IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Voronoi Boundary Visibility for Efficient Path Planning

MOHAMMED RABEEA HASHIM AL-DAHHAN[1], KLAUS WERNER SCHMIDT[2]

[1]Department of Electronic and Communication Engineering, Çankaya University, 06790 Ankara, Turkey (e-mail:alanymohammed5@gmail.com)
[2]Department of Electrical and Electronics Engineering, Middle East Technical University, 06800 Ankara, Turkey (e-mail: schmidt@metu.edu.tr)

Corresponding author: Klaus Werner Schmidt (e-mail: schmidt@metu.edu.tr).

ABSTRACT The subject of this paper is the computation of paths for mobile robots that navigate from a start position to a goal position in environments with static obstacles. Specifically, we focus on paths that are represented by straight lines. Such paths can for example directly be followed by omni-directional robots or can be used as an initial solution for path smoothing. In this context, the most common performance metrics are the path length, the obstacle clearance and the computation time. In this paper, we develop a new path planning algorithm that addresses all the stated performance metrics. Our method first determines all possible connections between the start position and goal position along the edges of the generalized Voronoi diagram (GVD) of a given obstacle map. The shortest connections are then refined using a balanced method for creating shortcuts along existing waypoints and introducing new waypoints in order to cut corners. As an important feature, our method reduces the number of required waypoints by iteratively adding new waypoints and then removing unnecessary waypoints along solution paths. Moreover, our method takes into account multiple start-goal connections, since the shortest start-goal connection along the edges of the GVD might not lead to the shortest solution path. A comprehensive computational evaluation for a large number of maps with different properties shows that the proposed method outperforms sampling-based algorithms such as Probabilistic Roadmaps (PRM) and exact methods such as Visibility Graphs (VG) by computing close-to-optimal solution paths with a specified minimum obstacle clearance in less time.

INDEX TERMS Mobile robots, path planning, obstacles, Voronoi diagram, safety.

I. INTRODUCTION

NOWADAYS, with the vast developments in technology, autonomous and mobile robots [1] are employed in many application fields such as autonomous driving [2], [3], navigation in complex environments [4]–[6], vehicle routing [7], buildings [8], industrial automation [9] and unmanned air vehicles [10], [11]. Depending on the application and the task to be solved, robots have to navigate in different types of environments without colliding with obstacles. In static environments, the location of obstacles is known [4], [12], [13], whereas obstacles can change their location in dynamic environments [14], [15]. When applying methods of computational geometry, maps represent objects such as obstacles in the form of geometrical shapes such as lines, polygons or circular shapes [16], [17]. On the other hand it is possible to employ binary images, where the free space and obstacles are represented by white and black pixels, respectively [18]–[20].

As an important application, path planning for mobile robots has attracted much attention in the recent years [12], [15], [21]. Path planning is concerned with finding a feasible robot path between a start and goal position, while avoiding obstacles in the robot environment [22], [23]. Hereby, the most common performance metrics to validate the quality of solution paths are the path length, the distance (clearance) to obstacles, which relates to path safety, and the computation time [16], [24].

There are different possible scenarios for robotic path planning depending on the availability of information about the environment [13], [25]–[27], the type of obstacles (static or dynamic) [15] and the robot type [21], [23], [28]. In this paper, we focus on the path planning in static environments, where robot paths are represented by straight-line segments. Such paths are for example suitable for omni-directional robots [28] or can be used as initial solutions when applying path smoothing [29], [30].

The most important performance metric in robotic path planning is the path length. In the recent literature, sampling-

based algorithms based on Probabilistic Roadmaps (PRM) or Rapidly exploring Random Trees (RRT) are most popular for path planning in known environments with static obstacles [13], [24]–[27], [31]. On the one hand, the PRM algorithm generates random sample nodes and introduces connections between close nodes in the obstacle-free region to determine a solution path [25]. On the other hand, the RRT algorithm is based on the idea of growing a tree in the obstacle-free region from the start position to the goal position [26]. Although these basic sampling-based algorithms generally cannot guarantee finding (optimal) solutions, their extensions such as the PRM* algorithm and the RRT* algorithm in [13] ensure convergence to an optimal path if the number of samples increases. Moreover, the Fast Marching Tree (FMT) algorithm in [27] combines features of PRMs and RRTs to determine shorter solution paths. Finding an optimal path is guaranteed when using the Visibility Graph (VG) for polygon maps [32]. Nevertheless, the VG cannot be directly computed for maps with general obstacle shapes and its computation is inefficient in practice without additional customized data structures [33].

A second important performance metric in robotic path planning is path safety in order to avoid collisions [24], to account for possible uncertainties during path following [20] and to facilitate re-planning in dynamic environments with moving objects [15]. Regarding this performance metric, two different objectives are considered in the literature. On the one hand, there are several methods aiming at the computation of short solution paths while maximizing the distance to obstacles [8], [18], [24], [34], [35]. On the other hand, various methods focus on minimizing the path length while guaranteeing a specified minimum distance of solution paths to obstacles [15], [16], [20], [36]. Methods addressing the first objective are frequently based on the generalized Voronoi diagram (GVD) [34], [37], which partitions an environment into Voronoi regions of points that are closest to an obstacle. Then, the Voronoi boundary (VB) represents the border of the Voronoi regions such that each point on the VB has a maximum possible distance to obstacles. Since solution paths along the VB can be unnecessarily long [34], several methods suggest to define a safe region for robot navigation around the VB for solution paths. The work in [34] inflates the VB depending on the maximum distance to obstacles at each point of the VB. Hereby, the degree of inflation is specified by a parameter that provides a tradeoff between path length and path safety. Differently, [8], [18], [35] inflate the VB by a fixed amount. [18] applies the fast marching method on this inflated VB, [8] proposes a skilled-RRT method that generates samples along the inflated VB and [35] generates short and safe solution paths by reducing the sampling space for the PRM* and FMT algorithms to the inflated VB. Sampling-based safe path planning with the aim of maximizing the obstacle clearance is as well addressed by the Confidence Random Tree (CRT) algorithm [24], which selects samples based on the clearance from obstacles (denoted as confidence) but without computing the VB. Regarding the second objective, [36] proposes the visibility-Voronoi complex for polygon maps in order to generate short solution paths with a minimum obstacle clearance whenever possible. [38] suggests to first refine the shortest path along the VB by removing unnecessary turns and then introduces additional points in order to shorten the solution path. Hereby, modifications of the path are only accepted if the obstacle clearance stays above a given value. A minimum obstacle clearance is ensured in [15], [20] by inflating the obstacle region using morphological dilation.

Computing robot paths quickly is of high importance especially in the case of real-time and dynamic path planning. Here, it is required to find a solution path within a few seconds after determining or updating the map of the robot environment. In this context, the Quick RRT* (Q-RRT*) algorithm in [31] speeds up the convergence of the RRT* algorithm by avoiding small turns and by employing informed-RRT* to explore narrow passages. Similarly, the synchronized biased-greedy RRT algorithm in [39] directly grows trees towards the goal location and the meta-algorithm in [40] speeds up the exploration phase of the RRT algorithm in order to find a high-quality initial solution path fast. A modification of the sampling strategy of the PRM algorithm is proposed in [41]. Small local roadmaps are generated based on the obstacle boundaries in difficult parts of the environment and the number of connections is reduced based on the size of the local roadmaps. The comparison of different path planning algorithms in [42] further indicates that the PRM algorithm is the best of the studied algorithms in finding short paths for real-time path planning. As an alternative methods such as [37] speed up the search process for a solution path by guiding the search along the VB.

The main motivation of this paper is the development of a path planning method that addresses all the performance metrics discussed above. In particular, we aim at the fast computation of close-to-optimal solution paths that ensure a given minimum obstacle clearance for maps with general obstacle shapes. In this context, it has to be noted that the stated performance metrics are in principle conflicting. That is, obtaining a shorter path generally leads to unsafe paths and a longer computation time, whereas solution methods with small computation times will generally produce longer robot paths.

Apart from this fact, it has to be pointed out that the path planning problem for mobile robots is performed either in 3D or 2D space. In particular, when considering nonholonomic mobile robots (that cannot arbitrarily move in any direction), it is important to take into account the robot position (x- and y-direction) as well as the heading angle when planning robot paths. However, when planning paths for robots that can move in any direction (such as omni-directional robots), straight-line paths that only depend on waypoints in 2D-space are suitable. More importantly, such straight-line paths can as well be used as initial solutions for generating smooth paths for nonholonomic mobile robots.

In this context, path planning methods for higher-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

dimensional spaces such as PRM* or RRT* are not as beneficial for the low-dimensional spaces of mobile robots. Although these methods are probabilistically complete (that is, the probability that the planner fails to return a solution decays to zero as the number of samples approaches infinity) and asymptotically optimal (the solution path length converges almost surely to the shortest path length) [13], they do not guarantee finding a short path when putting restrictions on the computation time. Specifically, these methods do not make use of the topology of the robot environment. Differently, the GVD captures the topology of the robot environment in the sense that the VB represents points in the robot environment with a maximum distance from obstacles. In particular, it is possible to follow the VB from the start position to a desired goal position of a mobile robot. That is, the existence of a solution path is guaranteed if there is a connection along the VB. Nevertheless, such solution path is generally unnecessarily long since it includes many turns to keep a maximum distance from obstacles [16].

In view of the above discussion, the main contribution of this paper is the efficient usage of the topology of the robot environment based on the GVD in order to generate short solution paths for mobile robots while ensuring a specified minimum obstacle clearance and requiring a small computation time. To this end, the paper develops a new method for the iterative refinement of initial solution paths along the VB. First, our method constructs a graph that captures the topology of the robot environment based on the GVD. The shortest paths in this graph along the VB are used as initial solutions. These initial solutions are then refined by iteratively applying two main steps. On the one hand, unnecessary waypoints are removed if it is possible to introduce an obstacle-free shortcut. On the other hand, additional waypoints are introduced in order to cut corners that are generated by the edges between waypoints. As the main difference to existing work such as [16], our method repeatedly adds and removes waypoints such that the total number of waypoints always remains small. This allows speeding up the solution process while obtaining shorter paths. In particular, our method is able to closely approximate solution paths based on the VG for polygon maps. In addition, our path planning algorithm supports the computation of safe paths with a minimum obstacle clearance by inflating obstacles. In order to evaluate our method, we perform a comprehensive comparison with existing state-of-the-art methods regarding path length, path safety and computation time.

The remainder of the paper is organized as follows. Section II introduces the required notation and gives the relevant background information on path planning for mobile robots. The proposed method is developed in Section III and its features are illustrated by a simple example environment. In Section IV we perform a comprehensive evaluation of the proposed algorithm with a large number of maps with different properties. Section V gives conclusions and ideas for future work.

II. BACKGROUND

A. NOTATION

The subject of this paper is the path planning for mobile robots in two-dimensional (2D) static environments with obstacles. Hereby, we focus on the generation of paths that consist of straight-line segments. Such paths can for example be followed by omni-directional robots, which are able to turn on the spot [28], [43], [44] or can be used as a starting point for generating smooth robot paths [29], [30]. Formally, the configuration space is defined as $\mathcal{C} \in \mathbb{R}^2$ and obstacles in $\mathcal{C}$ are represented by the obstacle region $\mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$. Accordingly, the obstacle-free region that is available for the robot motion is determined as $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$. Fig. 1 shows an illustration of the previously defined regions with an obstacle region that consists of three circular obstacles that should not be hit by the mobile robot.

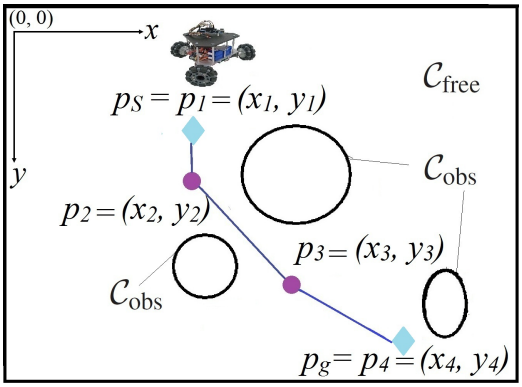

FIGURE 1: Example robot environment.

Any point $p \in \mathcal{C} \subseteq \mathbb{R}^2$ can be represented by its coordinates x and y. That is, we write $p = (x, y)$ as shown in Fig.1. Considering that we are interested in straight-line paths in this paper, a robot path P is defined by a sequence of n points $p_1, p_2, \ldots, p_n$ in $\mathcal{C}$. Hence, we write $P = (p_1, p_2, \ldots, p_n)$, whereby $p_i \in \mathcal{C}$ for $i = 1, \ldots, n$. Defining the start and goal point of a mobile robot as p_{s} and p_{g}, respectively, it must hold that $p_1 = p_{\text{s}}$ and $p_n = p_{\text{g}}$ for any suitable robot path. The actual robot path is then determined by connecting subsequent points p_i and p_{i+1} of a path P by straight lines $l_{p_i,p_{i+1}}$ for each $i = 1, \ldots, n-1$. Then, the set of points traversed by the robot from p_{s} to p_{g} is given by $\mathcal{P}_P \subseteq \mathcal{C}$, whereby $\mathcal{P}_P$ consists of all the points that are covered by the line segments $l_{p_1,p_2}, \ldots, l_{p_{n-1},p_n}$. Accordingly, we denote a path P as collision-free if $\mathcal{P}_P \cap \mathcal{C}_{\text{obs}} = \emptyset$, that is, there is no intersection of the points covered by the path and the obstacle region. For later use in different algorithms, we also introduce the function

$$\texttt{CollisionFree}(p, \hat{p}) = \begin{cases} \textbf{true} & \text{if } l_{p,\hat{p}} \cap \mathcal{C}_{\text{obs}} = \emptyset \\ \textbf{false} & \text{otherwise} \end{cases}$$

that determines if the straight-line connection $l_{p,\hat{p}}$ between two points p and $\hat{p}$ intersects the obstacle region $\mathcal{C}_{\text{obs}}$. We write

$$\mathcal{A} = \{P | \mathcal{P}_P \subseteq \mathcal{C}_{\text{free}}\} \tag{1}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

for the set of obstacle-free paths. We further introduce the distance between two points $p_i = (x_i, y_i), p_j = (x_j, y_j) \in \mathcal{C}$ as

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{2}$$

the minimum distance between a point $p \in \mathcal{C}$ and a subset $\mathcal{C}' \subseteq \mathcal{C}$ as

$$d(p, \mathcal{C}') = \min_{p' \in \mathcal{C}'} d(p, p') \tag{3}$$

and the minimum distance between two subsets $\mathcal{C}', \mathcal{C}'' \subseteq \mathcal{C}$ as

$$d(\mathcal{C}', \mathcal{C}'') = \min_{p' \in \mathcal{C}', p'' \in \mathcal{C}''} d(p', p''). \tag{4}$$

Using (4), the minimum distance of a path P from the obstacle region can be written as $d(\mathcal{P}_P, \mathcal{C}_{\text{obs}})$. Finally, we compute the path length of P as

$$L(P) = \sum_{i=1}^{n-1} d(p_i, p_{i+1}). \tag{5}$$

Using the notation introduced above, the main aim of this paper is the computation of suitable obstacle-free robot paths between a given start point $p_{\text{s}} \in \mathcal{C}_{\text{free}}$ and goal point $p_{\text{g}} \in \mathcal{C}_{\text{free}}$. In this context, we characterize suitability of robot paths by performance metrics such as the path length (finding the shortest path), path safety (finding a path with a certain distance specification with respect to obstacles) and the computation time.

