing the planning is canceled and restarted on the new data.

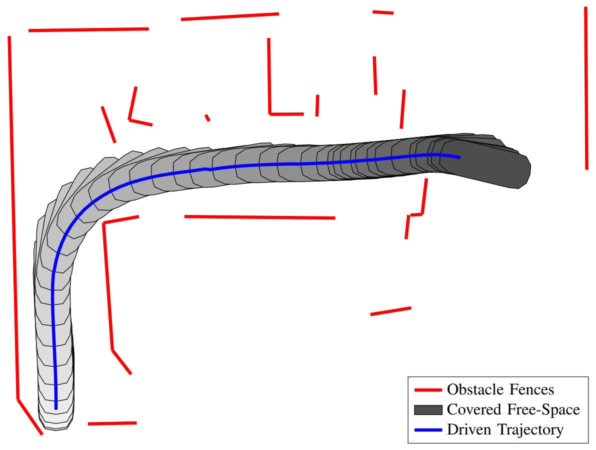


Fig. 8. Exemplary test drive with a prototype vehicle showing a simplified static obstacle map and the vehicle motion. Light to dark shapes depict increasing time.

E. Trajectory Generation and Control

The generated path is passed to a trajectory generator. The trajectory generation approach is based on [17] and outputs a comfortable (jerk minimal) and safe (collision free) trajectory. It is executed at a frequency of $2Hz$ and updates the obstacle map at each step. The collision check is executed on the obstacle map from the sensor data fusion component and not on a simplified free-space polygon. This guarantees a collision free motion or an emergency brake, if the tracked path collides with an obstacle. The input path might collide, as it is only checked against a simplified environment representation or the environment changed in the meantime as the path planning components runs on a lower frequency. The trajectory generator samples a fan of polynomial trajectories regarding spatial and temporal length and lateral offset from the path. This ensures a collision free motion, even if the input path collides. The resulting jerk optimal, collision free trajectory is stabilized using a state feedback controller with a cycle time of $20ms$.

VI. Evaluation on a Prototype Autonomous Vehicle

The described algorithm was evaluated on a prototype autonomous Audi A6 vehicle equipped with a front Lidar sensor with 144 degree field of view and an appropriate (proprietary) sensor data post processing to generate a map of obstacle fences.

A. Exemplary Test Drive Evaluation

We drive the vehicle through a parking garage passing a T-junction and various obstacles like parked vehicles and free-space areas. The directional decisions are received while driving by the navigator. The results are shown in Fig. 8. The shown trajectory and vehicle motion are recorded from

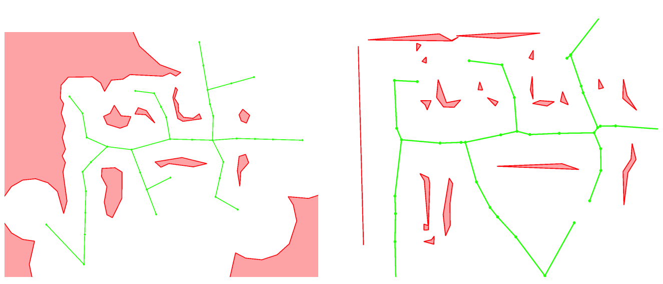

(a) Circle method (b) Voronoi method

Fig. 9. Comparison of circle and Voronoi method on the same map omitting the sensed fences. The free-space is white whereas obstacle areas are depicted in red. The roadgraph is shown in green.

the test drive. As the full time horizon is displayed, we do not show the actually observed and processed fences, but a simplified image of the sensed environment. A stable and smooth motion as well as a robust roadgraph and free-space representation is achieved. Repeating the experiment yields comparable results. As free-space and roadgraph creation method, the circle-based approach was used. A comparable motion is achieved using the Voronoi-based method.

