Int. J. Appl. Math. Comput. Sci., 2018, Vol. 28, No. 3, 493–504
DOI: 10.2478/amcs-2018-0038

AN EXACT GEOMETRY–BASED ALGORITHM FOR PATH PLANNING

HASSAN JAFARZADEH [a,*], CODY H. FLEMING [a]

[a]Department of Systems and Information Engineering
University of Virginia, 151 Engineer's Way, Charlottesville, VA, USA
e-mail: hj2bh@virginia.edu

A novel, exact algorithm is presented to solve the path planning problem that involves finding the shortest collision-free path from a start to a goal point in a two-dimensional environment containing convex and non-convex obstacles. The proposed algorithm, which is called the shortest possible path (SPP) algorithm, constructs a network of lines connecting the vertices of the obstacles and the locations of the start and goal points which is smaller than the network generated by the visibility graph. Then it finds the shortest path from start to goal point within this network. The SPP algorithm generates a safe, smooth and obstacle-free path that has a desired distance from each obstacle. This algorithm is designed for environments that are populated sparsely with convex and nonconvex polygonal obstacles. It has the capability of eliminating some of the polygons that do not play any role in constructing the optimal path. It is proven that the SPP algorithm can find the optimal path in $O(nn'^2)$ time, where n is the number of vertices of all polygons and n' is the number of vertices that are considered in constructing the path network $(n' \leq n)$. The performance of the algorithm is evaluated relative to three major classes of algorithms: heuristic, probabilistic, and classic. Different benchmark scenarios are used to evaluate the performance of the algorithm relative to the first two classes of algorithms: GAMOPP (genetic algorithm for multi-objective path planning), a representative heuristic algorithm, as well as RRT (rapidly-exploring random tree) and PRM (probabilistic road map), two well-known probabilistic algorithms. Time complexity is known for classic algorithms, so the presented algorithm is compared analytically.

Keywords: shortest possible path (SPP) algorithm, path planning, collision-free path.

1. Introduction

Path planning problems are concerned with finding a shortest possible path from a starting point toward a goal point in an environment populated with obstacles. In a path planning problem, an algorithm is given the required information of the components such as the location and the shape of the obstacles and the location of the start and goal points, and it generates an optimal or near optimal collision-free path in the least possible time. Because the generated path is an input for navigation, smoothness of the path is a critical factor in addition to path length and safety. Generally, the major problems for path planning are computational complexity, existence of local (or global) optima, and adaptability to different environments.

There is already an extensive literature on path planning, and methods have been developed to plan path in real time, for example, the Voronoi diagram method, the directed graph based method, A* algorithms, probabilistic road map (PRM) methods, rapidly-exploring random tree (RRT) based methods, mixed-integer linear programming (MILP) based methods, i.e., are used to model and solve the sophisticated problems with different constraints and objective functions (Klaučo *et al.*, 2016), potential field approaches, bouncing algorithms, etc. All these methods have been successfully applied to path planning applications, however there are still limitations with respect to computational complexity and optimality.

To solve this problem, an efficient and flexible algorithm is needed, which is applicable for real-time path planning problems. To handle computational complexity, the algorithm must search all the near optimal paths, including the optimal one, but overlook other poor solutions. The algorithm presented in this work, called the shortest possible path (SPP), is classified under a subset of classic algorithms called geometry based algorithms. Like existing geometry based algorithms, which will be described in the next section, SPP generates the optimal

[*]Corresponding author

path, which is not guaranteed by non-classic algorithms such as heuristic or probabilistic approaches. However, SPP has significantly lower computational complexity than the other geometry based algorithms.

The paper is organized as follows: the second section provides a review of the state of the art in path planning algorithms, including heuristic, probabilistic and classic algorithms, and Section 3 then describes SPP formally. Section 4 presents the results of the computer simulation and compares SPP performance with a baseline algorithm under a series of complex scenarios. Finally, the paper concludes with a summary and directions for future study.

2. Literature review

Though the body of research is fairly mature, with the increasing prevalence of autonomous systems it is important to develop alternative and more efficient techniques. To solve the problem in which the environment is assumed to be static, numerous algorithms have been presented and can be classified in three major groups: heuristic, probabilistic, and classic algorithms (Mac *et al*., 2016). Heuristic algorithms are flexible and can be adapted for different optimization and decision making problems such as path planning. Although they often generate a good solution, the optimality of the solutions is not guaranteed. There has been a trend towards increasing usage of heuristic algorithms in path planning, compared with classic algorithms.

The classic algorithms suffer from high time complexity in high dimensional spaces, and some fail to reach global optimal solution. To overcome the limitations of classic algorithms, probabilistic algorithms have been developed (Masehian and Sedighizadeh, 2007). Probabilistic algorithms do not provide optimal solutions necessarily but they are a suitable choice for three-dimensional problems such as path planning for manipulators. They also tend to generate solutions quickly, but they have problems expanding through narrow passages and getting around obstacles.

Being relatively easy to implement, the classical and probabilistic approaches are preferred in most of the real-time path planning applications. Although these methods are effective, most of these algorithms do not provide any theoretical guarantees for obtaining an optimal solution (Tang *et al*., 2012). Furthermore, like the heuristic algorithms, they also suffer from high computational complexity and, consequently, long running times (Tang *et al*., 2012).

In this section some of the path planning algorithms are studied in three major classes with focus on the geometry-based algorithms in classic algorithms group:

2.1. Heuristic algorithms. The term heuristic is used for algorithms which find a good solution but they do not guarantee the optimality. Therefore, they might be considered as approximate and not exact algorithms. These algorithms, usually find a solution close to the optimal one and they find it fast. Among the heuristic algorithms, genetic algorithms (GAs) (Davoodi *et al*., 2015; Jafarzadeh *et al*., 2017), artificial neural networks (ANNs) (Ni *et al*., 2017), and ant colony optimization (ACO) (Liu *et al*., 2017) have been the most popular for solving path planning problems (Masehian and Sedighizadeh, 2007). These algorithms take into account different objectives at the same time and can be adjusted for a wide variety of problems. But, they do not guarantee the optimality of the generated solution. Increasing usage of heuristic algorithms in path planning has been observed compared with the classical algorithms. The heuristic algorithms are considered more intelligent and advanced methods, which can handle uncertain and incomplete information in continuously changing environments and obtain near-optimal solutions. On the other hand, many of the heuristic algorithms require an additional learning phase and very high computational cost (Mac *et al*., 2016).