B. GENERALIZED VORONOI DIAGRAM

The results presented in this paper are based on the usage of the generalized Voronoi diagram (GVD), which is a basic data structure in robotic path planning [16], [34], [37]. In order to formalize the related terminology, we consider a configuration space $\mathcal{C}$ that contains a set of geometric objects $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m$ such that $\mathcal{O}_i \subseteq \mathcal{C}$ for $i = 1, \ldots, m$ as illustrated in Fig. 2.

Each object $\mathcal{O}_i$ is associated to a Voronoi region $\mathcal{V}_i$. Specifically, $\mathcal{V}_i$ defines the set of all points $p \in \mathcal{C}$ that are closer to $\mathcal{O}_i$ than to any other object $\mathcal{O}_j$ with $i \neq j$ [34], [37]. Formally, we define

$$\mathcal{V}_i = \{p \in \mathcal{C} | d(p, \mathcal{O}_i) \leq d(p, \mathcal{O}_j), \forall j \neq i\}. \tag{6}$$

Using (6), the collection of all regions $\mathcal{V}_1, \ldots, \mathcal{V}_m$ is denoted as the generalized Voronoi diagram (GVD). As can be seen in Fig. 2, adjacent Voronoi regions share a border that consists of all points with an equal distance to at least two objects. We write $\mathcal{V}$ for the set of all such points and we call $\mathcal{V}$ the Voronoi boundary (VB). Formally,

$$\mathcal{V} = \{p \in \mathcal{C} | \exists i \neq j, d(p, \mathcal{O}_i) = d(p, \mathcal{O}_j)\}. \tag{7}$$

That is, for each point $p \in \mathcal{V}$, there are at least two different objects $\mathcal{O}_i, \mathcal{O}_j$ with $i \neq j$ with an equal distance to p. In particular, any robot path following the VB keeps a maximum distance from the objects $\mathcal{O}_i$, $i = 1, \ldots, m$, which can be identified as obstacles. In addition, we denote points, where multiple borders of the VB meet as branching points (BPs). To this end, we define the set of BPs $\mathcal{B}$ as the set of all points on the boundary of at least 3 Voronoi regions.

$$\begin{aligned}\mathcal{B} = \{&p \in \mathcal{V} | \exists i, j, k, i \neq j \neq k, \\ &d(p, \mathcal{O}_i) = d(p, \mathcal{O}_j) = d(p, \mathcal{O}_k)\}.\end{aligned} \tag{8}$$

Finally, we denote parts of the VB between BPs as segments of the VB. Considering two BPs $b_i, b_j \in \mathcal{B}$, we write $\mathcal{P}_{b_i, b_j} \subseteq \mathcal{V}$ for the points covered by the segment between b_i and b_j and consider $|\mathcal{P}_{b_i, b_j}|$ as the length of the segment.

There are various studies for the computation of the GVD in the existing literature [16], [34], [45]. The underlying assumption in this paper is that the robot environment is represented by a binary image. That is, the relevant regions $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$ are not identified by geometric objects but by the pixel color such that obstacle pixels are black and the free space is characterized by white pixels as demonstrated in Fig. 2 (a). We note that the focus of this paper is not the computation of GVDs. Hence, we employ the fact that GVDs can be obtained based on the medial axis transform [46]. Accordingly, we use the morphological operation of "skeletonization" as in [47], [48] to determine an approximation of the VB that consists of the image pixels on the medial axis with an equal minimum distance to obstacle pixels (see (7)). Then, BPs are determined as image pixels that are connected to at least three segments of the VB.

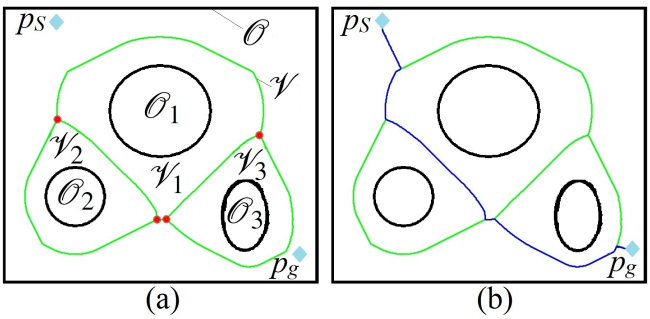

FIGURE 2: Example environment: (a) VB $\mathcal{V}$; (b) Shortest connection of p_{s} and p_{g} to $\mathcal{V}$.

C. VORONOI DIAGRAM AND SOLUTION PATH COMPUTATION

The VB $\mathcal{V}$ of the GVD can be used to determine robot paths with a maximum distance/clearance from the obstacles [15], [16], [34], [36]. Hereby, given a start point p_{s} and a goal point p_{g}, it is first required to connect these points to $\mathcal{V}$. Generally, such connection is achieved by computing the shortest collision-free path from p_{s} and p_{g} to $\mathcal{V}$ [16], [37] and extending the GVD by these paths. Then, Dijkstra's algorithm [49] or the A* search algorithm [50] can be applied to determine the shortest path between p_{s} and p_{g} in this extended GVD as depicted in Fig. 2 (b). In this paper, we employ A* search since it commonly leads to smaller computation times compared to Dijkstra's algorithm by guiding the search.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

III. VORONOI BOUNDARY VISIBILITY FOR EFFICIENT PATH PLANNING

In this section, we develop our proposed method for the fast computation of short solution paths using information from the GVD as described in Section II-B. Section III-A introduces a new method for connecting the start/goal point to the GVD and then defines a graph that captures the connectivity of the VB $\mathcal{V}$ using the BPs in $\mathcal{B}$. Then, Section III-B introduces the VB Visibility (VV) algorithm for determining short paths along the VB and Section III-C develops a new method for the iterative refinement of solution paths. Finally, Section III-E discusses how this method can be extended in case a minimum obstacle clearance is specified for path safety.

A. CONNECTION OF THE START AND GOAL POINT

As described in Section II-C, one way of connecting p_s and p_g to the VB is finding the shortest collision-free path from $p_\mathrm{s}/p_\mathrm{g}$ to $\mathcal{V}$. In this section, we propose an alternative method that leads to shorter solution paths. In particular, for p_s, we determine all points on the VB that have an equal distance to p_s and to the obstacle region $\mathcal{C}_\mathrm{obs}$. Formally, we determine the set $\mathcal{V}_\mathrm{s} \subseteq \mathcal{V}$ such that

$$\forall p_j \in \mathcal{V}_\mathrm{s}, d(p_j, p_\mathrm{s}) = d(p_j, \mathcal{C}_\mathrm{obs}). \tag{9}$$

Similarly, for p_g, we compute the set $\mathcal{V}_\mathrm{g} \subseteq \mathcal{V}$ such that

$$\forall p_j \in \mathcal{V}_\mathrm{g}, d(p_j, p_\mathrm{g}) = d(p_j, \mathcal{C}_\mathrm{obs}). \tag{10}$$

These points are guaranteed to have a collision-free connection to p_s and p_g, respectively. Moreover, these points can be computed efficiently based on an image with a modified obstacle region $\hat{\mathcal{C}}_\mathrm{obs} = \mathcal{C}_\mathrm{obs} \cup \{p_\mathrm{s}, p_\mathrm{g}\}$. To this end, we follow the procedure described in Algorithm 1.

Algorithm 1 Connection of the start and goal point

1: **Input:** $\mathcal{C}$, $\mathcal{C}_\mathrm{obs}$, p_s, p_g
2: **Output:** $\mathcal{V}$, $\mathcal{B}$
3: **Initialize:** $\hat{\mathcal{C}}_\mathrm{obs} = \mathcal{C}_\mathrm{obs} \cup \{p_\mathrm{s}, p_\mathrm{g}\}$
4: Determine the VB $\mathcal{V}$ for $\mathcal{C}$ and $\hat{\mathcal{C}}_\mathrm{obs}$
5: Determine all BPs $\mathcal{B}$ on the VB
6: Determine $\mathcal{V}_\mathrm{s} \subseteq \mathcal{B}$ as the BPs on the circle around p_s
7: Determine $\mathcal{V}_\mathrm{g} \subseteq \mathcal{B}$ as the BPs on the circle around p_g
8: Connect $p_\mathrm{s}/p_\mathrm{g}$ to the respective BPs in $\mathcal{V}_\mathrm{s}/\mathcal{V}_\mathrm{g}$
9: Remove all segments of $\mathcal{V}$ on the circles around $p_\mathrm{s}/p_\mathrm{g}$

That is, the proposed algorithm first determines the GVD for the image with the modified obstacle region $\hat{\mathcal{C}}_\mathrm{obs} = \mathcal{C}_\mathrm{obs} \cup \{p_\mathrm{s}, p_\mathrm{g}\}$ (line 3) with the VB $\mathcal{V}$. This GVD has the property that the start point p_s and the goal point p_g are encircled by the VB as illustrated in Fig. 3 (a). Then, the algorithm determines all BPs on $\mathcal{V}$ (line 5). The BPs on the circle around p_s are identified as $\mathcal{V}_\mathrm{s}$ (line 6), whereas the BPs on the circle around p_g belong to $\mathcal{V}_g$ (line 7) as can be seen in Fig. 3 (b). Then, the points $p_\mathrm{s}/p_\mathrm{g}$ are connected to the corresponding BPs in $\mathcal{V}_\mathrm{s}/\mathcal{V}_\mathrm{g}$ (line 8). This procedure is depicted in Fig. 3 (c). Finally, the unnecessary parts of the circles around p_s and p_g are removed (line 9) to obtain the resulting VB $\mathcal{V}$ in Fig. 3 (d).

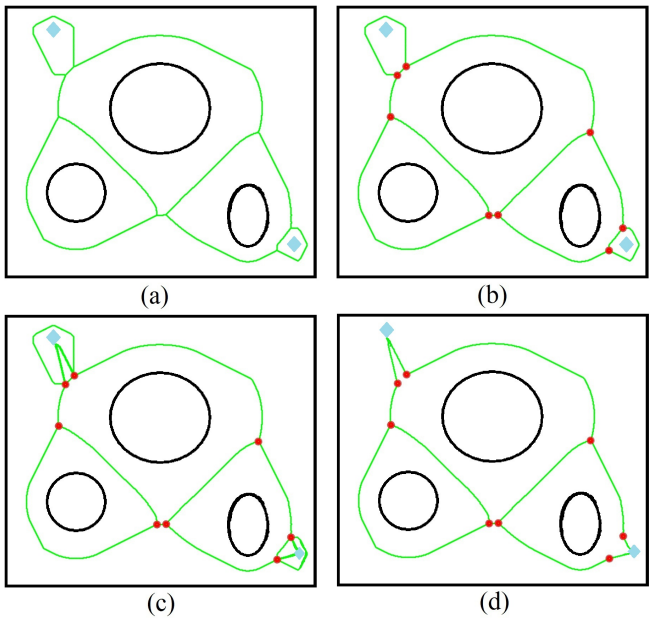

FIGURE 3: Connection of p_s and p_g to $\mathcal{V}$: (a) GVD; (b) BPs; (c) Connection to BPs on $\mathcal{V}_\mathrm{s}$ and $\mathcal{V}_\mathrm{g}$; (d) Resulting $\mathcal{V}$.

Using the outputs $\mathcal{V}$, $\mathcal{B}$ of Algorithm 1, we next compute a graph $G = (V, E)$ that characterizes the connectivity of the VB. That is, the vertexes of G are defined as $V = \mathcal{B}$ and the edges E contain all (unordered) pairs $\{b_i, b_j\}$ with $b_i, b_j \in V$ such that there is a direct connection between b_i and b_j on $\mathcal{V}$. We further take into account the special case, where different connections between a pair of vertexes b_i, b_j exist on $\mathcal{V}$. This case can for example be seen in Fig. 4. Here, there are two connections between the BPs b_1 and b_4. In order to resolve this ambiguity, we simply insert an artificial BP on one of the connections (for example the longer one as in Fig. 4 (b)) and update the edges accordingly.

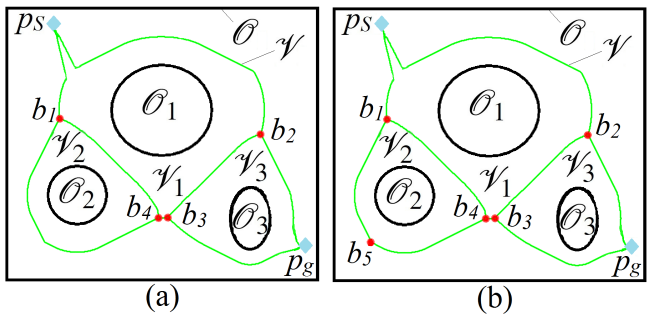

FIGURE 4: (a) Original GVD with BPs; (b) Artificial BP b_5.

The graph for the example environment in Fig. 4 is given by the vertexes $V = \{p_\mathrm{s}, p_\mathrm{g}, b_1, b_2, b_3, b_4, b_5\}$ and the edges $E = \{\{p_\mathrm{s}, b_1\}, \{p_\mathrm{s}, b_2\}, \{b_1, b_4\}, \{b_1, b_5\}, \{b_2, b_3\}, \{b_2, p_\mathrm{g}\}, \{b_3, p_\mathrm{g}\}, \{b_3, b_4\}, \{b_4, b_5\}\}$. We finally label each edge $\{b_i, b_j\} \in E$ by the path length $|\mathcal{P}_{b_i,b_j}|$ on $\mathcal{V}$ between the BPs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

b_i and b_j. The labeled graph for the example environment is shown in Fig. 5.