Fig. 9 shows a comparison of the roadgraph generated by the circle and the Voronoi method on the same map of fences taken from one time instance of the test drive above. Both graphs are comparable from a topological point of view. The leftmost T-junction is approximated more accurately by the Voronoi method; the circle method generates a Y-formed shape. The free-space for the circle method is limited by the circle exploration depth, uncovered areas are marked as obstacles. The circle method furthermore tends to turn straight lines into curved obstacles. Smoothing can help to attenuate this but introduces the risk to underestimate obstacles. With significant differences in the shape of the free-space, the drivable area is covered in a comparable way.

The resulting drivable and collision free paths at one time instance of the test drive are shown in Fig. 10. It can be observed that different roadgraphs lead to different paths. Both paths succeed in taking the right turn a the T-junction. Due to the limited sensor range, the shape of the free-space at the end of the path is fairly unknown. Therefore the optimization is parametrized to only reach the goal point very roughly. Shifting the vehicle position (in simulation) the path optimization fails and a Hybrid A* path is planned for the same situation as depicted in Fig. 11. Due to the vehicle's limited steering angle, the T-junction cannot be passed without maneuvering.

B. Stability of the Crossing Detection

The stability of the circle-based and the Voronoi exploration method can be compared by monitoring the positions of the detected crossing in a T-junction while the vehicle passes by. Fig. 12 shows the result of this comparison. The position of the crossing detected by the Voronoi-based method deviates only by a few centimeters. In contrast, the crossings detected by the circle-based method deviate

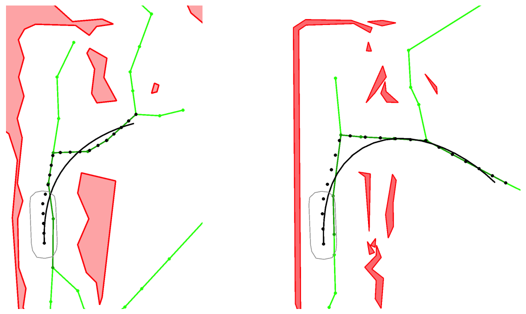

(a) Circle method (b) Voronoi method

Fig. 10. Comparison of the planned vehicle path using the optimization-based planning by circle and Voronoi method omitting the sensed fences. The free-space is white whereas obstacle areas are depicted in red. The roadgraph is shown in green, the part of the roadgraph to follow aka. the reference path in dark green. The planned path in shown in solid black, the pre-sampled path from the roadgraph as black dots. The current vehicle position/planning start point is depicted as grey vehicle shape.

by several meters. The detected crossing positions should correspond to the location the decision making entity would expect the actual crossing to be. All positions computed by the circle-based methods lie within the crossing and are hence valid. However, the navigator or an algorithm tracking the decision points has to cope with the position variance.

This further indicates that with the moving vehicle that Voronoi-based roadgraph is more stable.

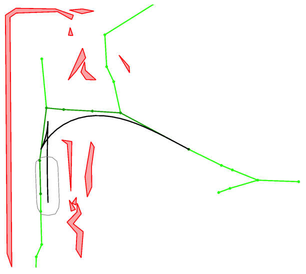


Fig. 11. Same situation as in Fig. 10, but with a different vehicle position. The optimization-based planner fails to find a path and the Hybrid A* planner introduces a maneuvering step.

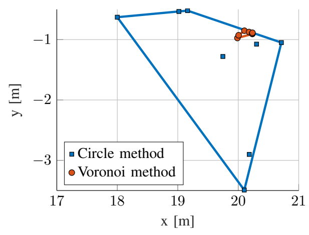


Fig. 12. Distribution of the extracted crossing position while the car is approaching the crossing (extracted from driving data)

VII. Discussion

Choosing a suitable roadgraph generation method depends on the actual use-case. If the method is only applied to a detected obstacle map as investigated in this paper, the Voronoi-based method outperforms the circle-based method in stability and practical applicability. If it is extended to incorporate more information and restrictions, such as further goals or areas to avoid, the circle-based method offers more flexibility and can cope better with noisy input data. With neither method, we encountered evaluation time issues. With a strong focus on performance we claim that the circle-based method can be implemented more efficient than the Voronoi-based method. In the evaluated low speed scenarios with a potentially high number of obstacle fences the approaches have been proven to be applicable in real time. All in all, both methods solve the problem analyzed in the work.