2.2. Probabilistic algorithms. These are a group of algorithms that take random samples from the configuration space, testing them for feasibility, and use a local planner to connect them to other nearby configurations. This class of algorithms, including rapidly-exploring random trees (RRTs) (LaValle, 1998) and probabilistic road maps (PRMs) (Kavraki *et al*., 1998) as well as other versions of PRM such as lazy PRM (Bohlin and Kavraki, 2000) and semi-lazy PRM (Akbaripour and Masehian, 2017), consists of two phases. In the first phase a probabilistic road map is incrementally constructed. In the second phase which is called query phase, the constructed road map is used for solving individual path planning in the given environment. The shortest path finder algorithms such as Dijkstra's algorithm is used to accomplish the query phase. Further details can be found in the work of Latombe (2012). RRT has difficulty in expanding through narrow passages and getting around the obstacles. In addition, the generated path is not smooth and suboptimal with respect to path length. To solve this problem, biased RRTs are proposed (Urmson and Simmons, 2003). Because these biased RRT algorithms are greedy, they tend to generate a local minimum solution.

One of the most recently published versions of RRT is IB-RTT* (Qureshi and Ayaz, 2015), which is specially designed for sophisticated cluttered environments. It uses the bidirectional tree technique and proposes an intelligent sample insertion heuristic to converge faster to the optimal solution. According to the provided experimental results, IB-RRT* shows superior computational efficiency in comparison with RRT and its variants. Although this is the fastest variant of RRT, it still needs considerable amount of time to compute a sufficient solution, and for some reported scenarios the running time exceeds 510 seconds. To reduce running time, IB-RRT* can present a local optimal solution which is relatively poor in smoothness and the path length, but by increasing the number of iterations (and thus running time) it improves the quality of the path.

2.3. Classic algorithms. The classic algorithms, also known as exact algorithms, are a group of algorithms that decompose the environment into different sections. A remarkable number of the studies in this group are an enhanced version of the preliminary motion planning algorithms such as the bug algorithm, potential fields (Ge and Cui, 2000), mathematical programming, and the geometry-based algorithm (Mac *et al*., 2016; Jafarzadeh *et al*., 2014). The presented algorithm is categorized in the geometry-based algorithms.

Bug algorithms assume that, although the agent knows the location of the global goal, only local information of the environment is available (Choset, 2005). Bug algorithms generate suboptimal (i.e., long) paths and have long running time. Another algorithm in this list is the potential field algorithm, which has the ability to generate a path in real time. However, the potential field algorithm cannot solve many complex scenarios, especially when there is a non-convex obstacle in the environment and either the starting or goal point is located within the convex hull of this obstacle (Tang *et al*., 2012). Also, the potential field algorithm does not generate a passage between very closely spaced obstacles. The other set of algorithms in this category is the mathematical programming approach which represents the requirement of obstacle avoidance and an objective function with a set of inequalities and equalities on the configuration parameters. MP is formulated then as a mathematical optimization problem that finds a curve between the start and goal points.

The last group of the classic algorithms in this classification are the geometry based algorithms. Because the SPP algorithm introduced in this paper is within this group, these algorithms warrant a relatively longer review. These algorithms try to provide a globally optimal path by generating a network of all possible paths between the start and goal points and then proceed with a shortest path algorithm over this network, such as Dijkstra's algorithm (Cormen, 2001). Because of the complexity of the generated network, the run time of these algorithms can be relatively long; however, the path length is generally globally optimal. There are four major algorithms in this sub-group: triangulation dual graph, generalized Voronoi diagram, cell decomposition, and visibility graph (VG). All the geometry based algorithms, also known as road map algorithms, assume that global knowledge of the environment is available. These algorithms try to generate paths by using sets of nodes and edges. Usually the graph is formed offline without information of the start and goal locations. Eventually, these locations are given as a query and some required edges are added to the graph, then the graph is searched to find the shortest path from the start to the goal point.

The first algorithm to be described in this group is the algorithm of triangulation dual graph paths (Choset, 2005), which decomposes the solution space environment into triangles, where each triangle composes a graph. Then the algorithm computes the dual graph of the original triangulated graph that generates a rough path. To do this, the algorithm finds the center of each triangle (an alternative method is finding the midpoint of triangulation edges) and connects each center if the triangles share an edge with each other. Then, one of the shortest path algorithms is utilized to find the best path within the designed graph. Sometimes, the generated path is infeasible and inefficient. To improve the given path, the algorithm applies a refining method that ensures the computed path will be acceptable and more efficient in terms of length.

A Voronoi road map (Choset, 2005) is a set of paths in the environment that ensures maximum clearance between obstacles. This method reduces the probability of the collision, and is thus preferred in robotics. A Voronoi cell is the set of points that are closer to an assumed or specified point (also called a seed) than to any of the other seed points. More formally, given a finite set of points $\{p_1, \ldots, p_n\}$, the Voronoi cell R_k associated with p_k of every point whose distance to p_k is less than or equal to its distance to any other p_i in the set. A generalized Voronoi diagram is designed for environments that are populated by polygons instead of points. After forming the Voronoi diagram, the desired graph can be processed by a shortest path algorithm. This algorithm generates a safe, obstacle-free path, which is suitable when the sensors on the robot include large errors, but it does not generate the shortest path.

The basic idea behind the cell decomposition method (Choset, 2005) is determining a path between the start and the goal configuration by dividing the obstacle-free space into smaller regions called cells. After this step, a connectivity graph is constructed according to the adjacency relationships between the cells, where the nodes represent the cells in the free space, and the edges between the nodes show that the corresponding cells are adjacent to each other. From this connectivity graph a path will be determined by following adjacent free cells from the start point to the goal. The final set of algorithms in the geometry based class are visibility graph techniques. The algorithms in this group are capable of finding an optimal path. Two arbitrary points a and b are visible to each other when the line segment $\overline{ab}$ does not intersect any polygonal obstacles (Latombe, 2012).

By connecting all mutually visible vertices, the visibility graph is constructed. According to the following lemma, the visibility graph includes the optimal path.

Lemma 1. (De Berg *et al.*, 2008) *Any shortest path between S and G among a set of polygonal obstacles is a polygonal path whose inner vertices are vertices of given obstacles.*

In this lemma, S and G indicate the starting point and the goal point, respectively. This lemma helps to shrink the solution space from continuous to discrete. According to the definition, the algorithms presented to construct the visibility graph take into account all the vertices of existing polygonal obstacles. In the literature, the effectiveness of algorithms in this class is often measured in terms of the time complexity in constructing the visibility graph.

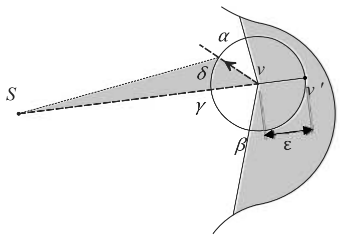
The simplest algorithm has $O(n^3)$ running time, where n is the number of all vertices in the given case (Lozano-Pérez and Wesley, 1979). This algorithm is a naive one, in which all the edges are drawn by using two arbitrary vertices and checked for the feasibility. In 1978, Lee presented an improved algorithm (Asano *et al.*, 1986) by applying the rotation sweep technique. In this algorithm vertices are sorted based on their angle relative an arbitrary point. The sorting operation can be completed in $O(n \log n)$ and after adding the visibility check of all n vertices, the running time will be $O(n^2 \log n)$. If cardinality of inter-polygonal edges, denoted by ε_ν is equal to $O(n^2)$, the overall running time after applying Dijkstra's algorithm will be $O(n^2 \log n)$.