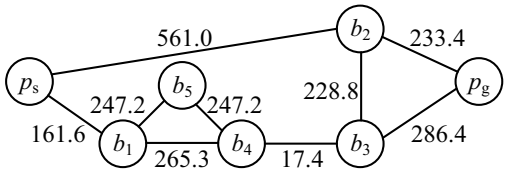

FIGURE 5: Graph for the example environment.

For the graph $G = (V, E)$, we define a walk in G by a finite sequence of vertexes $(v_1, v_2, \ldots, v_n)$ with $v_1, \ldots, v_n \in V$ such that each edge $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, n-1$. In particular, we are interested in walks from p_s to p_g, that is, $v_1 = p_\mathrm{s}$ and $v_n = p_\mathrm{g}$. In order to avoid confusion with the notion of a "path" in the robot environment, we use the notion of a start-goal walk for a walk from p_s to p_g in G. Using G, it is possible to determine the shortest start-goal walk along $\mathcal{V}$ using the A* search algorithm [50] (we denote the overall algorithm as VA* in the sequel). Moreover, it is possible to compute the k shortest (cycle-free) start-goal walks using Yen's algorithm [51]. Accordingly, we introduce the notation $W_1, \ldots, W_k$ to denote the k shortest (collision-free) start-goal walks from p_s to p_g along $\mathcal{V}$. We further note that the maximum number of different shortest start-goal walks depends on the topology of each environment and can also be determined by Yen's algorithm. For the example environment, there are 6 different start-goal walks, which are given in Table 1.

TABLE 1: $k = 6$ shortest start-goal walks for the example

Path	Length [px]
$W_1 = (p_\mathrm{s}, b_1, b_4, b_3, p_\mathrm{g})$	730.2
$W_2 = (p_\mathrm{s}, b_2, p_\mathrm{g})$	794.4
$W_3 = (p_\mathrm{s}, b_1, b_4, b_3, b_2, p_\mathrm{g})$	906.0
$W_4 = (p_\mathrm{s}, b_1, b_5, b_4, b_3, p_\mathrm{g})$	959.3
$W_5 = (p_\mathrm{s}, b_2, b_3, p_\mathrm{g})$	1076.2
$W_6 = (p_\mathrm{s}, b_1, b_5, b_4, b_3, b_2, p_\mathrm{g})$	1135.1

For each edge $\{v_i, v_{i+1}\} \in E$, we write $\mathcal{P}_{v_i, v_{i+1}} \subseteq \mathcal{V}$ for the corresponding path between the points $v_i, v_{i+1} \in V$ (recall that $V = \mathcal{B}$). Then, the path $\mathcal{P}_{W_j} \subseteq \mathcal{V}$ that corresponds to a start-goal walk $W_j = (v_1, \ldots, v_n)$, is given by the concatenation of the paths $\mathcal{P}_{v_1, v_2}, \ldots, \mathcal{P}_{v_{n-1}, v_n}$ (note that $v_1 = p_\mathrm{s}$ and $v_n = p_\mathrm{g}$). Accordingly, any path $\mathcal{P}_{W_j}$, $j = 1, \ldots, k$, constitutes a solution of the path planning problem and there is no solution of the path planning problem if $k = 0$.

B. VORONOI BOUNDARY VISIBILITY ALGORITHM

It is a well-known fact that robot paths following the VB $\mathcal{V}$ take unnecessary turns and are hence comparably long [16], [18], [34]. Accordingly, we suggest to first employ a shortcut heuristic to reduce the path length. Our method uses the information obtained from the VB $\mathcal{V}$ and the corresponding graph G in Section III-A to determine a collision-free path $P_\mathrm{VV} = (p_1, \ldots, p_{|P_\mathrm{VV}|})$ such that $\{p_1, \ldots, p_{|P_\mathrm{VV}|}\} \subseteq \mathcal{V}$.

Since our algorithm is based on the visibility of points on $\mathcal{V}$, we denote it as VB Visibility (VV) algorithm. We next explain the VV algorithm following the pseudo-code given in Algorithm 2. It is based on an initial solution path $P = (p_1, \ldots, p_{|P|})$ that is for example obtained from a start-goal walk W_j as in Section III-A and the obstacle region $\mathcal{C}_\mathrm{obs}$ (line 1). The algorithm then determines two different sequences of waypoints to be followed. In the first case ($l = 1$ in line 3), the algorithm follows the given path P from p_s to p_g. In the second case ($l = 2$), the waypoints of P are ordered in the reverse direction from p_g to p_s (line 5). P_VV is initialized with p_1, which is equal to p_s for $l = 1$ and to p_g for $l = 2$ (line 5). Moreover, the points p_cur and p_last keep track of the current point to be explored and the last collision-free connection point, respectively. The algorithm loops over all points in P (line 6). It is then checked if the straight-line connection from p_cur to p_i is collision-free (flag $=$ **true** in line 7). If flag is **true**, p_last is updated since there is no collision on the straight-line connection from p_cur to p_i (line 9). If flag is **false**, it holds that the straight-line connection from p_cur to p_i intersects with the obstacle region $\mathcal{C}_\mathrm{obs}$ (line 10). Nevertheless, we know from the previous iteration (where flag must have been **true**) that the straight-line connection from p_cur to p_last is collision-free. Hence, we accept p_last on the solution path (line 11) and restart the search for a collision-free connection from p_last (line 12 and 13). The search for new points terminates if a connection to the goal point p_g is found (line 14). The algorithm finally compares the two solution paths P_1 (for $l = 1$) and P_2 (for $l = 2$) and returns the shorter one as the result. In the sequel, we denote the path P_VV resulting from Algorithm 2 as a VV-path.

Algorithm 2 `ComputeVV`$(P, \mathcal{C}_\mathrm{obs})$

1: **Input:** $P, \mathcal{C}_\mathrm{obs}$
2: **Output:** P_VV
3: **for** $l = 1, 2$ **do**
4: **if** $l = 2$ **then**
5: $\forall i = 1, \ldots, |P|$: $p_i = p_{|P|-i+1}$
Initialize: $P_l = (p_1)$; $p_\mathrm{cur} = p_1$; $p_\mathrm{last} = p_1$
6: **for** $k = 2, \ldots, |P|1$ **do**
7: flag $=$ `CollisionFree`(p_cur, p_i)
8: **if** flag $=$ **true then**
9: $p_\mathrm{last} = p_i$
10: **else**
11: $P_l = (P_l, p_\mathrm{last})$
12: $p_\mathrm{cur} = p_\mathrm{last}$
13: $k = k - 1$
14: **if** $p_\mathrm{last} = p_{|P|}$ **then**
15: **break**
16: **return** $\arg\min_{P_1, P_2}\{L(P_1), L(P_2)\}$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

For illustration, we compute the VV-paths for W_1, W_2, W_3 and W_5 in Table 1. That is, for $i = 1, 2, 3, 5$, we apply Algorithm 2 with the initial path P_{W_i}. The resulting VV-paths are shown in Fig. 6.

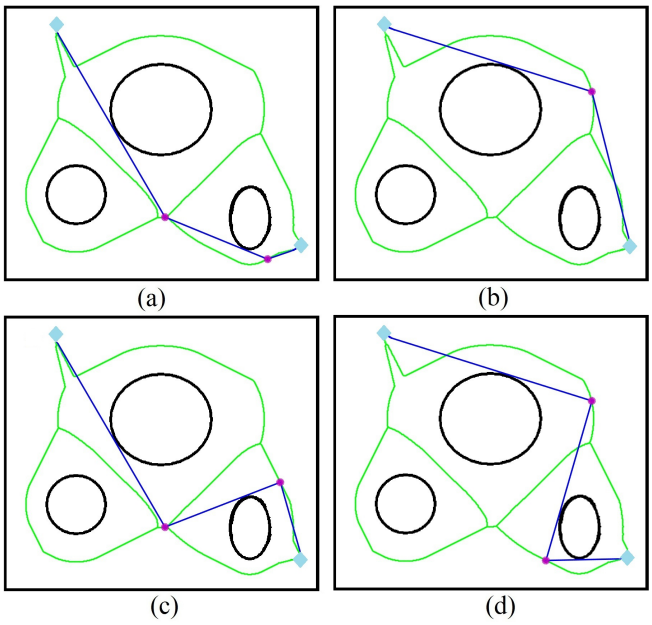

FIGURE 6: VV-path illustration for: (a) P_{W_1} with $L(P_{\text{VV}}) = 657.1$; (b) P_{W_2} with $L(P_{\text{VV}}) = 677.5$; (c) P_{W_3} with $L(P_{\text{VV}}) = 755.0$; (d) P_{W_5} with $L(P_{\text{VV}}) = 840.1$.

Remark 1: We note that shortcut heuristics with the same objective of reducing the number of waypoints and removing unnecessary turns were introduced in [16], [52]. Different from our algorithm, the algorithm in [16] suggests to iteratively remove points from a given path P if the connection between its adjacent points is collision-free. Although this method is also able to reduce the path length, we will show in Section IV that our algorithm generally leads to shorter paths. For later usage, we refer to the algorithm in [16] as the Remove Redundancy (RR) algorithm. In addition, we introduce the function `RemoveRedundancy` in the form

$$P_{\text{RR}} = \texttt{RemoveRedundancy}(P, \mathcal{C}_{\text{obs}})$$

such that P_{RR} is the solution path when applying the RR algorithm with an initial solution path P.

The algorithm in [52] also uses the idea of checking connections to waypoints until a collision is detected. Differently, that algorithm only evaluates the path from p_{s} to p_{g} but omits the path from p_{g} to p_{s}. It is obtained from Algorithm 2 by iterating only for $l = 1$ in line 3. □

For illustration, we show two paths that are obtained for the initial path P_{W_1} of the example environment. Fig. 7 (a) and (b) depict the solution paths P_{VV} and P_{RR} obtained from `ComputeVV` and `RemoveRedundancy`, respectively. It can be seen that both paths select waypoints on $\mathcal{P}_{W_1}$. Hereby, P_{VV} defines a shorter connection since `computeVV` looks ahead more.

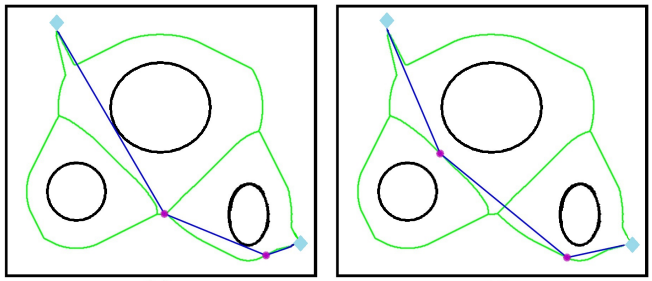

FIGURE 7: (a) P_{VV} with $L(P_{\text{VV}}) = 657.1$; (b) P_{RR} with $L(P_{\text{RR}}) = 666.2$.

C. VV WITH INTERMEDIATE POINTS

The VV-paths computed by Algorithm 2 are able to avoid unnecessary turns when following the VB as can be seen in Fig. 6. Nevertheless, it is still the case that VV-paths have unnecessary corners that increase the path length. Accordingly, we suggest to apply the technique of adding Steiner points in order to cut corners. We note that this technique was first introduced in [16]. Nevertheless, in this paper we propose a particular combination with Algorithm 2 and the graph G that both reduces the computation time and enables the reduction of the path length.

We first formalize the technique in [16] in Algorithm 3 for later usage in our improved algorithms. The algorithm is denoted as `ComputeST` since it introduces Steiner points to compute a solution path P_{S} (line 2) based on a given feasible path P that is for example obtained from Algorithm 2, the obstacle region $\mathcal{C}_{\text{obs}}$ and a distance value Δ (line 1). The solution path is initialized with the given path (line 3) and the algorithm terminates if P_{S} only contains the start and goal point (line 5). Otherwise, the algorithm repeats the following iteration (line 6 to 22): The algorithm tries to reduce the path length by looking at 3 consecutive points p_{L}, p, p_{R} (line 7) starting from $p_{\text{s}} = P_{\text{S}}[1]$. Hereby, the notation $P_{\text{S}}[i]$ denotes the i-th waypoint in P_{S} and the main idea is to remove a potential corner with p when moving from p_{L} to p_{R}. Then, the algorithm tries to put new points p_{SL} and p_{SR} between p, p_{L} and p, p_{R}. These points are generated at increasing distances $k \cdot \Delta$ from p (line 12). p_{SL} and p_{SR} are accepted as new waypoints if their straight-line connection is collision-free (line 15). No more new points can be generated if there is a collision (line 17) or the generated points do no longer lie between p, p_{L} and p, p_{R} (line 11). After completing the generation of new points, the algorithm checks if any new points could be found (line 18). If there are new points, the original point p is removed from P_{S} and replaced by $\hat{p}_{\text{SL}}$ and $\hat{p}_{\text{SR}}$ (line 19). Then, the algorithm continues from the next unvisited waypoint (line 20 or 22). If the goal point p_{g} is reached (line 9), the inner loop (line 9) terminates. If no more improvement in the solution path P_{S} can be achieved (line 6), P_{S} is returned.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

Algorithm 3 `ComputeST`$(P, \mathcal{C}_{\text{obs}}, \Delta)$

Input: Collision-free path $P = (p_1, \ldots, p_{|P|})$ such that $p_1 = p_{\text{s}}$ and $p_{|P|} = p_{\text{g}}$; $\mathcal{C}_{\text{obs}}$; Δ
Output: Solution path P_{S}
Initialize: $P_{\text{S}} = P$; $n_{\text{old}} = \infty$
if $|P_{\text{S}}| = 2$ **then**
  return P_{S}
while $|P_{\text{S}}| \neq n_{\text{old}}$ **do**
  $n_{\text{old}} = |P_{\text{S}}|$; $p = P_{\text{S}}[2]$; $p_{\text{L}} = P_{\text{S}}[1]$; $p_{\text{R}} = P_{\text{S}}[3]$
  $d_{\text{L}} = d(p_{\text{L}}, p)$; $d_{\text{R}} = d(p_{\text{R}}, p)$; $u_{\text{L}} = \frac{p_{\text{L}} - p}{d_{\text{L}}}$; $u_{\text{R}} = \frac{p_{\text{R}} - p}{d_{\text{R}}}$
  while $p \neq p_{\text{g}}$ **do**
    $\hat{p}_{\text{SL}} = p$; $\hat{p}_{\text{SR}} = p$; $k = 1$
    while $k \cdot \Delta < d_{\text{L}}$ **and** $k \cdot \Delta < d_{\text{R}}$ **do**
      $p_{\text{SL}} = p + u_{\text{L}} \cdot k \cdot \Delta$; $p_{\text{SR}} = p + u_{\text{R}} \cdot k \cdot \Delta$
      flag $=$ `CollisionFree`$(p_{\text{L}}, p_{\text{R}})$
      if flag $=$ **true then**
        $\hat{p}_{\text{SL}} = p_{\text{SL}}$; $\hat{p}_{\text{SR}} = p_{\text{SR}}$
      else
        break
    if $\hat{p}_{\text{SL}} \neq p$ **then**
      $P_{\text{S}} = (p_{\text{s}}, \ldots, p_{\text{L}}, \hat{p}_{\text{SL}}, \hat{p}_{\text{SR}}, p_{\text{R}}, \ldots, p_{\text{g}})$
      $p = \hat{p}_{\text{SL}}$
    else
      $p = p_{\text{R}}$
return P_{S}

Fig. 8 illustrates the successive application of `computeVV` in Algorithm 2 and Algorithm 3 for the initial paths P_{W_1} and P_{W_2} in Fig. 6. In both cases, it can be seem that additional waypoints are introduced in order to remove unnecessary corners in the VV-paths.

