Combining Voronoi Graph and Spline-Based Approaches for a Mobile Robot Path Planning

Evgeni Magid[1(✉)], Roman Lavrenov[1], Mikhail Svinin[2], and Airat Khasianov[3]

[1] Intelligent Robotics Department, Higher School of Information Technology and Information Systems, Kazan Federal University, Kremlyovskaya str. 35, Kazan, Russian Federation
{magid,lavrenov}@it.kfu.ru
http://kpfu.ru/robolab.html

[2] Robot Dynamics and Control Laboratory, College of Information Science and Engineering, Ritsumeikan University, Kyoto, Japan
svinin@fc.ritsumei.ac.jp

[3] Higher School of Information Technology and Information Systems, Kazan Federal University, Kremlyovskaya str. 35, Kazan, Russian Federation
ak@it.kfu.ru

Abstract. Potential function based methods play significant role in both global and local path planning. While these methods are characterized with good reactive behaviour and implementation simplicity, they suffer from a well-known problem of getting stuck in local minima of a navigation function. In this paper we propose a modification of our original spline-based path planning algorithm for a mobile robot navigation, which succeeds to solve local minima problem and considers additional criteria of start and target points visibility to help optimizing the path selection. We apply a Voronoi graph based path as an input for iterative multi criteria optimization algorithm and present a path finding strategy within different homotopies that uses the new method. The algorithm was implemented in Matlab environment and demonstrated significantly better results than the original approach. The comparison was based on success rate, number of iterations and running time of the algorithms. In total, several thousands tests were performed in 18 different simulated environments.

Keywords: Robotics · Path planning algorithm · Voronoi diagram · Potential field · Mobile robot · Simulated experiments · MATLAB

O. Gusikhin and K. Madani (Eds.): ICINCO 2017, LNEE 495, pp. 475–496, 2019.
https://doi.org/10.1007/978-3-030-11292-9_24

1 Introduction

Today robotic applications are targeting for industrial production speed increase and quality improvement as well as for human replacement in various scenarios, which range from social-oriented human-robot interaction [16] to dangerous for a human urban search and rescue scenarios [15]. The later may include indoor and outdoor environments and require autonomous navigation efficiency within unknown GPS-denied environments [20], ability to deal with computational complexity of simultaneous localization and mapping (SLAM) [2], capabilities of negotiation and collaboration with other robots within a team [17] and other functionality.

Path planning is the most essential part of autonomous navigation for a mobile robot. A mobile vehicle should plan its movement within multiple tasks, including localization, mapping and manipulation. A good path planning algorithm should be always capable of finding a route when it exists (or otherwise, display a message that the path does not exist). The classical global path planning approach, which is usually referred as *piano movers problem*, utilizes complete *a priori* knowledge about environment. In such algorithms we could use information about robot shape, initial and target pose (including position and orientation), and information about a set of 2D or 3D environment obstacles with their shapes, positions, orientations in space and other task-related data (e.g., texture or traversability index [18]). Next, the robot searches for a continuous path from the initial pose to the goal pose while avoiding static obstacles. In order to simplify the procedures, often the concept of multi-dimensional configuration space [10] is applied for the planning. In such settings, the path planning operation becomes an off-line one-time action since a complete knowledge about the environment is available in advance.

One of classical and popular path planning approaches is based on an artificial potential field concept, which is referred as potential field method [8,11]. Within this approach a goal position simulates an attractive pole while obstacles are represented with repulsive surfaces. Then a robot navigates with the potential gradient in the direction of its minimum. Potential field could be generated either at a global or a local level, and this is purely a matter of available information about the environment [1]. Potential field methods are popular because of simplicity and a stationary and mobile obstacles fast reactive avoidance capability. Unfortunately, potential field methods are typically featured with such drawbacks as path oscillations for certain configurations of obstacles (e.g., narrow passages) and local minima problem, which captures a robot in a local minima of a potential function and requires a special escape procedure in order to proceed the path planning procedure (e.g., sequence of random steps in arbitrary direction).