Quadratic algorithms (Asano *et al.*, 1986; Edelsbrunner *et al.*, 1986) can construct the visibility graph in $O(n^2)$. In the case when the cardinality of ε_ν is $O(n^2)$, these algorithms work optimally with respect to the running time of Dijkstra's algorithm. Most of them utilize a triangulation over the obstacle-free space or mapping to the dual space and using the rotation sweep technique to check the feasibility of inter-polygonal edges.

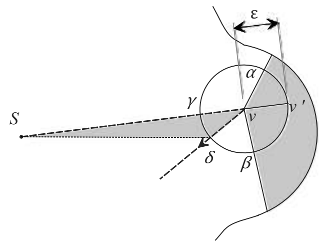
Another quadratic algorithm, which does not use the triangulation method and is relatively intuitive and easy to implement, was presented by Welzl (1985). This algorithm has been designed to construct the visibility graph for line segments that can be easily generalized to polygonal obstacles.

Finally, an output-sensitive algorithm proposed by Ghosh and Mount (1991) outperforms all previous algorithms when the cardinality of ε_ν is less than $O(n^2)$. The time complexity of this algorithm is $O(n \log n + \mathrm{card}(\varepsilon_\nu))$.

Rohnert introduced an improved version of the visibility graph called the reduced-visibility-graph algorithm that considers only the supporting and separating edges to find the shortest path (Rohnert, 1986).



(a)



(b)

Fig. 1. Vertex v is not a part of the shortest path.

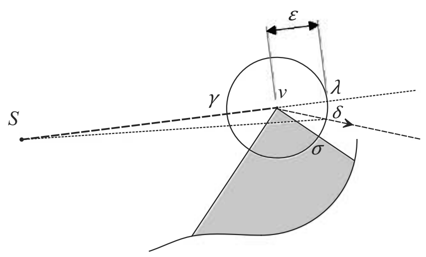


Fig. 2. Vertex v might be a part of the shortest path.

By applying this method, the number of inter-polygonal edges decreases significantly to the visibility graph. The main thought behind Rohnert's algorithm is computing the part of the visibility graph which plays a role in finding the shortest path in $O(n + p^2 \log n)$ time, where p is the number of polygons in the obstacle space. The following lemma ensures that the reduced visibility graph includes the optimal path.

Lemma 2. (Rohnert, 1986) *The shortest collision-free path between S and G among a set of polygonal obstacles, runs through edges of the polygons and supporting and separating segments of pairs of polygons.*

The number of edges created by the reduced visibility graph to find the shortest path is bounded by $O(p^2)$ which is less than $O(n^2)$. Unfortunately, this algorithm is limited to convex polygons and does not cover nonconvex polygons in $O(p^2)$ time complexity.

3. SPP algorithm description

As mentioned above, SPP is classified in the geometry based algorithms. Although the reduced visibility graph algorithm considers just the extreme vertices of the obstacles, it cannot handle nonconvex obstacles and in the convex polygonal obstacles it still uses all the obstacles in its calculations. It is the first algorithm that intelligently eliminates a significant part of the solution space while ensuring the optimal solution is not in the eliminated set. However, not all obstacles in an environment will necessarily play a role in constructing the optimal path. For example, consider a sparsely populated environment, i.e., one with numerous obstacles that are far from each other and the line between the start and goal points $\overline{SG}$. Why would a path planning algorithm need to consider such obstacles?

Therefore, an algorithm is needed to ignore those obstacles from its calculations. SPP overlooks those irrelevant obstacles by cleverly constructing the network of the polygonal paths so that the size of the network significantly decreases in terms of the number of nodes and the number of edges. We introduce *effective polygons* as a new term in this paper and define them as follows.

Definition 1. *Effective polygons* are those whose vertices (and edges) are contained in the optimal path from S to G. Polygons that are not included in the optimal path are called *ineffective polygons*.

SPP attempts to ignore the ineffective obstacles in its path finding process. There is currently no guarantee that SPP can eliminate all of the ineffective polygons for every scenario, but it shows significantly improved performance in a variety of environments, particularly those that do not have a dense population of obstacles. The fundamental idea behind the SPP algorithm is considering just the obstacles that block its way toward G. Before starting the path planning, to release the shape of the agent (i.e., a disk shape) the obstacles are expanded as much as the size of the agent's radius and then the algorithm considers it as a point.

First, SPP checks the shortest path from S to G, which is straight line $\overline{SG}$. If it is feasible the algorithm returns it as the optimal path, otherwise, according to Lemma 1, it uses the vertices of the blocking polygonal obstacles (i.e., either convex or nonconvex) to construct its path to reach the goal point. Lemma 3 gives a detailed instruction to find the desirable vertices in convex and nonconvex polygons (although, there is a lemma in (Rohnert, 1986) which helps to find the end-points and delete the other vertices, it is limited to convex polygons and is not applicable to the nonconvex ones).

Lemma 3. *The shortest collision-free path between S and G among a set of convex and nonconvex polygonal obstacles does not run through any vertex v such that an arbitrary prolongation of line segment $\overline{Sv}$ lies inside a polygon.*

Proof. If line segment $\overline{Sv}$ goes through a polygon, then this is an infeasible segment and should be ignored in constructing the network. Assume that the line segment $\overline{Sv}$ is feasible and vertex v is a part of an optimal path, but its prolongation, which is shown by line segment $\overline{vv'}$, enters a polygon (see Fig. 1).

Because $\overline{Sv}$ is feasible, there is a circle with radius $\varepsilon > 0$ (i.e., an arbitrarily small number) at v such that the line segment starting from S and ending on arc $\alpha\gamma\beta$ is also feasible. Assume that v is not the goal point and should be left, and recall that we know that line segment $\overline{vv'}$ is an infeasible direction. The only feasible exit region from vertex v lies along arc $\alpha\gamma\beta$. The leaving ray intersects the imaginary circle at a point called δ anywhere along the arc $\alpha\gamma\beta$. Constructed the triangle $Sv\delta$ and note that $|\overline{v\delta}| = \varepsilon$. By the triangle inequality, $|\overline{S\delta}| < |\overline{Sv}| + |\overline{v\delta}|$ and therefore $\overline{S\delta}$ represents a shortcut from the original path through v and v is not on the optimal path.

On the other hand, if prolongation $\overline{Sv}$ does not enter the polygon, then there would exists an arc such as $\sigma\delta\lambda$ (in Fig. 2) for a ray leaving v in which the shortcut $\overline{S\sigma}$ is infeasible. The region of rays going from v through this arc can be a part of an optimal path. ■

The following pseudocode applies Lemma 3 to find the required vertices in constructing the optimal path. To do this, based on the location of the current point, goal point, and shape of the obstacle, the algorithm constructs the network with the encountered polygonal obstacle.