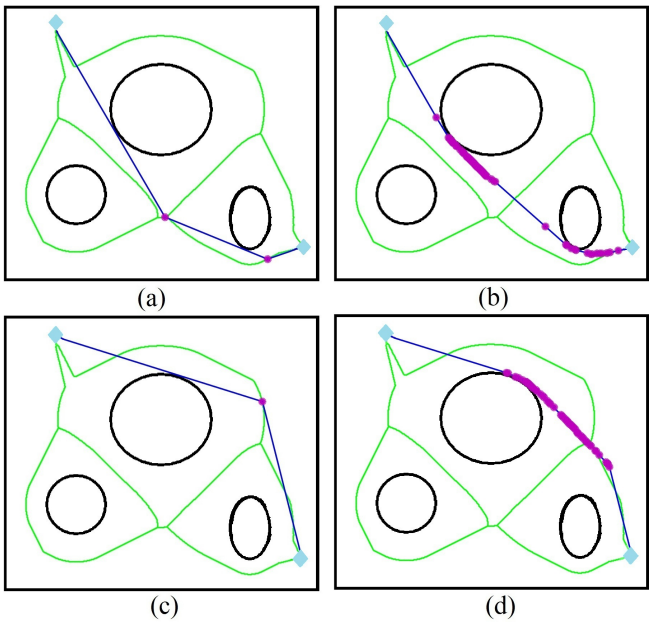

FIGURE 8: (a) Algorithm 2 for P_{W_1} with $L(P_{\text{VV}}) = 657.1$; (b) Algorithm 2 and 3 for P_{W_1} with $L(P_{\text{S}}) = 637.0$; (c) Algorithm 2 for P_{W_2} with $L(P_{\text{VV}}) = 677.5$; (d) Algorithm 2 and 3 for P_{W_2} with $L(P_{\text{S}}) = 645.4$.

D. OVERALL PATH COMPUTATION

As can be seen in Fig. 8, the length of solution paths can be reduced when applying Algorithm 2 and 3 consecutively. Nevertheless, it also has to be noted that the solution path P_{S} of Algorithm 3 can have an increased number of waypoints since it repeatedly adds new waypoints. That is, shorter connections between some of the added waypoints might be possible. Accordingly, we next define two particular combinations of Algorithm 2 and 3 for the efficient computation of short solution paths with a small number of waypoints. The first method, which constitutes the main contribution of the paper, is stated in Algorithm 4. Here, the input parameters Δ_{init} and Δ_{min} represent the initial and minimum distance value for the refinement according to Algorithm 3. The algorithm first computes the VB $\mathcal{B}$ and the set of BPs $\mathcal{B}$ according to Algorithm 1 (line 3) and then determines the k shortest paths $P_{W_1}, \ldots, P_{W_k}$ from p_{s} to p_{g} along the VB following the graph construction in Section III-A (line 4). Then, the shortest path is initialized with the shortest path P_{W_1} along the VB and the algorithm tries to reduce all the paths P_{W_k}, $i = 1, \ldots, k$ (line 6 to 14). In each iteration, the solution candidate P is initialized with P_{W_i}. The algorithm first applies the VV algorithm (line 8) and then repeatedly inserts Steiner points in P in order to remove corners (line 10). Hereby, Δ is decreased until the minimum resolution given by Δ_{min} is reached (line 9 and 12). Moreover, `ComputeVV` is repeatedly applied (line 11) in order to keep the number of points in P_{S} small. At the end of each iteration, the solution candidate is updated if the current path P is shorter than the previously found paths (line 14). The algorithm returns the shortest path P_{S} found among all the candidates (line 15). Since Algorithm 4 repeatedly adds Steiner points and applies VB visibility, it is denoted as VV-ST-R algorithm.

Algorithm 4 `VV-ST-R`$(\mathcal{C}, \mathcal{C}_{\text{obs}}, p_{\text{s}}, p_{\text{g}}, k, \Delta_{\text{init}}, \Delta_{\text{min}})$

1: **Input:** $\mathcal{C}$; $\mathcal{C}_{\text{obs}}$; p_{s}; p_{g};Δ_{init}; Δ_{min}.
2: **Output:** Solution path P_{S}
3: Compute $\mathcal{V}$ and $\mathcal{B}$ using Algorithm 1
4: Compute the k shortest paths $P_{W_1}, \ldots, P_{W_k}$ as described in Section III-A
5: **Initialize:** $P_{\text{S}} = P_{W_1}$
6: **for** $i = 1, \ldots, k$ **do**
7: $P = P_{W_1}$; $\Delta = \Delta_{\text{init}}$
8: $P =$ `ComputeVV`$(P, \mathcal{C}_{\text{obs}})$
9: **while** $\Delta \geq \Delta_{\text{min}}$ **do**
10: $P =$ `ComputeST`$(P, \mathcal{C}_{\text{obs}}, \Delta)$
11: $P =$ `ComputeVV`$(P, \mathcal{C}_{\text{obs}})$
12: $\Delta = \Delta/2$
13: **if** $L(P) < L(P_{\text{S}})$ **then**
14: $P_{\text{S}} = P$
15: **return** P_{S}

The second method is a modification of Algorithm 4 that is stated for comparison with the work in [16]. The main difference between Algorithm 4 and 5 is the usage of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

`ComputeVV`. While this function is used repeatedly in each iteration of the while loop in Algorithm 4, it is only used at the beginning and end of the computation for each shortest path P_{W_i} in Algorithm 5 (line 8 and 12). Accordingly, Algorithm 5 is denoted as VV-ST algorithm.

Algorithm 5 `VV-ST`$(\mathcal{C}, \mathcal{C}_{\text{obs}}, p_{\text{s}}, p_{\text{g}}, k, \Delta_{\text{init}}, \Delta_{\text{min}})$

1: **Input:** $\mathcal{C}$; $\mathcal{C}_{\text{obs}}$; p_{s}; p_{g};Δ_{init}; Δ_{min}.
2: **Output:** Solution path P_{S}
3: Compute $\mathcal{V}$ and $\mathcal{B}$ using Algorithm 1
4: Compute the k shortest paths $P_{W_1}, \ldots, P_{W_k}$ as described in Section III-A
5: **Initialize:** $P_{\text{S}} = P_{W_1}$
6: **for** $i = 1, \ldots, k$ **do**
7: $\quad P = P_{W_1}$; $\Delta = \Delta_{\text{init}}$
8: $\quad P = \texttt{ComputeVV}(P, \mathcal{C}_{\text{obs}})$
9: $\quad$ **while** $\Delta \geq \Delta_{\text{min}}$ **do**
10: $\quad\quad P = \texttt{ComputeST}(P, \mathcal{C}_{\text{obs}}, \Delta)$
11: $\quad\quad \Delta = \Delta/2$
12: $\quad P = \texttt{ComputeVV}(P, \mathcal{C}_{\text{obs}})$
13: $\quad$ **if** $L(P) < L(P_{\text{S}})$ **then**
14: $\quad\quad P_{\text{S}} = P$
15: **return** P_{S}

Remark 2: We note that Algorithm 5 is shown in this paper in order to evaluate the improvements of our method compared to [16]. In particular, using $k = 1$ and replacing "`ComputeVV`" by "`RemoveRedundany`" in line 8 and 12 leads to the algorithm in [16]. This algorithm is denoted as RR-ST algorithm in the sequel since it applies the RR algorithm and uses the addition of Steiner points. The comprehensive evaluation in Section IV will reveal that both Algorithm 4 and 5 lead to considerable reductions in the path length without increasing the computation time. We further note that we evaluated different methods for creating additional waypoints. For example, [53] uses the idea of Bisection in order to generate Steiner points on neighboring edges of a solution path. Nevertheless, there was no considerable effect on the final result (regarding path length and computation time) when using different methods. □

For illustration, we apply Algorithm 4 and 5 to the paths P_{W_1} and P_{W_2} of the example environment as shown in Fig. 9. Due to the repeated application of the VV algorithm in Algorithm 4, it is the case that shorter solution paths are found by this algorithm. Specifically, we get $L(P_{\text{S}}) = 633.9$, $L(P_{\text{S}}) = 635.6$, $L(P_{\text{S}}) = 631.6$ and $L(P_{\text{S}}) = 644.6$ for the solution paths in Fig. 9 (a), (b), (c) and (d), respectively. It is further interesting to note that the solution path of Algorithm 4 for the second-shortest start-goal walk W_2 along VB is shorter than the solution path for the shortest start-goal walk W_1 along VB. This demonstrates the advantage of taking into account several short start-goal walks instead of only the shortest start-goal walk.

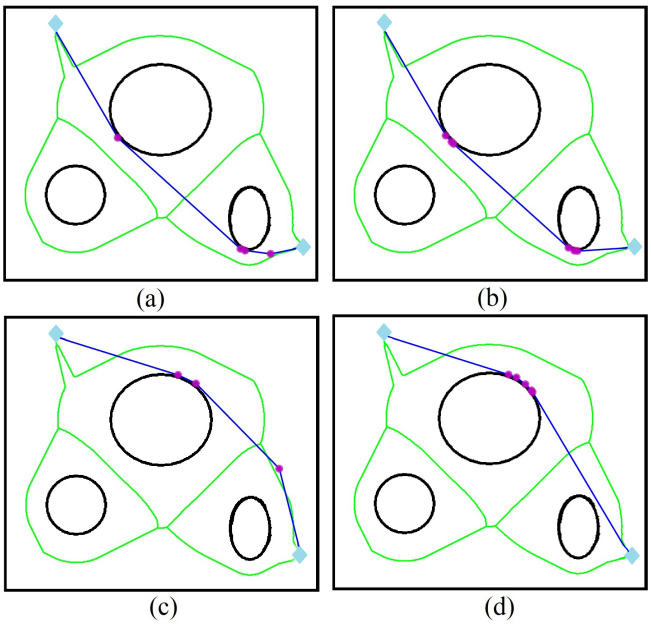

FIGURE 9: (a) Algorithm 5 for P_{W_1} with $L(P_{\text{S}}) = 635.6$; (b) Algorithm 4 for P_{W_1} with $L(P_{\text{S}}) = 633.9$; (c) Algorithm 5 for P_{W_2} with $L(P_{\text{S}}) = 644.6$; (d) Algorithm 4 for P_{W_2} with $L(P_{\text{S}}) = 631.6$.

E. INFLATED OBSTACLE REGION

In addition to finding a shortest path in a given environment, it is frequently required to determine a path that keeps a minimum safety distance D_{S} from the obstacle region $\mathcal{C}_{\text{obs}}$ [15], [16], [20], [20], [24]. That is, it is desired for the solution path P_{s} that

$$d(\mathcal{P}_{P_{\text{S}}}, \mathcal{C}_{\text{obs}}) > D_{\text{S}}. \tag{11}$$

In order to address this issue within the proposed methodology, we define the inflated obstacle region $\mathcal{C}_{\text{obs}}(D_{\text{S}})$ that contains all points in $\mathcal{C}$, whose distance from $\mathcal{C}_{\text{obs}}$ is less or equal to D_{S}:

$$\mathcal{C}_{\text{obs}}(D_{\text{S}}) = \{p \in \mathcal{C} | d(p, \mathcal{C}_{\text{obs}}) \leq D_{\text{S}}\}. \tag{12}$$

In particular, it holds that $\mathcal{C}_{\text{obs}} = \mathcal{C}_{\text{obs}}(0)$. A viable method for determining $\mathcal{C}_{\text{obs}}(D_{\text{S}})$ in a digital map is to inflate $\mathcal{C}_{\text{obs}}$ using the morphological dilation operation [20]

$$\mathcal{C}_{\text{obs}}(D_{\text{S}}) = \mathcal{C}_{\text{obs}} \oplus B_{D_{\text{S}}} = \bigcup_{b \in B_{D_{\text{S}}}} \mathcal{C}_{\text{obs},b}, \tag{13}$$

whereby $B_{D_{\text{S}}}$ represents a disk with radius D_{S}, $\oplus$ represents the dilation operation and $\mathcal{C}_{\text{obs},b}$ is the translation of $\mathcal{C}_{\text{obs}}$ by $b \in B_{D_{\text{S}}}$.

Using $\mathcal{C}_{\text{obs}}(D_{\text{S}})$ instead of $\mathcal{C}_{\text{obs}}$, all the algorithms in the previous sections (Algorithm 1 to 5) can be applied in order to find short solution paths that keep a distance of at least D_{S} to the obstacle region $\mathcal{C}_{\text{obs}}$. In this case, we first compute $\mathcal{C}_{\text{obs}}(D_{\text{s}})$ using (13) and determine the VB for the resulting map with inflated obstacle region using Algorithm 1. After that, we apply Algorithm 4 using $\mathcal{C}_{\text{obs}}(D_{\text{S}})$ to obtain a solution path P_{VV}.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

For illustration, we apply the proposed method to the example environment. The inflated obstacle region and the resulting safe solution paths for $D_S = 5$ and $D_S = 10$ are shown in Fig. 10. As can be seen from Fig. 9 (d) and Fig. 10, the solution paths become longer as the safety distance increases. This is expected since the available free space for the robot motion is reduced for larger values of D_S.