The optimization-based path planning has shown to converge fast to a valid and drivable path even with a bad initial solution from the roadgraph generation. With the used problem formulation, maneuvering is not possible, also the optimization might fail or get stuck in undesired local minima. Executing the graph-search-based Hybrid A* planner in these cases resolves the problem by planning a path with high deviation from the roadgraph path.

VIII. Conclusion and Future Work

In this work, we introduce a Voronoi-based and a circle-based exploration method for geometrically exploring the maneuvering free-space and a topological roadgraph network. We use this for navigation, decision making and path planning in an environment with unknown structure. We also compare a Hybrid A*-based and an optimization-based path planning method. Both generate smooth drivable paths along the reference track derived by geometric exploration.

Evaluations in simulation and using a research vehicle demonstrate the applicability of all four methods. The quantitative comparisons favor the Voronoi-based exploration in combination with the optimization-based path planning method in most cases. However, the Hybrid A* algorithm is valuable as a fall back mechanism to increase functional safety at an early research state. The circle-based exploration is more flexible for adaptations.

In future projects, we plan to incorporate camera information like detected lane markings or maps with low accuracy into the approach. We also aim to incorporate the estimated motion of dynamic obstacles into the planning approach.

To prove the universal real time applicability of the presented algorithms a decent qualitative runtime evaluation in a broader set of scenarios and various vehicle speeds will be conducted.

References

[1] T. Kubertschak, M. Maehlisch, and H.-J. Wuensche, “Towards a unified architecture for mapping static environments,” *17th International Conference on Information Fusion (FUSION)*, pp. 1–8, 2014.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] C. Chen, M. Rickert, and A. Knoll, “Path planning with orientation-aware space exploration guided heuristic search for autonomous parking and maneuvering,” *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2015-Augus, pp. 1148–1153, 2015.

[4] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving,” *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, pp. 32–37, 2008.

[5] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path planning for autonomous vehicles in unknown semi-structured environments,” *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[6] G. Tanzmeister, M. Friedl, D. Wollherr, and M. Buss, “Path planning on grid maps with unknown goal poses,” in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, IEEE, Oct. 2013, pp. 430–435.

[7] B. Li and Z. Shao, “A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles,” *Knowledge-Based Systems*, vol. 86, pp. 11–20, 2015.

[8] A. Bar 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.

[9] S. Hänisch, R. H. Evangelio, H. H. Tadjine, and M. Patzold, “Free-space detection with fish-eye cameras,” in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 135–140.

[10] W. Sanberg, G. Dubbleman, and P. de With, “Free-space detection with self-supervised and online trained fully convolutional networks,” *Electronic Imaging*, vol. 2017, no. 19, pp. 54–61, 2017. arXiv: `1604.02316`.

[11] K. Na, B. Park, and B. Seo, “Drivable space expansion from the ground base for complex structured roads,” *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings*, pp. 373–378, 2017.

[12] Q. Li, L. Chen, M. Li, S. L. Shaw, and A. Nüchter, “A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios,” *IEEE Transactions on Vehicular Technology*, vol. 63, no. 2, pp. 540–555, 2014.

[13] Z. Liu, S. Yu, X. Wang, and N. Zheng, “Detecting drivable area for self-driving cars: an unsupervised approach,” 2017. arXiv: `1705.00451`.

[14] *Boost c++ libraries*, `http://www.boost.org`, 2017.

[15] D. Lenz, T. Kessler, and A. Knoll, “Stochastic model predictive controller with chance constraints for comfortable and safe driving behavior of autonomous vehicles,” in *IEEE Intelligent Vehicles Symposium (IV)*, 2015.

[16] S. G. Johnson, *The nlopt nonlinear-optimization package*, `http://ab-initio.mit.edu/nlopt`, 2017.

[17] M. Werling, L. Gröll, and G. Bretthauer, “Invariant trajectory tracking with a full-size autonomous road vehicle,” *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 758–765, 2010.