This algorithm is invoked by command $f(s', M_i^j)$ in the main SPP algorithm, in which s' is the current point and M_i^j shows the set of vertices of the j-th obstacle that makes the i-th path infeasible and the desirable output would be the set of acceptable vertices (i.e., V) in M_i^j and are accessible from s'. Wherever the SPP algorithm confronts this command, this means that the first algorithm will take the vertices of the j-th obstacle from set M_i (i.e., the set of obstacles that make the i-th path infeasible) and s' and returns acceptable vertices V.

Algorithm 1. Finding acceptable vertices ($f(s', M_i^j)$).

```
1: $V \leftarrow \varnothing$
2: for $\{\forall v_k | v_k \in M_i^j\}$ do
3:     calculate $v'_k$
4:     if $\overline{s'v'_k} \in O_{\text{free}}$ then
5:         $V \leftarrow V \bigcup v_k$
6:     end if
7: end for
8: return $V$
```

In the given pseudocode O_{free} represent the space free of obstacles. In the first step, the algorithm checks for two end-points (e_u and e_l) in the obstacle which are called *extrema* in this work. These extrema are the two outermost tangential points of the obstacle identified by the smallest two-dimensional cone that includes all of the encountered obstacles with the cone origin set on the current location S. If there exists a cone starting from point S (i.e. $\mathrm{card}(V) = 2$), again based on Lemma 1 of Rohnert (1986), both of *extrema* which are constructing the *supporting* line segments will be returned as the acceptable vertices and are enough to find the optimal path. If these points can be found by the algorithm, they are called e_u and e_l and saved in set V, which will be returned as a desired output. However, in cases where the start point is located in the inner section of spiral polygon, this cone cannot be drawn from point S (i.e., $\mathrm{card}(V) \neq 2$). Therefore, in this case we need to take into account the other vertices to establish the network of the path to reach the goal (see Fig. 3).

According to Lemma 3, the algorithm should not consider all of the vertices in the polygon to find the optimal path. If the prolongation of the connecting line S and v_k enters into the polygonal obstacle, then this vertex is not a part of the optimal path from S to G. By this method, most of the vertices are eliminated and the size of the network is decreased significantly. For instance, if line segment ($\overline{Sv_1}$) is prolonged as much as ε (where ε is an arbitrarily small number) to obtain a new point (v_1') line segment ($\overline{Sv_1'}$) will be infeasible, because (v_1') is inside of the polygon. This vertex will not be in the optimal path from S to G, and the only vertex should be considered in the current position S is v_6.

Lemma 4. *In a polygonal obstacle with number n_k vertices, finding the vertices such that the prolongation of the line segment starting from S is feasible takes $O(nn_k)$.*

Proof. According to the pseudocode given in Algorithm 1, for each vertex v_k the algorithm finds v_k' which takes $O(1)$, then it checks if line segment $\overline{Sv_k'}$ is feasible or not in time $O(n)$, where n is all the vertices. The algorithm repeats this step for all vertices of the k-th polygon, i.e., n_k. Altogether, the time complexity for this part of the algorithm is $O(nn_k)$. ■

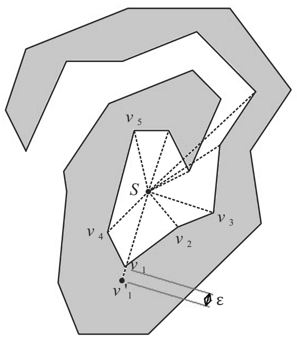


Fig. 3. Nonconvex polygon with no extrema from point S.

There is provided pseudocode for SPP in Algorithm 2.

Algorithm 2. SPP algorithm.

```
1: $\rho \leftarrow \langle S \rangle$
2: $\xi \leftarrow \langle S, G \rangle$
3: **while** $\{\exists k | \neg\rho^k \mathbf{or} \rho^k_{\|\rho^k\|} \neq G\}$ **do**
4:    calculate $\xi$
5:    **if** $\|\xi\| \rangle 0$ **then**
6:       **for** $i = 1$ to $\|\xi\|$ **do**
7:          calculate $M_{\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle}$
8:          delete $\xi^i$ from $\rho$
9:          **for** $j = 1$ to $\|M_{\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle}\|$ **do**
10:            $\theta \leftarrow f(\xi^i_{\|\xi^i\|-1}, M^j_{\langle \xi^i_{q_i-1}, \xi^i_{q_i} \rangle})$
11:            **if** $\langle\langle \xi^i - \xi^i_{\|\xi^i\|} \rangle, \theta >\notin \rho$ **then**
12:               $\rho \leftarrow \langle \rho, \langle\langle \xi^i - \xi^i_{\|\xi^i\|} \rangle, \theta \rangle\rangle$
13:            **end if**
14:          **end for**
15:       **end for**
16:       **go to** 4
17:    **else**
18:       **for** $q = 1 : \|\rho\|$ **do**
19:          $\rho^q \leftarrow \langle \rho^q, G \rangle$
20:       **end for**
21:    **end if**
22: **end while**
```

What follows is a brief explanation of the given pseudocode. First, we define two factors ξ and ρ. ξ is an n-tuple of an ordered list of infeasible paths and ρ is another n-tuple of paths that SPP tries to make them all feasible and this pseudocode returns that as its output. At the beginning, SPP assumes that the connecting line from its current position S to the goal point G is infeasible and saves it in ξ. It is a repetitive procedure that will continue till finding an n-tuple ρ of feasible paths that ends into G. If n-tuple ρ has at least one infeasible path (i.e., $\neg\rho^k$ is true) or the last point of at least one path in n-tuple ρ is not the goal point (i.e., $\rho^k_{\|\rho^k\|}$), it will repeat the "while" loop to satisfy both of the conditions. Here ρ^k shows the k-th path in n-tuple ρ, and $\rho^k_{\|\rho^k\|}$ represents the last point of the path ρ^k where $\|\rho^k\|$ is the number of the points constructing the path ρ^k.

Within this loop, the algorithm finds the infeasible paths and saves them as ξ, if this n-tuple is empty ($\|\xi\|= 0$) and all paths are feasible it adds the goal point to the end of each paths in n-tuple ρ and checks the twofold conditions again, but if $\|\xi\| > 0$, for each of the paths in ξ the algorithm will calculate an alternative path. Because the algorithm corrects the infeasibility of the paths in each step, then the origin of the infeasibility is the last point added to the existing path. The line segment constructed by the last point and the point before that makes the path infeasible (i.e., $\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle$). Accordingly, at the first step, it calculates the set of obstacles that make the assumed line segment infeasible, and saves them as $M_{\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle}$ and deletes them from ρ. Now, for each of these obstacles (i.e., $\|M_{\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle}\|$ is the number of these obstacles), the algorithm invokes the first algorithm by command $f(\xi^i_{\|\xi^i\|-1}, M^j_{\langle \xi^i_{q_i-1}, \xi^i_{q_i} \rangle})$ to calculate the acceptable vertices from the current point $\xi^i_{\|\xi^i\|-1}$ and substitutes it. The algorithm checks whether or not the generated path $\langle \langle \xi^i - \xi^i_{\|\xi^i\|} \rangle, \theta \rangle$ is already in the current n-tuple ρ. If it is repetitive, it deletes them from the process to make the performance of the algorithm better.