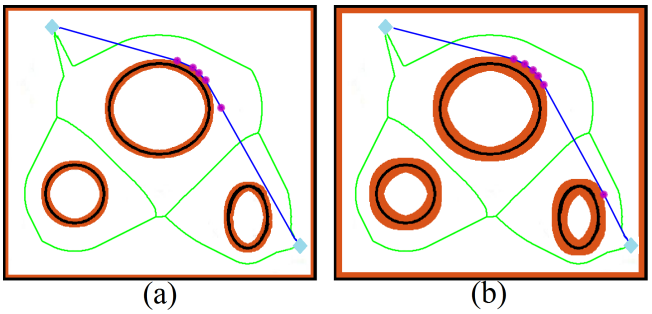

FIGURE 10: Inflated obstacle region for the example environment: (a) $\mathcal{C}_{\text{obs}}(5)$ with $L(P_S) = 636.9$; (b) $\mathcal{C}_{\text{obs}}(10)$ with $L(P_S) = 642.4$.

IV. EVALUATION

In this section, we evaluate the solution paths P_{VV} of the proposed algorithms regarding the obtained path length $L(P_{VV})$ and the computation time T_{VV} for obtaining the solution path. To this end, we compare the obtained solutions to several state-of-the-art methods for a large variety of environment maps. All the algorithms are implemented in Matlab [54] and run under the same conditions on a Laptop with Intel(R) Core(TM) i7-4510 CPU @ 2.60Ghz processor and 6GB RAM. Section IV-A investigates the properties of the VV algorithm for maps with different properties. A comprehensive comparison of different algorithms is provided in Section IV-B and IV-C.

A. PROPERTIES OF THE VV ALGORITHM

We first analyze the properties of the proposed path planning algorithm depending on its constituent components. That is, we evaluate the improvements achieved when applying Algorithm 4 compared to Algorithm 2 and 5. For comparison, we also show the results when using the VA* algorithm as explained in Section III-A, the RR algorithm and the RR-ST algorithm according to [16] and the PRM* algorithm in [13]. Here, the PRM* algorithm is applied with a large number of 15 000 samples (denoted as PRM*-L) and serves as a reference due to its proven convergence to a shortest path. A brief description of the PRM* algorithm is provided in Section I of the supplementary material.

Our evaluation in this section is based on four maps which feature different properties and that are shown together with their graphs in Fig. 11 and 12. Since this section focuses on the evaluation of the VV algorithm, we denote these maps as VV1, VV2, VV3, VV4. The maps VV1 and VV2 in Fig. 11 allow for multiple start-goal walks with similar length in their respective graphs G_{VV1} and G_{VV2}. Hereby, VV1 provides scattered circular and polygonal shapes that leave sufficient space for the robot motion, whereas the maze map VV2 offers tight passages as well as regions with free space. The maps VV3 and VV4 in Fig. 12 have a single start-goal walk. Although there is sufficient space for the robot motion in VV3, this maps requires sharp turns of the robot. Differently, the map VV4 is a maze map with curved obstacle boundaries and tight passages. For each map, we consider three scenarios with safety distances of $D_S = 0$ (original map), $D_S = 5$ and $D_S = 10$ in order to validate the performance of our algorithm in the case of minimum clearance requirements for path safety as specified in Section III-E.

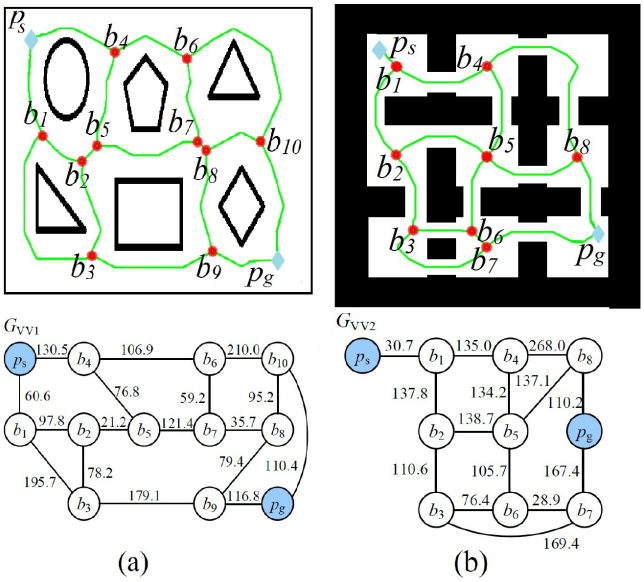

FIGURE 11: Environment maps and corresponding graphs: (a) VV1; (b) VV2.

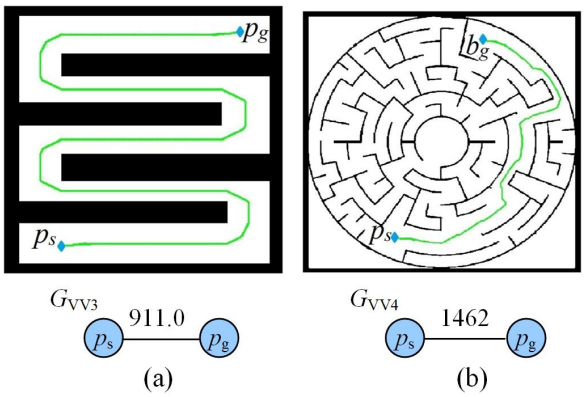

FIGURE 12: Environment maps and corresponding graphs: (a) VV3; (b) VV4.

We first compare the different methods regarding the path length. The obtained results for the different maps are shown in Fig. 13 to 16. It is readily observed that VV-ST-R in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

Algorithm 4 produces the shortest paths among the VB-based algorithms for all the maps and for all the considered safety distances D_S. It can also be seen that replacing the function `RemoveRedundancy` in [16] by the proposed VV-algorithm (Algorithm 2) already leads to a considerable improvement. That is, VV improves on RR and VV-ST improves on RR-ST. Moreover, an additional improvement is achieved by VV-ST-R since it iteratively removes corners by adding Steiner points and then reduces the number of waypoints using the VV-algorithm. Hereby, it is interesting to note that VV-ST-R produces path lengths that are either very close to or shorter than the ones from the close-to-optimal PRM*-L paths.

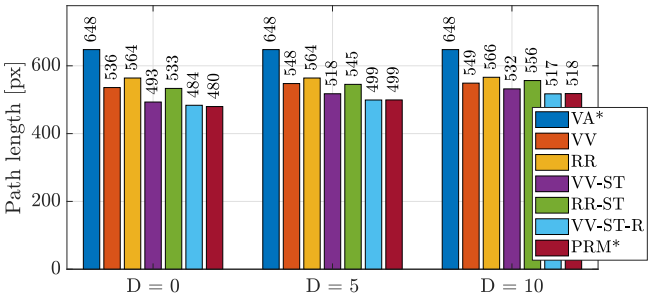

FIGURE 13: Path length comparison for VV1.

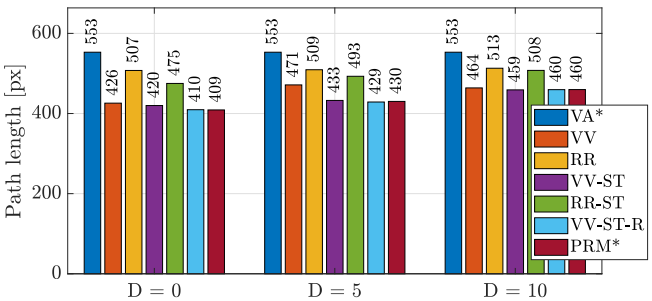

FIGURE 14: Path length comparison for VV2.

In addition, our results regarding the computation time are depicted in Fig. 17 to 20. Here, we note that the computation time for PRM*-L is not shown in these figures since it exceeds 10 min and the computation time for all the algorithms includes the computation of the GVD, which is the prerequisite of all the VB-based algorithms. Since both the RR algorithm and the RR-ST algorithm are based on finding the shortest start-goal walk along the VB, the computation of this walk using the VA* algorithm is included in their computation time. Differently, the computation time for the proposed algorithms VV, VV-ST and VV-ST-R includes the computation of k shortest start-goal walks using Yen's algorithm. In our experiments, we use $k = 4$. It holds that all algorithms show a similar computation time with slight variations depending on the chosen map. Hereby, it is interesting to point out that applying the VV-algorithm repeatedly (VV-ST-R) as suggested in Algorithm 4 reduces the path lengths compared to Algorithm 5 and the algorithm in [16] without increasing the computation time.

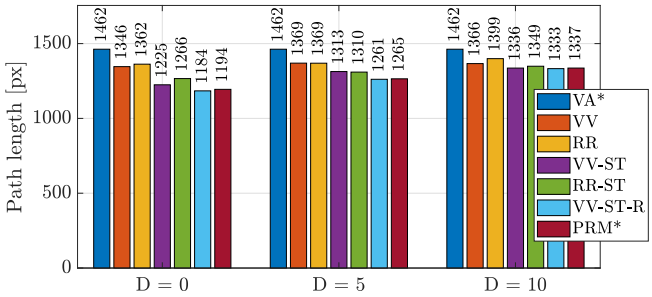

FIGURE 15: Path length comparison for VV3.

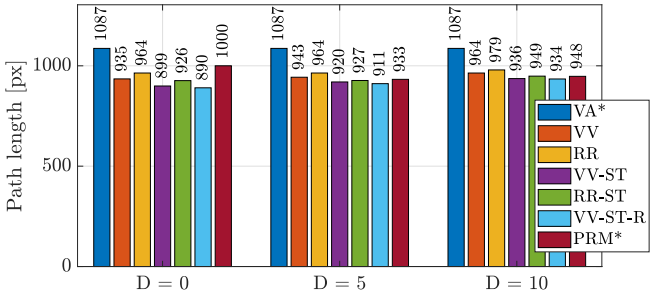

FIGURE 16: Path length comparison for VV4.

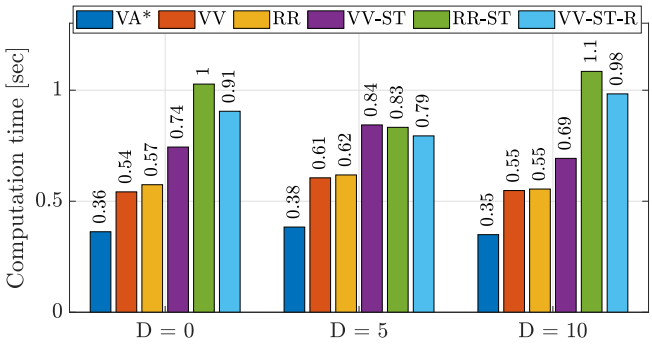

FIGURE 17: Computation time comparison for VV1.

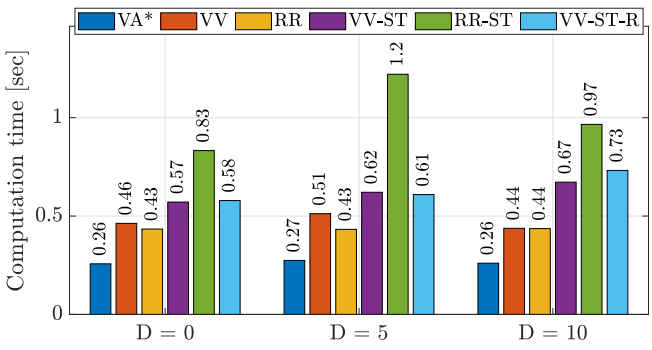

FIGURE 18: Computation time comparison for VV2.

According to our analysis, the computation time of the algorithms RR-ST, VV-ST and VV-ST-R is dominated by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

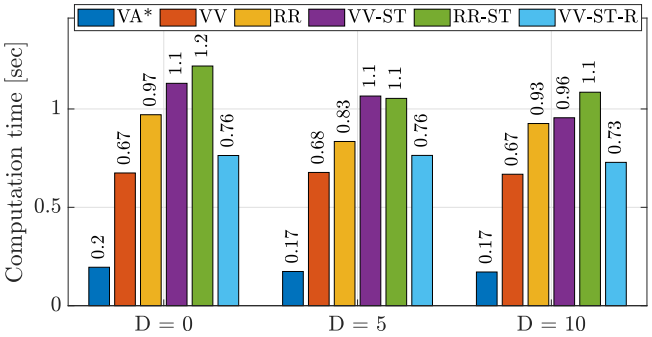

FIGURE 19: Computation time comparison for VV3.

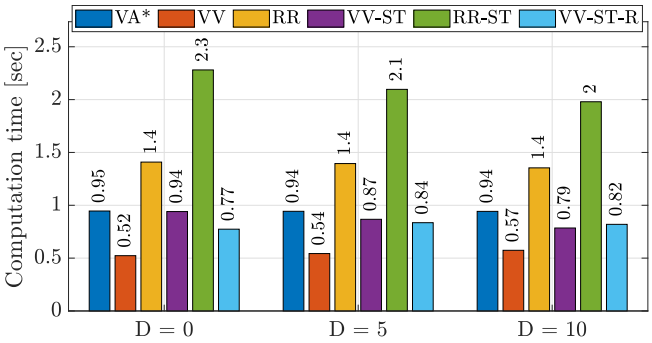

FIGURE 20: Computation time comparison for VV4.

the insertion of Steiner points using `ComputeST`. That is, a good indicator for the computation time is the number of iterations in the while loop in line 9 of Algorithm 3. Table 2 shows this number of iterations for the different algorithms and maps considered in this section. Hereby, we note that the displayed number of iterations for the RR-ST algorithm includes a single start-goal walk, whereas up to $k = 4$ start-goal walks are considered for the VV-ST algorithm and the VV-ST-R algorithm depending on the map.