In our previous research we had proposed a path planning spline-based navigation algorithm for a car-like mobile robot within a well-defined in advance environment [12]. It uses potential field method for obstacle collision avoidance and provides a locally sub-optimal path with regard to path length, path smoothness and safety optimization criteria. While typical path-planning algorithms prefer

final smoothing of a path at the last stages only [6,7], our algorithm special feature is the smoothness criterion integration into path optimization from the first stage of the algorithm. Other criteria (path length and safety) played secondary roles in the optimization and therefore a collision-free but not sufficiently smooth path was also treated as a low quality option. In order to improve the original algorithm performance, to add flexibility for path optimization and a possibility for a fast dynamic replanning (in a case the initially off-line selected path becomes unavailable), we combined it with Voronoi Diagram approach.

The rest of the paper is organized as follows. Section 2 briefly describes our previous research on spline-based path planning. It includes a potential function and optimization criteria definitions, path planning algorithm formulation and its particular weak points analysis. In Sect. 3 we present new criteria that improve the algorithm performance and add more flexibility for a user while selecting path evaluation criteria [13]. Section 4 presents our modification of the spline-based algorithm, which successfully overcomes the weaknesses of the initial approach. In addition, in this section we demonstrate how to apply the algorithm for searching paths within several homotopies. Section 5 demonstrates the original and the new algorithm examples, where the new algorithm shows successful solutions of the original algorithm failing cases. This section provides statistical comparison of the new and the original algorithms within several thousand tests in 18 different environments. Finally, we conclude and discuss our future work in Sect. 6.

2 Spline-Based Robot Navigation with Original Potential Field Approach

The spline-based method, proposed by Magid et al. [12] navigates a car-type robot in a planar known environment populated with static obstacles. The algorithm considers an omnidirectional circle-shape robot, which allows removing robot orientation and reducing a search space by one dimension. Therefore all further path planning is performed in a 2D configuration space with a point robot. Each obstacle is represented with a set of intersecting circles of different sizes. An obstacle may be constructed with just a single circle or a finite number of circles as we assume any arbitrary obstacle could be approximated with a finite set of circles up to a satisfactory level. Next, given a complete information about environment, a start and a target positions of the robot, the robot searches for a collision-free path applying a pre-determined cost function. The details of the algorithm could be found in [12], while this section briefly describes the selected cost functions, overviews the original algorithm, and demonstrates examples of its execution. Finally, we explain the reasons of algorithm drawbacks that lead to its failures in particular cases of environment in Subsect. 2.3. The algorithm modification, which successfully overcomes these difficulties is presented in Sect. 4.

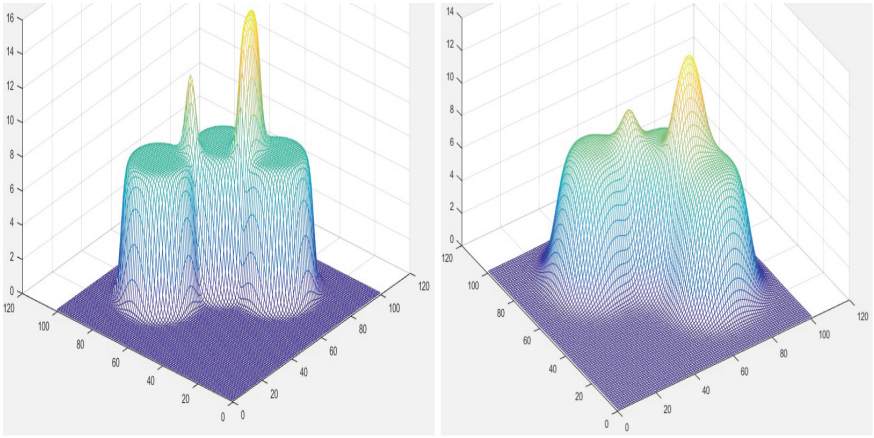

Fig. 1. Repulsive potential function of Eq. 1 for $\alpha = 0.5$ (left) and $\alpha = 0.2$ (right) that corresponds to the obstacles in Fig. 4 [13].