After constructing the network of the paths, the SPP algorithm chooses the shortest path from the n-tuple ρ. Then to make the output path applicable for navigation, it will be smoothed by a given algorithm at the end. All the obstacles participating in constructing this network are *effective polygons* which can obtained by uniting $M_{\langle \xi^i_{\|\xi^i\|-1}, \xi^i_{\|\xi^i\|} \rangle}$ in each iteration of the algorithm. The obtained set is not necessarily equal to the whole set of obstacles and the SPP algorithm just considers those obstacles that blocks its way toward the goal point. By this approach it skips *ineffective obstacles* and saves time, especially in the environments in which obstacles are located far from each other.

Lemma 5. *The SPP algorithm can find the shortest collision-free path from starting point S to goal point G in the presence of convex and nonconvex polygonal obstacles in $O(nn'^2)$ time, where n is the number of all vertices and n' is the number of vertices of the polygons that construct the path network.*

Proof. The SPP algorithm draws line segment $\overline{v_{ij}G}$ from the i-th vertex of the j-th *effective* polygon toward the goal point to find the other effective polygons and checks for feasibility, which takes $O(nn')$. Also, according to Lemma 4, it finds the required vertices from each effective polygon in time $O(nn')$. The algorithm completes these steps for all of the vertices in effective polygons, and the overall time complexity for constructing the network of paths is $O(n'(nn'+nn'))$, which yields $O(nn'^2)$. Finally, to find the shortest path in the constructed network, Dijkstra's algorithm is applied over the n' graph, resulting in $O(nn'^2+n'\log n')$ complexity and reduces to $O(nn'^2)$. ■

Note that for all cases, we have $n' \leq n$. In many environments, including the scenarios presented in Section 4 (Figs. 5 and 6), $n' \ll n$. The result, as will be seen in following sections, is that in practice SPP executes very fast. In the environments with sparsely distributed polygonal obstacles in which $n' \ll n$, the time complexity of the SPP algorithm becomes linear in n, but the most effective given algorithm for this problem has quadratic (i.e., the cardinality of the number of edges) or in the best case has quasilinear time complexity.

After generating $[\rho]$, a shortest path algorithm (e.g., Dijkstra's algorithm) is applied to find the shortest path between the start and the goal points among matrix $[\rho]$. Finally, this path is smoothed for navigation. There are many metrics for smoothness, but the overarching goal of this smoothing function is to limit the sharp discontinuities that exist between two segments of the path generated by SPP. Consider a path $R-P-Q$, i.e., a path that begins at point R, makes a discontinuous turn at point P, and then ends at point Q. The function selects a point to begin a "turn" some distance before the discontinuity (T_1), another point to exit the turn after the discontinuity and begin traveling on a straight line again (T_2), and a turning radius with which to complete the turn ω. See Fig. 4 for an example of this geometry.

The smoothing algorithm selects T_1 and T_2 such that they are equidistant from the original, discontinuous turning point, P. For conservatism, this distance d is selected as $d = \min\{d_t, d_1, d_2, d_o\}$, where d_t is a user-selected parameter that represents the desired distance to begin the turn, d_1 is the distance of the path segment entering the turn generated by SPP (i.e., the path from R to P), d_2 is the distance of the path segment exiting the turn generated by SPP (the path from P to Q), and d_o is the distance necessary to ensure that the smoothed path remains obstacle free. The turning radius is then simply $\omega = d/\tan(\theta)$. The turn from T_1 to T_2 sweeps an arc of radius ω through an angle 2θ. The algorithm identifies the desired direction of turn and computes the center of curvature and the sweep angle accordingly, based on the geometry presented in Fig. 4.

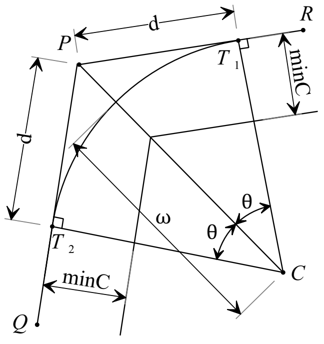


Fig. 4. Geometry used in the smoothing algorithm.

The resulting arc is continuous but can be discretized according to user preferences. For the simulations in the next section, each smoothed turn is discretized into 50 equidistance points between T_1 and T_2.

The smooth path is guaranteed to be obstacle free because of the buffer added to each obstacle. However, in the form presented in Fig. 4 it is not guaranteed to remain outside of the buffer defined by $\min C$. Our implementation includes the option for an additional buffer, ε, at the turning point in the direction of vector CP. With sufficient ε, the new path from $R-(P+\varepsilon)-Q$ can be guaranteed to lie on our outside of the area defined by $\min C$.

To evaluate the smoothness of the generated path, a new evaluation measure called ADCP (average degree of discontinuous connection point) has been used (Suzuki *et al.*, 2009). ADCP is calculated by the sum of the discontinuous angles divided by the count of discontinuous events. Therefore, the value of the smoothest path becomes zero in this measure. The formula of the smoothness evaluation ADCP is

$$\text{Smoothness} = \frac{1}{D_C}\sum_{i=S}^{G} D_A^i, \tag{1}$$

where D_A^i is the discontinuity angle of the i-th turn, D_C discontinuity counts, $i = S$ represents the start of the path, and G the goal.

4. Results and discussion

This section provides different scenarios to evaluate the performance of the SPP algorithm relative to the heuristic and probabilistic algorithms. Since the presented algorithm is exact, it deterministically generates a single solution for the same scenario in different runs. We used MATLAB as a platform to implement the SPP algorithm.

To validate the SPP algorithm, we compare its performance with the genetic algorithm from the heuristic algorithms, the rapidly-exploring random tree (RRT) and the probabilistic road map (PRM) from the probabilistic algorithms. Because the time complexity of the SPP algorithm has been presented in Lemma 5 and it is available for other classic algorithms in the literature, the running time of this group of algorithms can be compared based on their time complexities. We therefore do not consider this group in the simulations. For the other two groups (i.e., heuristic and probabilistic algorithms) we chose GAMOPP (multi-objective path planning) (Davoodi *et al.*, 2015), RRT (Kala, 2014b) and PRM (Kala, 2014a).