It is readily observed that the smaller computation time of VV-ST-R is related to the smaller number of iterations compared to the other methods. The main reason for this outcome is the repeated application of the VV algorithm, reducing the number of waypoints of the intermediate solutions paths P supplied to the function `ComputeST` in Algorithm 4 and 5. Hereby, it has to be noted that the number of waypoints generally increases when adding Steiner points, whereas the number of waypoints is reduced by the application of `RemoveRedundancy` or `ComputeVV`. Considering RR-ST and VV-ST, it holds that the number of waypoints is only reduced at the beginning (line 8) and end (line 12) of the corresponding Algorithm 5. Accordingly, the intermediate solution paths in line 10 of Algorithm 5 potentially accumulate a large number of waypoints, leading to an increased number of iterations when applying `ComputeST`. Differently, using VV-ST-R, the number of waypoints of intermediate solutions paths P supplied to `ComputeST` in line 10 of Algorithm 4 is directly reduced after adding Steiner points with `ComputeST` by applying `ComputeVV` in line 11 of Algorithm 4. Hence, a smaller number of iterations is required when adding Steiner points using `ComputeST`. In summary, this discussion further clarifies the advantage of iteratively adding new waypoints and then removing unnecessary waypoints by the VV-ST-R algorithm.

TABLE 2: Number of iterations for the different algorithms

Method	VV-ST	RR-ST	VV-ST-R
	Map VV1		
$D_S = 0$	499	991	418
$D_S = 5$	597	699	366
$D_S = 10$	737	1 318	711
	Map VV2		
$D_S = 0$	331	972	210
$D_S = 5$	550	2 033	261
$D_S = 10$	510	1 191	390
	Map VV3		
$D_S = 0$	1 356	1 267	600
$D_S = 5$	1 356	757	639
$D_S = 10$	939	1 207	522
	Map VV4		
$D_S = 0$	720	2 796	465
$D_S = 5$	717	3 845	570
$D_S = 10$	459	2 096	402

In addition to the computational results, Fig. 21 to Fig. 24 show a selection of solution paths for different maps and methods (the complete set of figures with an animation of the iteration steps is shown in Section II of the supplementary material. The experiment can be repeated using the code in [55]). Fig. 21 and 22 illustrate that the solution paths of Algorithm 4 do not necessarily follow the shortest start-goal walk, which is different from the algorithm in [16]. Furthermore, Fig. 23 and 24 highlight that the paths generated by Algorithm 4 are able to tightly follow the shape of obstacles, while favoring straight-line connections in free space. As a result, these paths are even shorter than the paths obtained from applying the PRM*-L algorithm with a large number of nodes.

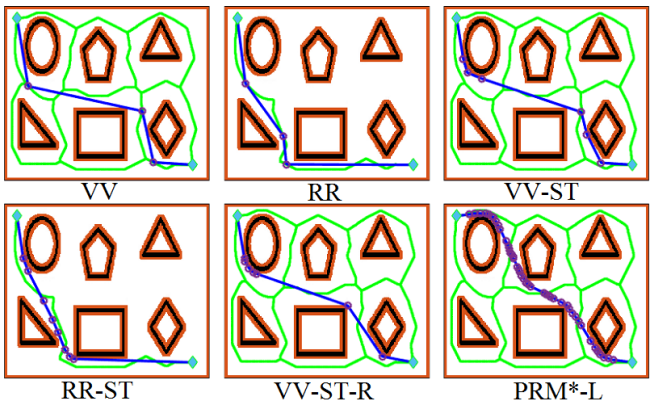

FIGURE 21: Solution paths for VV1 and $D_S = 5$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

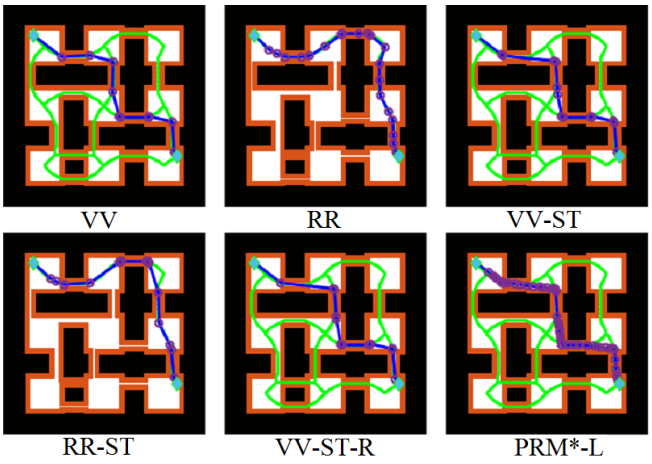

FIGURE 22: Solution paths for VV2 and $D_S = 10$.

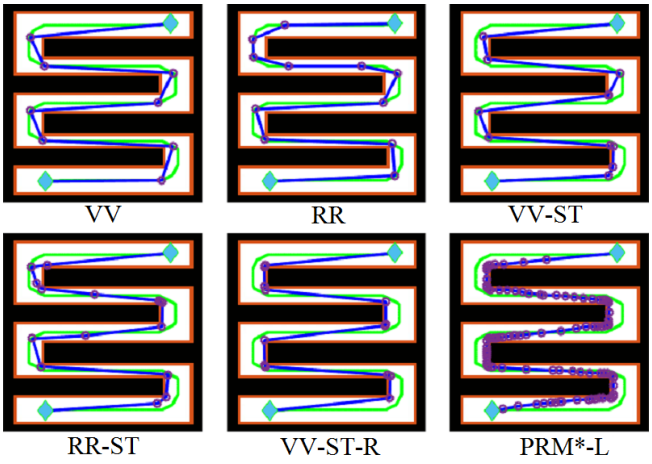

FIGURE 23: Solution paths for VV3 and $D_S = 5$.

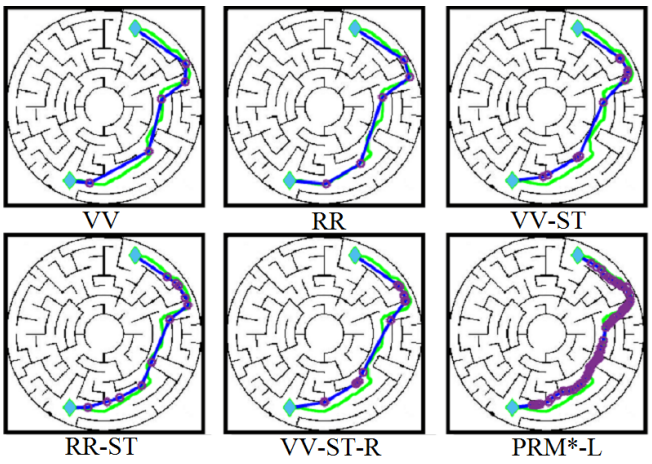

FIGURE 24: Solution paths for VV4 and $D_S = 0$.

B. COMPARISON WITH STATE-OF-THE-ART ALGORITHMS

The results in the previous section indicate that the VV-ST-R algorithm outperforms other algorithms that are based on a refinement of solution paths that follow the VB. We next perform a comparison of this algorithm with several state-of-the-art sampling-based path planning methods. To this end, we consider the 15 different maps in Fig. 25 that were previously used in path planing applications.

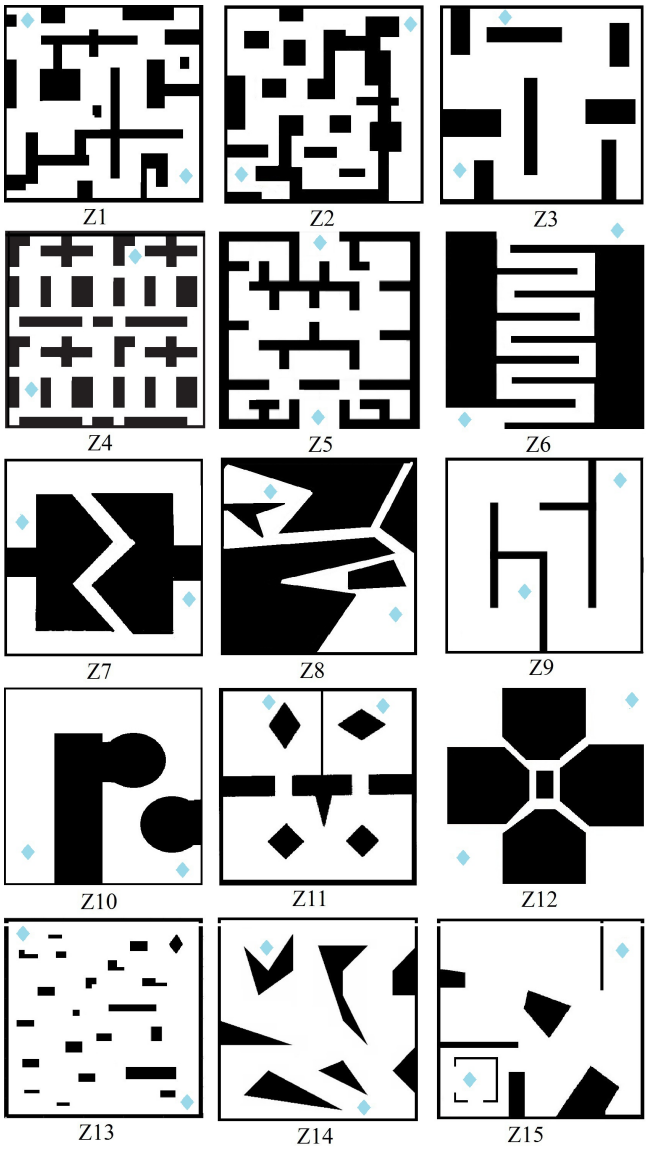

FIGURE 25: Maps for the comparative evaluation.

For the comparison, we choose the PRM* algorithm, which extends the Probabilistic Roadmap (PRM) algorithm, and the RRT* algorithm, which extends the Rapidly exploring Random Tree (RRT) algorithm, since these algorithms are probabilistically complete and asymptotically optimal [13]. In addition, we apply the Fast Marching Tree (FMT) algorithm that is supposed to generate shorter paths than PRM* and RRT* with a reasonable computation time with-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

out being asymptotically optimal [27]. Finally, the very recent Confidence Random Tree (CRT) algorithm is tailored for finding solution paths that keep a safe distance from obstacles without computing the GVD [24]. A description of these algorithms is provided in Section I of the supplementary material. For all these sampling-based algorithms, we take the average of 100 trials and we select a suitable number of nodes/samples in order to ensure finding a solution path in more than 98% of the trials. In addition, similar to the previous section, we perform runs of the PRM*-L algorithm with 15 000 nodes and a long computation time above 10 minutes as a reference for close-to-optimal paths.

The results of our computational experiments are summarized in Table 3 and 4. For convenience the cells of the method with the shortest path and the shortest computation time are shaded in gray. As the first main observation, it can be seen that the VV-ST-R algorithm always generates the shortest path among the algorithms that produce a solution path in a practical time. In addition, VV-ST-R has the smallest computation time in almost all of the cases. As a further advantage, the VV-ST-R algorithm is guaranteed to find a solution path if such path exists since it starts from a connection of the start and goal point along the VB. The implication of this fact can for example be seen when inspecting the computation time for the map Z7 or Z12 for $D_S = 5$. Here, PRM* and FMT require a large number of nodes in order to find a solution path since (i) there is only a narrow passage between obstacles and (ii) most of the sampled nodes are generated in the obstacle region or in the part of the free space that does not contribute to the solution path. In contrast, VV-ST-R is able to shorten the initial solution path, which is given by the connection of p_s and p_g along the VB.

We further highlight several interesting observations from the solutions obtained using the VV-ST-R algorithm for the maps Z6, Z9, Z10, Z14 and Z15. A comparison of the solution paths for VV-ST-R, PRM*-L and RR-ST can be found in Fig. 26. The upper plot shows Z6 for $D_S = 0$. Here, it can be seen that VV-ST-R is able to generate solution paths that turn around sharp corners, while finding short connections in free space. In contrast, RR-ST cannot reduce the path length in case the generated waypoints become too close to each other. This observation is also supported by the second plot for Z9 and $D_S = 0$. This plot further illustrates that the solution paths of VV-ST-R can tightly follow straight walls. This is not the case for the solution paths of the PRM*-L algorithm, which depend on the locations of the randomly sampled nodes. The middle plot shows solution paths for Z10 and $D_S = 0$. It can be readily observed that solution paths generated by VV-ST-R can follow curved objects by placing an increased number of waypoints where necessary. The second plot from the bottom with Z14 and $D_S = 5$ illustrates the advantage of computing solution paths for multiple start-goal walks along the VB. Here, VV-ST-R chooses a different start-goal walk and is hence able to generate a solution path that is considerably shorter than the path for RR-ST. Finally, the lower plot with Z15 demonstrates that solution paths with a reduced number of waypoints are obtained when applying the VV-ST-R algorithm. For completeness, the solution paths of VV-ST-R for the remaining maps and $D_S = 0$ are shown in Fig. 27.

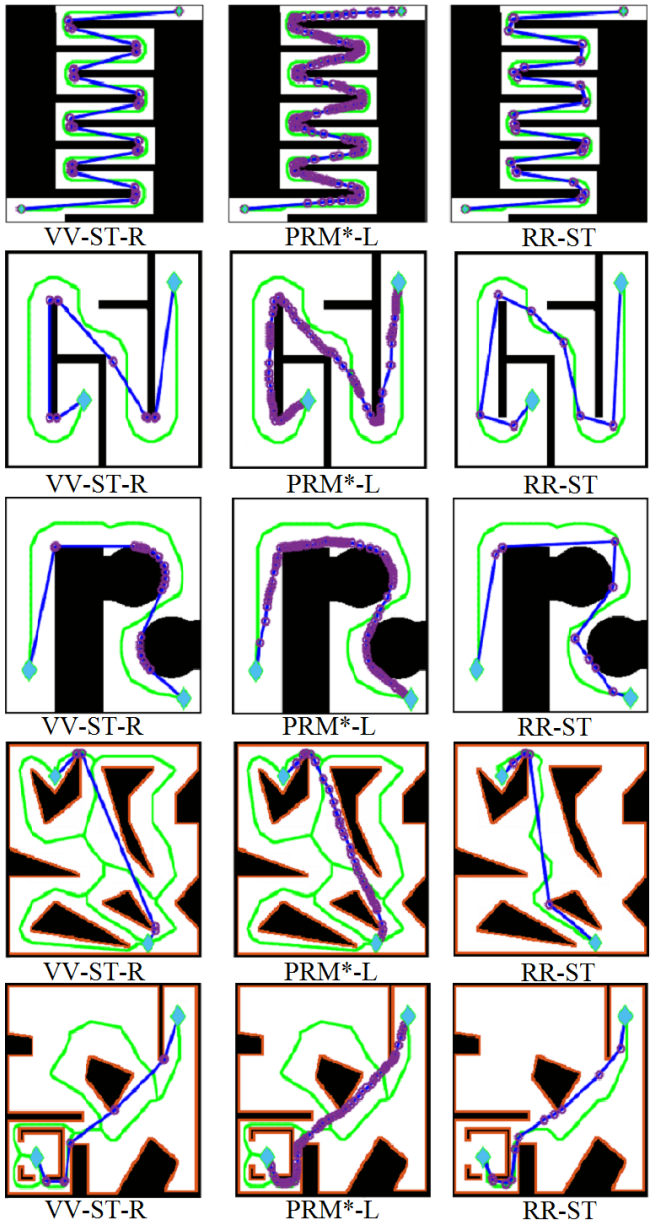

FIGURE 26: Several interesting cases of solution paths for VV-ST-R, PRM*-L and RR-ST.