2.1 Original Algorithm Cost Function

To provide collision free path, a repulsive potential function is featured with a high value inside an obstacle and on its border and a small value within free space. This way, during local optimization procedure high value of the potential function in obstacle centre "pushes" all points of a path outside in order to minimize path cost. The potential field begins to drastically decline on obstacle boundary, keeps declining with distance as a point moves away from the border and becomes zero rather fast in a close vicinity of the obstacle. Let $q(t) = (x(t), y(t))$ be the robot position at time-stamp t. Then repulsive potential of a

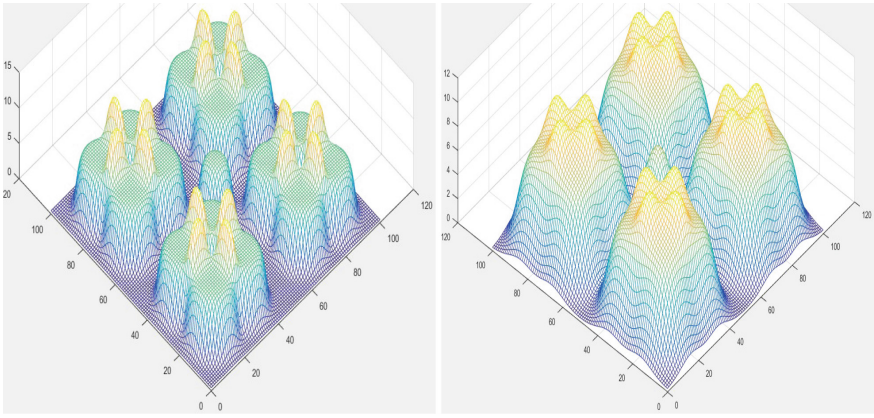

Fig. 2. Repulsive potential function of Eq. 1 for $\alpha = 0.5$ (left) and $\alpha = 0.2$ (right) that corresponds to the obstacles in Fig. 5 [13].

single circle (which is a part of an obstacle) contribution to the global potential function is defined with the following equation:

$$U_{rep}(q) = 1 + \tanh(\alpha(\rho - \sqrt{(x(t) - x)^2 + (y(t) - y)^2})) \tag{1}$$

where ρ is the radius of the obstacle with the centre at (x, y) and α is an empirically defined parameter, which is responsible for pushing a path outside of an obstacle. Figure 1 demonstrates the examples of two different selections of α parameter for the environment with a single obstacle that is formed by three intersecting circles, shown in Fig. 4. Here in the left sub-figure $\alpha = 0.5$ and in the right sub-figure $\alpha = 0.2$. Similarly, Fig. 2 demonstrates the examples of $\alpha = 0.5$ (in the left sub-figure) and $\alpha = 0.2$ (in the right sub-figure) for the environment with one circular obstacle in the centre and four symmetrical concave obstacles that are formed by four intersecting circles each. The correspond to this map environment is shown in Fig. 5. In the example with $\alpha = 0.5$, potential function forms distinct peaks at the circle intersections.