GAMOPP represents a state-of-the-art path planning algorithm and has been published recently in the literature. Also, in the GAMOPP paper, the authors have compared the performance of their algorithm with two other previously presented powerful algorithms, i.e., the improved strength Pareto evolutionary algorithm (SPEA2) (Zitzler *et al.*, 2001) and multi-objective particle swarm optimization (MOPSO) (Zhang *et al.*, 2013; Coello *et al.*, 2004), and they show that their algorithm outperforms those. SPEA2 is an effective algorithm and several evolutionary algorithms have compared their results with this algorithm in the literature (e.g., Deb, 2001). Particle swarm optimization (PSO) is a swift and simple randomized search algorithm applied to optimize numerous NP-hard problems, and by now, several versions of multi-objective PSO have been proposed (Coello *et al.*, 2004; 2007). One application of this algorithm is in the path planning problems (Zhang *et al.*, 2013; Purcaru *et al.*, 2013).

Because GAMOPP, MOPSO, and SPEA2 are heuristic algorithms, they produce a different solution for the same problem for each run. They thus perform their algorithm several times and study its results using statistical tools. Alternatively, SPP produces a single path for each problem. Davoodi *et al.* (2015) highlight that sometimes MOPSO is not able to find any feasible solution in 100 generations, especially when there is a narrow passage framework, or numerous obstacles have cluttered the environment, but GAMOPP always provides a feasible solution per each run. Therefore, we select GAMOPP presented by Mohanta *et al.* (2011) and

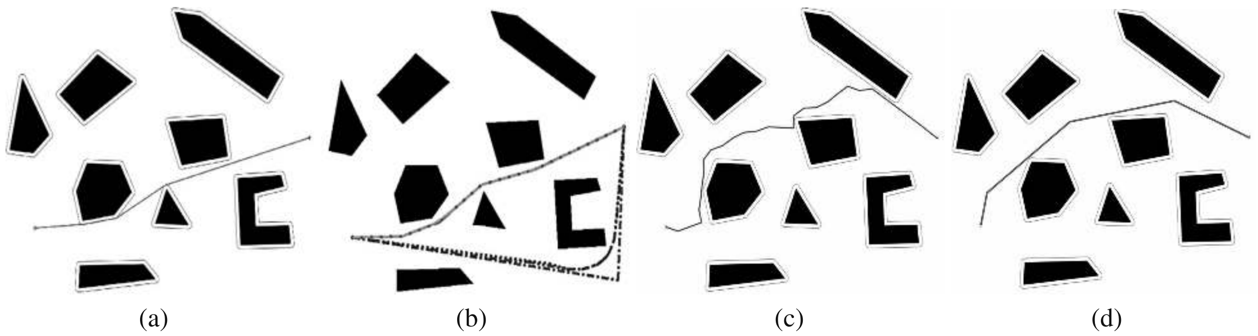


Fig. 5. Standard framework ($\min C = 10$): SPP (a), GAMOPP (b), RRT (c), PRM (d).

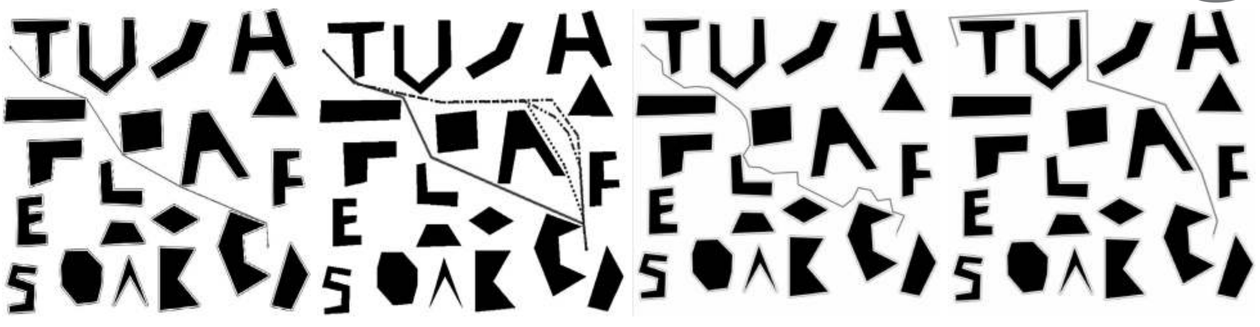

(a) (b) (c) (d)

Fig. 6. Clutter space ($\min C = 4$): SPP (a), GAMOPP (b), RRT (c), PRM (d).

compare its results with SPP in terms of running time and quality of solutions.

Being probabilistic, RRT and PRM have the same problem as GAMOPP and they cannot find a feasible solution for the given problem all the time. To solve this problem, we increased the number of randomly chosen points from the environment in PRM and the number of trials in RRT. Although this approach increases the running time in PRM, it improves the likelihood of achieving a feasible solution. Increasing the number of trials will not increase the running time in RRT, because whenever it reaches the goal point it stops but it affects the likelihood of producing a feasible solution. Accordingly, these factors were adjusted such that the probability of generating a feasible solution in both algorithms would be 90%.

To have a clear comparison between these algorithms, we use the same test problems from the work of Davoodi *et al.* (2015) to evaluate the performance of the SPP algorithm. These four cases examine the capabilities of each algorithm under difference scenarios. Figure 5 shows the paths produced by GAMOPP.

GAMOPP attempts to optimize a multi-objective function with three output parameters. The first function is concerned with minimizing the length of the path, the next two functions maximize the smoothness of the path and the last two terms try to maximize the clearance of the path. GAMOPP seeks to maximize the minimum distance of the path from the closest obstacle and also the minimum distance from the nearest obstacle along the path. Alternatively, the SPP algorithm adds the desirable distance of the agent from each obstacle in the first step and substitutes the expanded obstacles in its calculation. This factor is accounted for, which allows for a comparison with respect to the length and smoothness of the paths.

In the SPP, RRT and PRM algorithms we have extended the polygons as much as the desired distance value and considered these extended polygons as the obstacles that the paths need to avoid.

The first scenario in this group is called the *standard framework* (Fig. 5). The *desired clearance* ($\min C$) for this case has been set at 10. The expanded obstacles around the real obstacles (filled with dark color) show the virtual obstacles that involve $\min C$. The output of GAMOPP for different runs has been shown.

The number of randomly generated points in PRM was set at 50 and the size of each step for RRT was set to 20.

Three other scenarios that have been provided to challenge the effectiveness of the presented SPP algorithm are used to evaluate its performance. The four algorithms

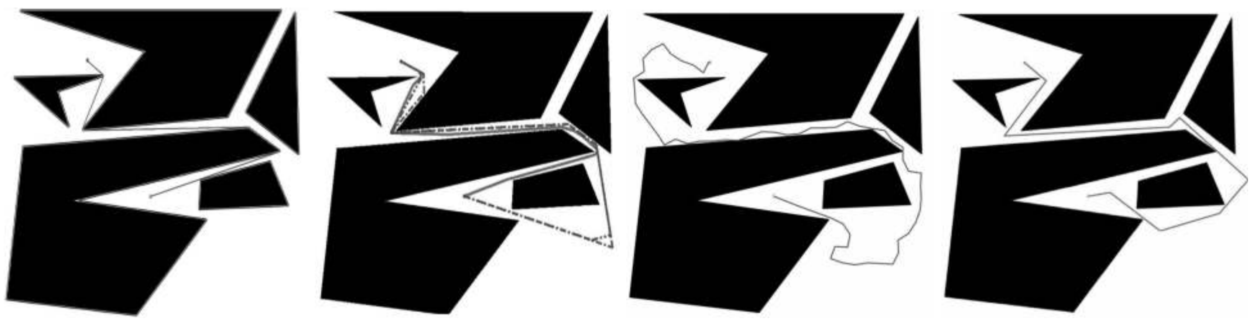

(a) (b) (c) (d)

Fig. 7. Narrow passage (corridor) ($\min C = 3$): SPP (a), GAMOPP (b), RRT (c), PRM (d).