We next discuss the ability of the VV-ST-R algorithm to compute safe solution paths with a specified minimum clearance. To this end, we write D_{CRT} for the minimum safety distance obtained for each map by applying the CRT algorithm, which tries to maximize the obstacle clearance at each point of the solution path. Then, we select $D_S = D_{CRT} + 1$ as the safety distance for the other methods. We note that this choice of D_S is made in order to compare the idea of inflating the obstacle region with a method that generates safe paths without inflating the obstacle region. In principle, any choice of D_S would be possible as long as there is a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

TABLE 3: Path length comparison

Map	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11	Z12	Z13	Z14	Z15
	$D_S = 0$														
VV-ST-R	720	661	533	486	500	961	931	651	1234	915	651	456	533	585	551
PRM*-L	727	671	542	490	508	958	945	655	1267	925	660.7	457	536	601	557
RR-ST	769	693	577	498	511	1040	983	658	1421	1005	700	462	541	616	564
PRM*	787	690	571	511	509	1009	1012	675	1379	981	699	462	547	617	592
RRT*	1086	856	819	655	748	1302	1350	912	2004	1316	1102	476	663	817	735
FMT	756	678	551	495	511	1046	955	674	1289	954	671	463	537	596	568
CRT	878	806	665	606	590	1209	1151	775	1484	1138	809	509	613	682	657
	$D_S = 5$														
VV-ST-R	746	686	553	503	511	1104	992	677	1277	946	677	470	539	600	575.4
PRM*-L	757	691	564	511	514	1106	997	684	1305	955	688	471	541	607	582
RR-ST	778	720	562	540	530	1122	1053	701	1459	995	723	472	557	619	583
PRM*	765	698	638	525	531	1124	1020	691	1486	1006	723	473	559	652	603
RRT*	1148	876	850	707	823	1277	1324	853	2024	1317	1174	479	676	846	745
FMT	759	698	605	521	525	1123	1020	703	1363	986	705	475	547	627	592
CRT	884	797	667	606	590	1208	1151	777	1483	1131	808	508	613	684	658

TABLE 4: Computation time comparison

Map	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11	Z12	Z13	Z14	Z15
	$D_S = 0$														
VV-ST-R	1.7	1.4	1.7	1.3	1.2	0.49	1.1	1.2	1.5	1.2	1.5	0.9	1.4	1.4	1.5
RR-ST	1.6	1.6	2.2	2.2	1.8	0.89	2.1	2.2	2.5	1.7	1.9	1.9	2.4	2.5	2.1
PRM*	2.1	14	10	3.8	4.3	9.2	4.4	9.8	3.8	1.9	1.9	2.3	2.8	4.3	2.7
RRT*	2	9.1	9	12	1.2	3.8	4.4	1.3	3.4	1.6	1.8	1.6	4	3.5	5.5
FMT	4.1	39	28	14	7.6	4.4	5.1	2.9	5.6	3.3	9.1	3.9	9	13.2	10
CRT	13	12	8.7	12	2.4	4.8	10	2.3	14	7.7	10.5	4.3	10	15	9.1
	$D_S = 5$														
VV-ST-R	1.7	1.4	1.6	1.3	1.2	0.49	1.1	1.2	1.5	1.3	1.6	0.9	1.5	1.6	1.6
RR-ST	2.1	2	1.9	2	1.6	0.9	1.9	1.7	3.6	2.2	2.2	1.4	2.5	1.8	2.1
PRM*	16	16	1.6	8.1	5.5	8.2	87	12	1.5	2	1.9	32.5	1.9	1.6	8.6
RRT*	3.4	3.1	1.5	7.5	3	5.8	13	3	1.3	1.3	1.3	4.5	1.5	1.4	4.9
FMT	54	70	3.1	17	12	16	29	28	3.7	3.4	5.4	20	6.6	4.3	16
CRT	5.6	5.3	8.4	15	12	7.2	12	4.5	14	7.6	9.8	5.2	10	15	9.3

TABLE 5: Performance evaluation for safe paths with $D_S = D_{CRT} + 1$

Map	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11	Z12	Z13	Z14	Z15
	Path length														
VV-ST-R	798	680	604	543	534	961	1034	705	1371	1027	727	470	561	632	594
PRM*-L	802	691	607	552	543	1106	1042	701	1384	1023	732	471	563	666	602
RR-ST	817	720	628	559	535	1040	1079	708	1435	1069	795	472	566	658	595
PRM*	819	698	618	586	681	1124	1050	707	1478	1033	743	473	575	690	610
RRT*	1176	876	903	706	826	1277	1320	830	2046	1316	1186	479	717	957	735
FMT	886	698	636	577	621	1123	1060	710	1438	1025	755	475	572	684	604
CRT	883	797	654	608	589	1208	1163	774	1485	1131	807	508	614	685	656
	Computation time														
VV-ST-R	1.7	.97	1.7	1.7	1.3	0.49	1.6	2.4	2.1	2.2	1.6	0.95	2	1.8	1.7
RR-ST	2.3	2.9	2	2.1	2.1	0.94	2.5	1.7	2.8	2.4	1.9	1.4	2	2.1	2.2
PRM*	8.3	16	9.6	23	12	8.2	150	62	4.2	36	9.4	32	13	8.9	45
RRT*	2.3	3.1	2.7	3.3	3.7	5.8	19	5.3	3.4	3.5	2.4	4.5	4.2	3.7	5.7
FMT	25	70.4	27	46	51	16	53	140.5	5.6	26	28	19.9	43	14	41
CRT	4.1	5.3	4.3	5.7	12	7.2	11	9.3	13	7.7	3.7	5.2	10	15	9.1
	Minimum distance to $\mathcal{C}_{obs}$														
VV-ST-R	14	6	16	11.7	12	6	11	8.5	16	15.9	14	6.4	16.8	13	10
PRM*-L	14	6	16.1	12	13	6	11	9	16	16	14.5	6	18	15	10.5
RR-ST	14	6.1	17	11.2	12	6	12	8.5	19.1	18.2	15	6.1	16.9	13	10.2
PRM*	14.2	6.1	17.4	12	12.2	6	11.3	9.1	17.9	15.8	14.6	6.5	17.1	17.5	9.4
RRT*	14	6	16.9	12	12.1	6	11.8	9	17.3	16.5	14.4	6.4	15.8	17.4	9
FMT	14.7	6.3	17.5	12.1	12.2	6	11.8	9	17.8	15.6	14.8	6.6	17.2	17.2	9.6
CRT	10.8	5.4	13.4	9.2	9.8	4.6	10.2	7.2	13	13.7	11.4	5.7	12.9	12.2	8.6

connection between p_s and p_g along the VB. In addition to path length and computation time, we also investigate the minimum distance to $\mathcal{C}_{obs}$ as shown in Table 5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

Looking at the results for the path length, we first note that the methods based on the inflated obstacle region are able to find a shorter solution path than the CRT algorithm. This is an expected result since the CRT algorithm is targeted for achieving a maximum obstacle clearance. Most importantly, VV-ST-R always finds the shortest safe solution paths among all the methods with a practical computation time. In most of the cases, the solution paths of VV-ST-R are determined with the shortest computation time and are as well shorter than the ones of PRM*-L.

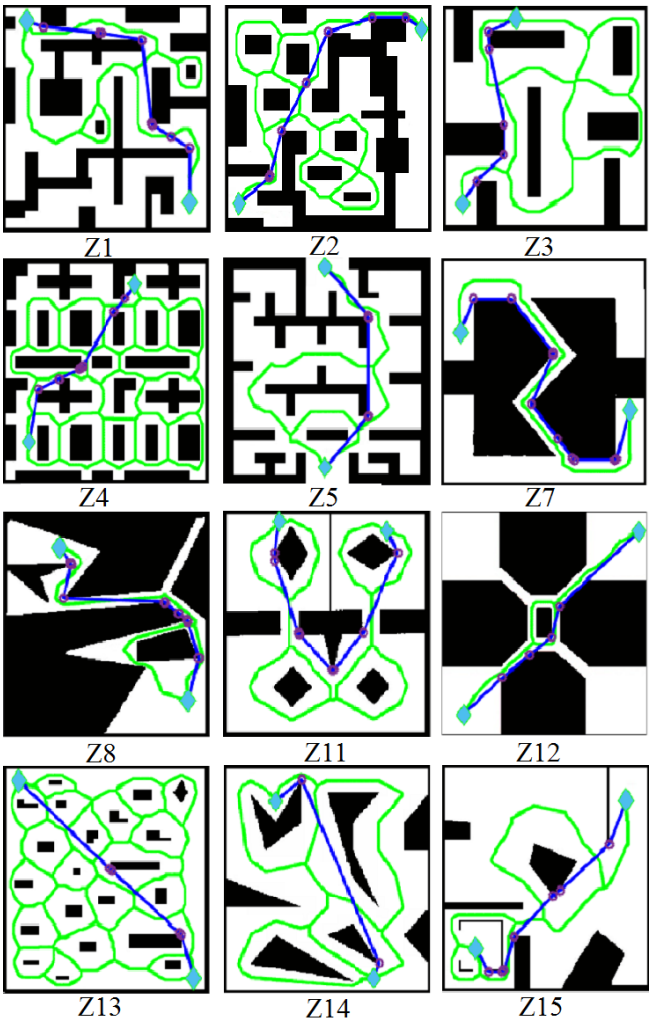

FIGURE 27: Solution paths of VV-ST-R for different maps.

We finally illustrate the findings of this experiment by comparing the solution paths of VV-ST-R, PRM*-L and CRT for the maps Z4, Z13 and Z14 in Fig. 28. Here, we first note that both PRM*-L and CRT require a large number of waypoints, leading to large computation times. The upper plot shows that, even all the methods follow the same start-goal walk, the shortest path is computed by VV-ST-R due to the effective placement of a small number of waypoints. In contrast, the solution paths of CRT show unnecessary turns since this algorithm tries to find a path with the highest confidence of avoiding obstacles but without knowledge about the VB. This is true both in cases where CRT follows the correct start-goal walk (upper and center plot) and in cases where CRT follows a different start-goal walk (bottom plot). Furthermore, the safety distance achieved by CRT is a result of applying the algorithm and cannot be adjusted as for VV-ST-R by setting D_S. We note that the complete set of solution paths for the different maps is given in Section III of the supplementary material and the related Matlab code is available in [55].

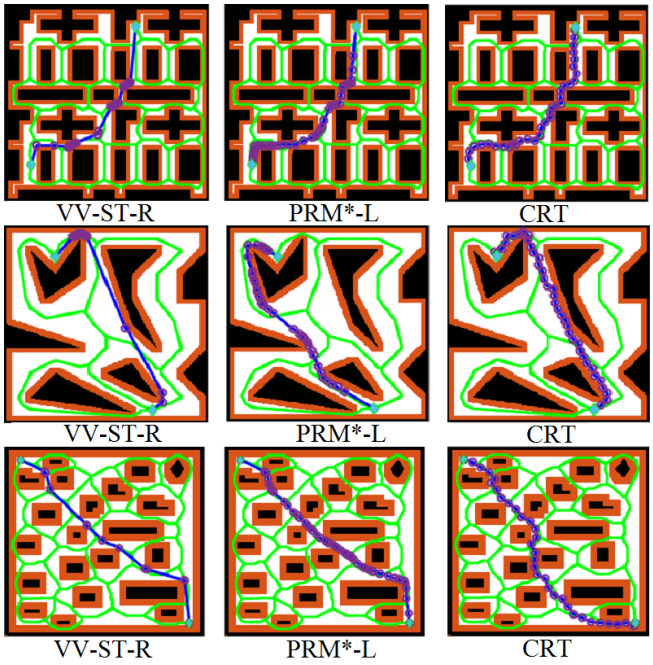

FIGURE 28: Comparison of safe solution paths.

C. COMPARISON TO THE VISIBILITY GRAPH COMPUTATION

It is a well-known fact that Visibility Graphs (VGs) can be used to find the shortest straight-line path between start and goal points in robot environments with polygonal obstacles [32], [56]. Hereby, the computation time strongly depends on the number of obstacle vertexes N_V that need to be considered during the computation. In this section, we compare the solution paths of the proposed method in Algorithm 4 and from the VG for polygon maps with different numbers of obstacle vertexes. The obtained results are summarized in Table 6 and the solution paths are shown in Fig. 29 to 32.

TABLE 6: Path length and computation time comparison for VV-ST-R and VG.

Map	N_V	Path length [px]		Computation time [s]	
		VV-ST-R	VG	VV-ST-R	VG
VV3	22	1184	1178	0.44	1.74
Z4	134	485.0	484.7	0.68	55.3
Z13	96	533.2	533.0	0.82	24.4
Z15	45	551.4	549.3	0.35	5.2

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010819, IEEE Access

It is readily observed that the number of vertexes significantly affects the computation time when using the VG. Differently, the VV-ST-R algorithm is able to compute close-to-optimal solution paths in a very short time. In this context, we note that Table 6 reveals a common property of solution paths obtained from the VG. Since the shortest paths in the VG are generally tangent to obstacles or cross obstacle vertexes, they contain semi-free configurations. That is, robots will be in contact with the obstacle (zero clearance) when passing such configurations [36]. In contrast, our method determines solution paths that are completely outside obstacles. The slightly longer solution paths obtained by the VV-ST-R algorithm are due to this property of the VG. We further emphasize that the VG is only applicable as an exact method for polygon maps. Hence, a great advantage of the proposed VV-ST-R algorithm is the computation of close-to-optimal solution paths for general maps with very small computation times. This is for example confirmed for the map in Section III and the maps VV1, VV4 and Z10 with curved objects.