Topology function $T(q)$ takes into an account all N obstacles of the environment and their influence on the robot along the entire path, which is defined as a parametric function at the interval [0, 1]:

$$T(q) = \sum_{j=0}^{N-1} \int_{t=0}^{1} {U^j}_{rep}(q) \cdot \delta l(t) \cdot dt \tag{2}$$

where $\delta l(t)$ is simply a length of a segment:

$$\delta l(t) = \sqrt{(x'(t))^2 + (y'(t))^2} \tag{3}$$

Roughness function $R(q)$ is responsible for path smoothness property and is also integrated along the entire path:

$$R(q) = \sqrt{\int_{t=0}^{1} (x''(t))^2 + (y''(t))^2 dt} \tag{4}$$

Path length function $L(q)$ accounts for the path length, summing up the lengths of all path segments:

$$L(q) = \int_{t=0}^{1} \delta l(t) \cdot dt \tag{5}$$

Then, the final path cost function accumulates all the three above components:

$$F(q) = \gamma_1 T(q) + \gamma_2 R(q) + \gamma_3 L(q), \tag{6}$$

where $\gamma_{i=1..3}$ are the weight factors that set an influence of the corresponding component on a total cost of the path. In particular obstacle penalty influence component is empirically defined as $\gamma_1 = \frac{\beta}{2}$, where β ranges over a predefined array, which correlates with array of α parameters from Eq. 1 [12].

2.2 The Original Algorithm

The original algorithm of spline-based path planning works iteratively. Start point S, target point T, and the environment obstacles serve as input data. An initial path is suggested as a straight line between points S and T, and an equidistant point that lies on the straight line between S and T is calculated. These three points define a first spline-based path. With Eq. 6 a current path cost is calculated and next its is further optimized with Nelder-Mead Simplex Method [9] in order to minimize the path cost. A resulting *better* path serves as an initial guess for the next iteration.

The optimization procedure directly uses only a limited number of path points that define its spline, while path cost is calculated for all points of the path. The path-defining spline is rebuilt at each iteration using information from a previous stage, increasing the number of spline's points by one and adjusting parameters of the target cost function. Once the algorithm succeeds detecting obstacles-free path, a few more iterations are conducted in order to attempt improving the resulting path locally. The algorithm terminates in the two cases: (1) iteration count exceeds a user-defined limit or (2) iteration $i + 1$ does not improve previous iteration i, while increasing the spline complexity increases relatively to i-th spline. Figure 3 demonstrates an example of the original algorithm successful execution in a relatively simple environment, which contains a mix of convex and concave obstacles. Here the initial path is an optimized with regard to Eq. 6 spline with a single via point between S and T (Fig. 3a); after four iterations the spline has four via points but still collides with the obstacles (Fig. 3b); after seven iterations the 7-via-points-spline could be already applied for navigation (Fig. 3c), but two more iterations succeed to improve path's length and smoothness properties (Fig. 3d, 9th iteration); however, any further via points number increase does not improve the path. It took less than 10 min to obtain the final path after 9 iterations of the algorithm.

2.3 Drawbacks of the Original Approach

The original spline-based method succeeds to obtain a collision free smooth path for any complexity of the environment under the following set of conditions:

1. All obstacles of the environment are convex and do not intersect.
2. Each obstacle of the environment is approximated with a single circle.
3. There is some minimal distance between the distinct obstacles that provides a safety gap for a mobile robot passing.

In such environments the global potential function diminishes rapidly as the robot moves away from obstacle boundary. Moreover, the probability of getting stuck in a local minima within such environment is very small and could be neglected. However, if the obstacles are formed with several intersecting circles, each intersection introduce a potential field local maxima. Furthermore, if in such settings an initial spline passes through intersection of several obstacles, the cost function F(q) (Eq. 6) concentrates on pushing the spline out of intersection area