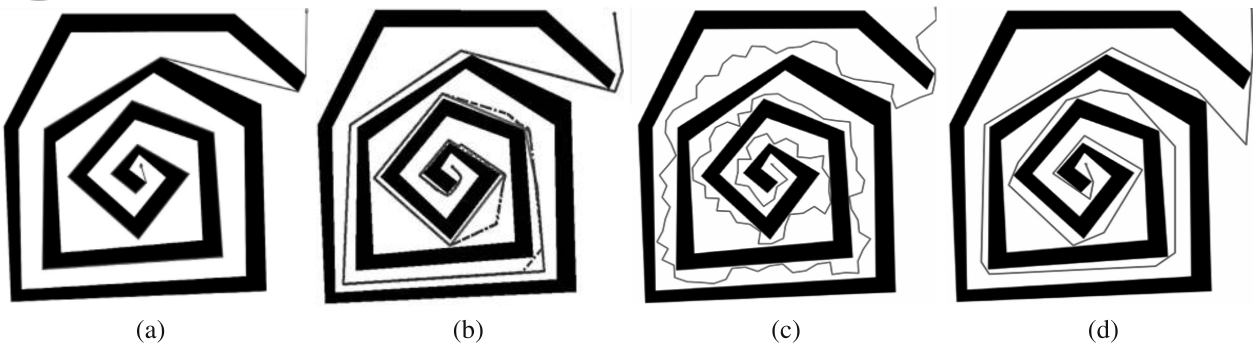


Fig. 8. Spiral shape ($minC = 0$): SPP (a), GAMOPP (b), RRT (c), PRM (d).

are compared across several scenarios, and their results are summarized in Table 1 in terms of running time, length and smoothness of the generated paths. The second scenario is called the *clutter space* and 20 convex and nonconvex obstacles have been distributed randomly in it (Fig. 6). This is a challenging case, because it contains more obstacles that the other scenarios since there are numerous local optima for this scenario. The minimum clearance for this case is set to 4.

To make sure that algorithms can handle a case with a narrow passage, the scenario shown in Fig. 7 is used. Here $\min C$ is 3, resulting in a very narrow clearance between the virtual expanded obstacle and the real obstacle. The last scenario is a spiral obstacle (Fig. 8). For this case $\min C$ is set to zero, so the algorithms generate paths relative to the real obstacle.

Three outputs with different results have been presented by GAMOPP for this case but we considered the best running time and path length in the Table 1.

5. Conclusions and future work

In this paper, a novel algorithm, which is called SPP, has been presented for solving path-planning problems in a static environment with convex and nonconvex obstacles.

Although SPP is presented for static environments and this version does not address unknown and dynamic areas, being fast it can applied on different screen shots of environment repetitively to handle them easily and in real time.

The SPP algorithm is classified as a geometry based algorithm in which the first step is coming up with a network of the segment lines that connect the vertices of obstacles and location of the start and goal points. The visual graph and the improved version of this algorithm, RVG—for a reduced visibility graph, generate this graph of possible paths for this problem, but the advantage of SPP over existing techniques is shrinking the size of this network as much as possible while preserving optimality conditions. It was proven mathematically that the SPP algorithm generates an optimal path and

Table 1. Comparing the results of the SPP, GAMOPP, RRT and PRM algorithms.

Scenarios	Algorithm	Clearance	Smoothness	Path length	Running time [s]
Standard framework	SPP	**10**	**0.41**	**818.4**	**0.01**
	GAMOPP	10.24	12.60	924.8	1.5
	RRT	10	33.42	1353.1	6.23
	PRM	10	35.1	1238.4	1.04
Clutter space	SPP	**4**	**0.98**	**1541.3**	**0.67**
	GAMOPP	4.49	18.61	1613	4
	RRT	4	35.78	1944.6	5.33
	PRM	4	59.61	2220.7	1.93
Narrow passage	SPP	**3**	**2.15**	**2034.9**	**0.02**
	GAMOPP	**3**	29.97	2201	1
	RRT	3	29.37	4292.2	8.94
	PRM	3	70.68	3315.6	2.1
Spiral shape	SPP	**0**	**1.65**	**3729.8**	**0.13**
	GAMOPP	**0**	40.49	4147	3
	RRT	0	46	7017.7	25.81
	PRM	0	54.73	4424.8	53.21

the time complexity of this algorithm was calculated as $O(nn'^2)$. The SPP algorithm has the capability of discerning the ineffective polygon and eliminating them from its calculations.

In a densely populated environment, in which $n' > \sqrt{n}$, the advantages of SPP over the other algorithms are somewhat minimized. As seen in Table 1, SPP still outperforms the other algorithms. However, in the absolute worst-case environment, i.e., if all polygons become so-called effective polygons, SPP has $O(n^3)$ time complexity.

To evaluate the effectiveness of SPP, we selected one from each of path planning algorithm groups and compared the results. The selected algorithm from the heuristic algorithms is the genetic algorithm for multi-objective path planning (GAMOPP) which outperforms two other state-of-the-art path planning algorithms (i.e., improved strength pareto evolutionary algorithm (SPEA2) and multi-objective particle swarm optimization (MOPSO)). The selected algorithms from the probabilistic algorithms are the rapidly exploring random tree (RRT) and the probabilistic road map (PRM). The provided results show that SPP algorithms generated better solutions in terms of running time as well as length and smoothness of the path. In some cases GAMOPP, RRT and PRM could not find a feasible solution, but as long as there is a feasible solution SPP is able to give a feasible and optimal solution. The SPP algorithm has been tested by scenarios designed for different purposes, and its results have been presented in tabular format as well as graphically. This algorithm generates the shortest paths for different scenarios.

The future work would be exploring this class of algorithms, which emphasize on distinguishing and eliminating the redundant polygons from visibility graph algorithms in path planning.

References

Akbaripour, H. and Masehian, E. (2017). Semi-lazy probabilistic roadmap: A parameter-tuned, resilient and robust path planning method for manipulator robots, *International Journal of Advanced Manufacturing Technology* **89**(5–8): 1401–1430.