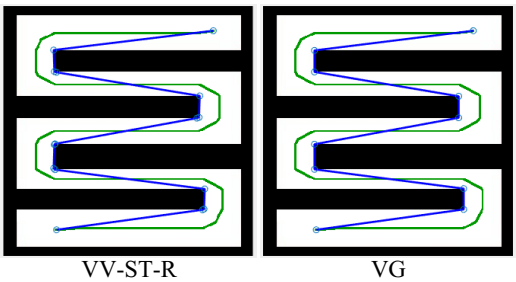

FIGURE 29: VG comparison for the map VV3.

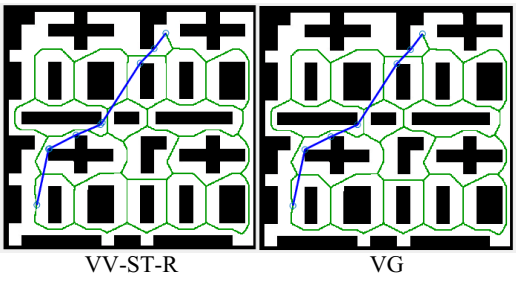

FIGURE 30: VG comparison for the map Z4.

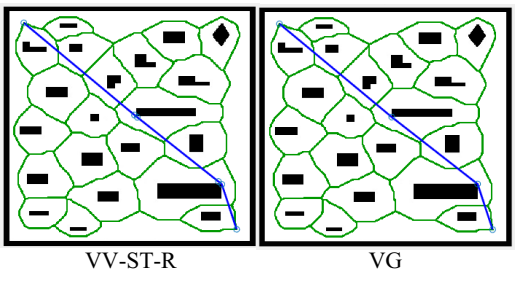

FIGURE 31: VG comparison for the map Z13.

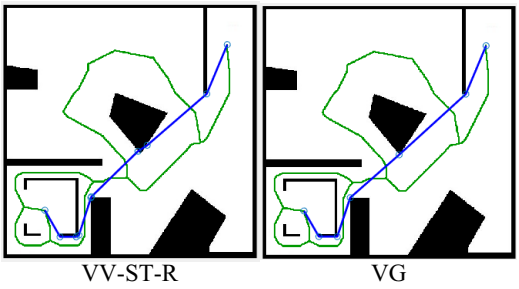

FIGURE 32: VG comparison for the map Z15.

V. CONCLUSION

The subject of this paper is the path planning problem for mobile robots in a two-dimensional configuration space with obstacles. Hereby, we consider the practical case, where an environment map is given as a digital map instead of a collection of geometric objects with precise coordinates. When solving the described path planning problem, different performance metrics have to be taken into account. First, it is desired to compute robot paths that are as short as possible. Second, it is beneficial for safety if solution paths have a specified minimum obstacle clearance. Finally, it is very important for real-time applications to generate suitable paths with a short computation time.

In order to address the stated requirements, we develop a new path planning method that uses information about the topology of the environment that is obtained from the generalized Voronoi diagram (GVD) in order to determine initial solution paths. These paths are then refined by iteratively removing waypoints to realize shortcuts and adding new waypoints to cut corners. Since the solution paths in each iteration contain a small number of waypoints, our method is computationally efficient, while producing close-to-optimal solutions. Hereby, our method is guaranteed to find a solution path whenever such path exists.

Our comparative evaluation with different state-of-the-art methods and a large number of maps with different properties shows that the proposed algorithm is able to produce better solution paths in a short time. We further demonstrate that our method allows addressing path safety by using environment maps with inflated obstacles.

The proposed method is currently formulated for static environments with straight-line solution paths. Accordingly, our future work will focus on the extension to environments with dynamic obstacles and the application of smoothing to make the generated solution paths usable for nonholonomic robots.

REFERENCES

[1] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," International Journal of Advanced Robotic Systems, vol. 16, no. 2, pp. 1–22, 2019.

[2] S. Spanogianopoulos and K. Sirlantzis, Car-Like Mobile Robot Navigation: A Survey, pp. 299–327. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[3] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," Journal of Intelligent & Robotic Systems, vol. 91, pp. 35–58, 2018.

[4] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," Robotica, vol. 33, no. 3, p. 463âĂŞ497, 2015.

[5] I. Noreen, A. Khan, H. Ryu, N. L. Doh, and Z. Habib, "Optimal path planning in cluttered environment using RRT*-AB," Intell. Serv. Robot., vol. 11, p. 41âĂŞ52, Jan. 2018.

[6] L. G. D. O. Véras, F. L. L. Medeiros, and L. N. F. GuimarÃąes, "Systematic literature review of sampling process in rapidly-exploring random trees," IEEE Access, vol. 7, pp. 50933–50953, 2019.

[7] D. G. Macharet and M. F. M. Campos, "A survey on routing problems and robotic systems," Robotica, vol. 36, no. 12, p. 1781âĂŞ1803, 2018.

[8] Y. Dong, E. Camci, and E. Kayacan, "Faster RRT-based nonholonomic path planning in 2D building environments using skeleton-constrained path biasing," J. Intell. Robotics Syst., vol. 89, p. 387âĂŞ401, Mar. 2018.

[9] S. Agnisarman, S. Lopes, K. C. Madathil, K. Piratla, and A. Gramopadhye, "A survey of automation-enabled human-in-the-loop systems for infrastructure visual inspection," Automation in Construction, vol. 97, pp. 52 – 76, 2019.

[10] R. Cui, Y. Li, and W. Yan, "Mutual information-based multi-AUV path planning for scalar field sampling using multidimensional RRT*," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 46, pp. 993–1004, July 2016.

[11] G. Jain, G. Yadav, D. Prakash, A. Shukla, and R. Tiwari, "MVO-based path planning scheme with coordination of UAVs in 3-D environment," Journal of Computational Science, vol. 37, p. 101016, 2019.

[12] T. T. Mac, C. Copot, T. T. Duc, and R. D. Keyser, "Heuristic approaches in robot path planning: A survey," Robotics and Autonomous Systems, vol. 86, pp. 13–28, 2016.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, vol. 30, no. 7, p. 846âĂŞ894, 2011.

[14] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," Robotics and Autonomous Systems, vol. 100, pp. 171 – 185, 2018.

[15] B. B. K. Ayawli, X. Mei, M. Shen, A. Y. Appiah, and F. Kyeremeh, "Mobile robot path planning in dynamic environment using Voronoi diagram and computation geometry technique," IEEE Access, vol. 7, pp. 86026–86040, 2019.

[16] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning - using the Voronoi diagram for a clearance-based shortest path," IEEE Robotics & Automation Magazine, vol. 15, 2008.

[17] P. Loncomilla, J. R. del Solar, and L. Martínez, "Object recognition using local invariant features for robotic applications: A survey," Pattern Recognition, vol. 60, pp. 499 – 514, 2016.

[18] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin, "Path planning for mobile robot navigation using Voronoi diagram and fast marching," in IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2376–2381, IEEE, 2006.

[19] R. Samaniego, J. Lopez, and F. Vazquez, "Path planning for non-circular, non-holonomic robots in highly cluttered environments," Sensors, vol. 17, no. 8, 2017.

[20] B. Ayawli, X. Mei, M. Shen, A. Y. Appiah, and F. Kyeremeh, "Optimized RRT-A* path planning method for mobile robots in partially known environment," Information technology and control, vol. 48, no. 2, pp. 179–194, 2019.

[21] A. Khan, I. Noreen, and Z. Habib, "On complete coverage path planning algorithms for non-holonomic mobile robots: Survey and challenges," J. Inf. Sci. Eng., vol. 33, pp. 101–121, 2017.

[22] P. Corke, Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Springer Publishing Company, Incorporated, 1st ed., 2013.

[23] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, 5 2005.

[24] Y. N. Kim, D. W. Ko, and I. H. Suh, "Confidence random tree-based algorithm for mobile robot path planning considering the path length and safety," International Journal of Advanced Robotic Systems, vol. 16, no. 2, pp. 1–10, 2019.

[25] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.

[26] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," The international journal of robotics research, vol. 20, no. 5, pp. 378–400, 2001.

[27] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," The International journal of robotics research, vol. 34, no. 7, pp. 883–921, 2015.

[28] M. R. H. Al-Dahhan and M. M. Ali, "Path tracking control of a mobile robot using fuzzy logic," in International Multi-Conference on Systems, Signals & Devices, pp. 82–88, IEEE, 2016.

[29] M. Elbanhawi, M. Simic, and R. N. Jazar, "Continuous path smoothing for car-like robots using b-spline curves," Journal of Intelligent & Robotic Systems, vol. 80, no. 1, pp. 23–56, 2015.

[30] K. R. Simba, N. Uchiyama, and S. Sano, "Real-time smooth trajectory generation for nonholonomic mobile robots using bézier curves," Robotics and Computer-Integrated Manufacturing, vol. 41, pp. 31–42, 2016.

[31] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate," Expert Systems with Applications, vol. 123, pp. 82–90, 2019.

[32] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," Communications of the ACM, vol. 22, no. 10, pp. 560–570, 1979.

[33] S. Oh and H. W. Leong, "Edge n-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths," in Tenth Annual Symposium on Combinatorial Search, pp. 64–72, 2017.

[34] E. Masehian and M. Amin-Naseri, "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning," Journal of Robotic Systems, vol. 21, no. 6, pp. 275–300, 2004.

[35] M. R. H. AL-DAHHAN and K. W. Schmidt, "Safe and efficient path planning for omni-directional robots using an inflated voronoi boundary," Çankaya University Journal of Science and Engineering, vol. 16, no. 2, pp. 46–69, 2019.

[36] R. Wein, J. P. van den Berg, and D. Halperin, "The visibilityâĂŞVoronoi complex and its applications," Computational Geometry, vol. 36, no. 1, pp. 66–87, 2007. Special Issue on the 21st European Workshop on Computational Geometry.

[37] Q. Wang, M. Wulfmeier, and B. Wagner, "Voronoi-based heuristic for nonholonomic search-based path planning," in Intelligent Autonomous Systems 13, pp. 445–458, Springer, 2016.

[38] P. Bhattacharya and M. L. Gavrilova, "Geometric algorithms for clearance based optimal path computation," in Annual ACM International Symposium on Advances in Geographic Information Systems, 2007.

[39] K. Yang, "Anytime synchronized-biased-greedy rapidly-exploring random tree path planning in two dimensional complex environments," International Journal of Control, Automation and Systems, vol. 9, no. 4, p. 750, 2011.

[40] J. Wang, W. Chi, M. Shao, and M. Q.-H. Meng, "Finding a high-quality initial solution for the RRTs algorithms in 2D environments," Robotica, vol. 37, no. 10, pp. 1677–1694, 2019.

[41] R. Kala, "Homotopic roadmap generation for robot motion planning," Journal of Intelligent & Robotic Systems, vol. 82, no. 3-4, pp. 555–575, 2016.

[42] M. Korkmaz and A. Durdu, "Comparison of optimal path planning algorithms," in International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering, pp. 255–258, 2018.

[43] H. K. B. K. Kim, "Minimum-energy cornering trajectory planning with self-rotation for three-wheeled omni-directional mobile robots," International Journal of Control and Automation Systems, vol. 15, pp. 1857–1866, 2017.

[44] A. Sheikhlar and A. Fakharian, "Online policy iteration-based tracking control of four wheeled omni-directional robots," Journal of Dynamic Systems, Measurement, and Control, vol. 140, 03 2018. 081017.

[45] Q. Wang, M. Wulfmeier, and B. Wagner, "Voronoi-based heuristic for nonholonomic search-based path planning," in Intelligent Autonomous Systems 13, pp. 445–458, Springer, 2016.

[46] D. T. Lee, "Medial axis transformation of a planar shape," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-4, pp. 363–369, July 1982.

[47] R. Ogniewicz and M. Ilg, "Voronoi skeletons: theory and applications," in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 63–69, June 1992.

[48] L. Mestetskiy and A. Semenov, "Binary image skeleton - continuous approach," in International Conference on Computer Vision Theory and Applications, 2008.

[49] E. W. Dijkstra et al., "A note on two problems in connexion with graphs," Numerische mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[50] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.
[51] J. Y. Yen, "Finding the k shortest loopless paths in a network," Management Science, vol. 17, no. 11, pp. 712–716, 1971.
[52] M. Hasan, M. L. Gavrilova, and J. G. Rokne, "A geometric approach to clearance based path optimization," in Computational Science and Its Applications – ICCSA 2007 (O. Gervasi and M. L. Gavrilova, eds.), (Berlin, Heidelberg), pp. 136–150, Springer Berlin Heidelberg, 2007.
[53] D. Hsu, J. . Latcombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," in IEEE International Symposium on Assembly and Task Planning, pp. 280–285, July 1999.
[54] MATLAB, version 9.6.0.1072779 (R2019a). Natick, Massachusetts: The MathWorks Inc., 2019.
[55] M. R. H. Al-Dahhan and K. W. Schmidt, "Path-planning for mobile robots based on Voronoi diagrams." http://users.metu.edu.tr/schmidt/software/VoronoiVisibility.zip, 2020.
[56] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Computational Geometry: Algorithms and Applications. Springer-Verlag Berlin Heidelberg, 3rd ed., 2008.

MOHAMMED RABEEA HASHIM AL-DAHHAN received the B. Eng. degree in Computer Technology Engineering from Al-Maaref College University, Iraq in 2013, he completed the Msc. Mechatronics program at Philadelphia University, Jordan in 2015, and received the Ph.D. degree in Electronic and Communication Engineering from Çankaya University, Turkey in 2020. His research interests include robotics and intelligent control.

KLAUS WERNER SCHMIDT received the Diploma and Ph.D. degrees from the University of Erlangen-Nürnberg, Germany, in 2002 and 2005, respectively, both in Electrical, Electronic, and Communication Engineering. He is currently a Professor at the Department of Electrical & Electronics Engineering, Middle East Technical University, Ankara.

His research interests include supervisory control for discrete event systems, industrial automation systems, industrial communication networks, intelligent transportation systems and industrial project control. He is an Associate Editor for Discrete Event Dynamic Systems and the Turkish Journal of Electrical Engineering & Computer Sciences.