Asano, T., Asano, T., Guibas, L., Hershberger, J. and Imai, H. (1986). Visibility of disjoint polygons, *Algorithmica* **1**(1): 49–63.

Bohlin, R. and Kavraki, L.E. (2000). Path planning using lazy PRM, *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA'00, San Francisco, CA, USA*, Vol. 1, pp. 521–528.

Choset, H.M. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation*, MIT Press, Cambridge, MA.

Coello, C.A.C., Pulido, G.T. and Lechuga, M.S. (2004). Handling multiple objectives with particle swarm optimization, *IEEE Transactions on Evolutionary Computation* **8**(3): 256–279.

Coello, C.C., Lamont, G.B. and Van Veldhuizen, D.A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*, Springer, New York, NY.

Cormen, T.H. (2001). *Introduction to Algorithms*, MIT Press, Cambridge, MA, pp. 595–601.

Davoodi, M., Panahi, F., Mohades, A. and Hashemi, S.N. (2015). Clear and smooth path planning, *Applied Soft Computing* **32**: 568–579.

De Berg, M., Cheong, O., Van Kreveld, M. and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*, Springer-Verlag TELOS, Santa Clara, CA.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*, Wiley, New York, NY.

Edelsbrunner, H., Guibas, L.J. and Stolfi, J. (1986). Optimal point location in a monotone subdivision, *SIAM Journal on Computing* **15**(2): 317–340.

Ge, S.S. and Cui, Y.J. (2000). New potential functions for mobile robot path planning, *IEEE Transactions on Robotics and Automation* **16**(5): 615–620.

Ghosh, S.K. and Mount, D.M. (1991). An output-sensitive algorithm for computing visibility graphs, *SIAM Journal on Computing* **20**(5): 888–910.

Jafarzadeh, H., Gholami, S. and Bashirzadeh, R. (2014). A new effective algorithm for on-line robot motion planning, *Decision Science Letters* **3**(1): 121–130.

Jafarzadeh, H., Moradinasab, N. and Elyasi, M. (2017). An enhanced genetic algorithm for the generalized traveling salesman problem, *Engineering, Technology & Applied Science Research* **7**(6): 2260–2265.

Kala, R. (2014a). Code for robot path planning using probabilistic roadmap, Indian Institute of Information Technology, Allahabad, `http://rkala.in/codes.php`.

Kala, R. (2014b). Code for robot path planning using rapidly-exploring random trees, Indian Institute of Information Technology, Allahabad, `http://rkala.in/codes.php`.

Kavraki, L.E., Kolountzakis, M.N. and Latombe, J.-C. (1998). Analysis of probabilistic roadmaps for path planning, *IEEE Transactions on Robotics and Automation* **14**(1): 166–171.

Klaučo, M., Blažek, S. and Kvasnica, M. (2016). An optimal path planning problem for heterogeneous multi-vehicle systems, *International Journal of Applied Mathematics and Computer Science* **26**(2): 297–308, DOI: 10.1515/amcs-2016-0021.

Latombe, J.-C. (2012). *Robot Motion Planning*, Springer, New York, NY.

LaValle, S.M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*, Iowa State University, Ames, IA.

Liu, J., Yang, J., Liu, H., Tian, X. and Gao, M. (2017). An improved ant colony algorithm for robot path planning, *Soft Computing* **21**(19): 5829–5839.

Lozano-Pérez, T. and Wesley, M.A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles, *Communications of the ACM* **22**(10): 560–570.

Mac, T.T., Copot, C., Tran, D.T. and De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey, *Robotics and Autonomous Systems* **86**: 13–28.

Masehian, E. and Sedighizadeh, D. (2007). Classic and heuristic approaches in robot motion planning—a chronological review, *World Academy of Science, Engineering and Technology* **23**(5): 101–106.

Mohanta, J.C., Parhi, D.R. and Patel, S.K. (2011). Path planning strategy for autonomous mobile robot navigation using Petri-GA optimisation, *Computers & Electrical Engineering* **37**(6): 1058–1070.

Ni, J., Wu, L., Shi, P. and Yang, S. X. (2017). A dynamic bioinspired neural network based real-time path planning method for autonomous underwater vehicles, *Computational Intelligence and Neuroscience* **2017**, Article ID: 9269742.

Purcaru, C., Precup, R.-E., Iercan, D., Fedorovici, L.-O. and David, R.-C. (2013). Hybrid PSO-GSA robot path planning algorithm in static environments with danger zones, *Proceedings of the 17th International Conference System Theory, Control and Computing (ICSTCC), Sinaia, Romania*, pp. 434–439.

Qureshi, A.H. and Ayaz, Y. (2015). Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments, *Robotics and Autonomous Systems* **68**(6): 1–11.

Rohnert, H. (1986). Shortest paths in the plane with convex polygonal obstacles, *Information Processing Letters* **23**(2): 71–76.

Suzuki, Y., Thompson, S. and Kagami, S. (2009). Smooth path planning with pedestrian avoidance for wheeled robots: Implementation and evaluation, *4th International Conference on Autonomous Robots and Agents, ICARA 2009, Wellington, New Zealand*, pp. 657–662.

Tang, S., Khaksar, W., Ismail, N. and Ariffin, M. (2012). A review on robot motion planning approaches, *Pertanika Journal of Science and Technology* **20**(1): 15–29.

Urmson, C. and Simmons, R. (2003). Approaches for heuristically biasing RRT growth, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), Las Vegas, NV, USA*, Vol. 2, pp. 1178–1183.

Welzl, E. (1985). Constructing the visibility graph for n-line segments in $O(n^2)$ time, *Information Processing Letters* **20**(4): 167–171.

Zhang, Y., Gong, D.-W. and Zhang, J.-H. (2013). Robot path planning in uncertain environment using multi-objective particle swarm optimization, *Neurocomputing* **103**: 172–185.

Zitzler, E., Laumanns, M. and Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm, *Working paper*, ETH Zürich, Zürich.

Hassan Jafarzadeh received his BS and MSc, both in industrial engineering, from Tabriz University and the K.N. Toosi University of Science and Technology, respectively. He is currently a PhD candidate with the Systems and Information Department at the University of Virginia. His research interests include connected and autonomous vehicles, computational geometry, and optimization models and methods.

Cody H. Fleming joined the Department of Systems and Information Engineering, University of Virginia, in 2015. He received his doctoral degree in aeronautics and astronautics at the Massachusetts Institute of Technology, where he focused on formally and rigorously integrating safety analysis into early concept development of complex systems. He holds a Bachelor's degree in mechanical engineering from Hope College with honors (*summa cum laude*) and a Master's degree from MIT. Prior to getting his doctorate, he spent 5 years working in space system development for various satellite and laser projects, specializing in dynamics, design, and systems integration. His research interests are in methods for development and verification of safety-critical systems, particularly those with high levels of automation, aside general interests in dynamic systems and control.

Received: 12 July 2017
Revised: 6 January 2018
Re-revised: 24 February 2018
Accepted: 9 April 2018