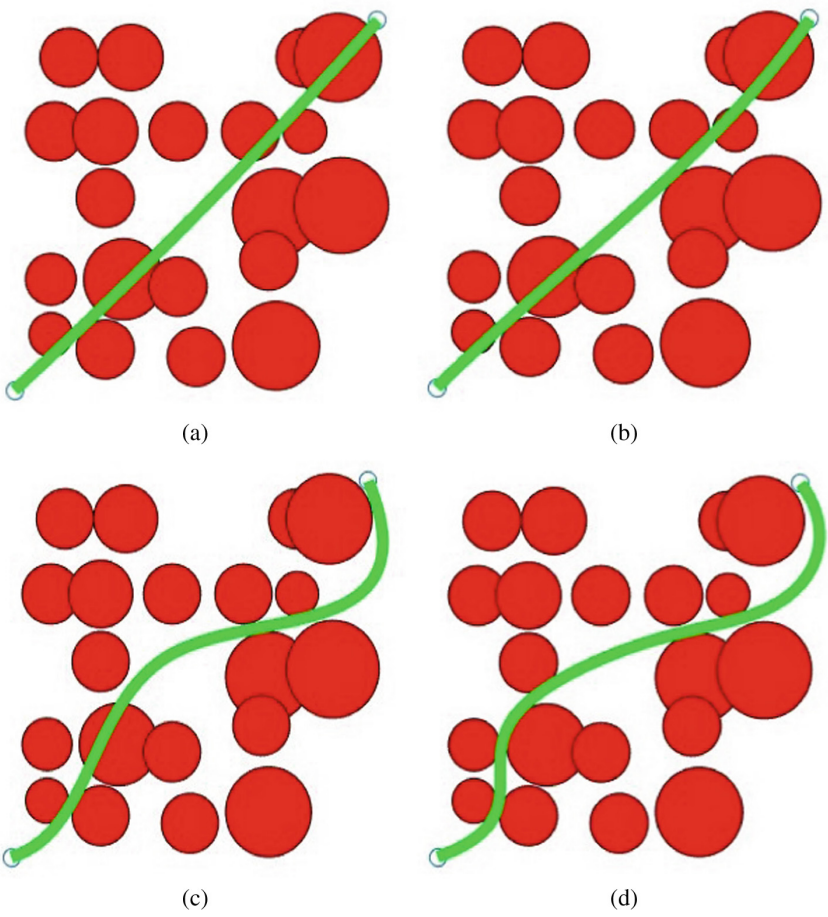

Fig. 3. Simple mixed environment: (a) the initial state, (b) 4 iterations, (c) 7 iterations, (d) the final path after 9 iterations [13].

that forms a local maximum. Upon (a guaranteed) successful escape from the intersection area, a all further iterations usually get stuck in a local minima as all next iteration splines, with no regard to their complexity, can not "jump over" some obstacle components because of the optimization process local nature.

A particular example of local minima issue is demonstrated in Fig. 4 where a relatively simple obstacle is formed by three intersecting circles. The corresponding potential field is presented in Fig. 1. Even though the original spline-based approach succeeds avoiding a local maximum and optimizing the path locally, it gets stuck at a local minimum. After 12 iterations the algorithm stops as no further improvement is possible, while the resulting path intersects a circle in the middle of the obstacle.

Figure 5 demonstrates an example with four simple concave obstacles and a convex obstacle in between. Circle intersections generate local repulsive potentials that in turn create local maxima and minima (Fig. 2). Similarly to the previous example, the local optimization successfully avoids local maxima, but even after 18 iterations the final path still intersects with obstacles and could not be applied for navigation.

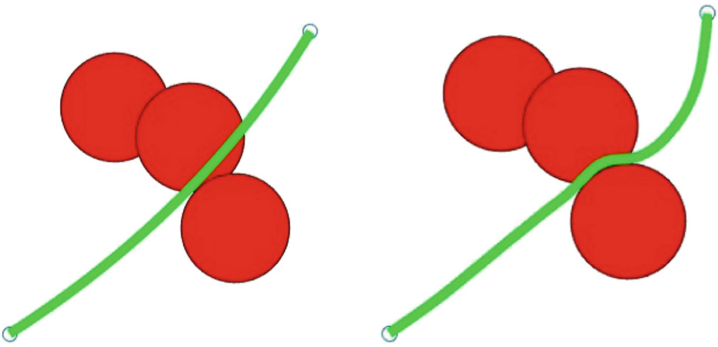

Fig. 4. Simple concave obstacle: the first iteration path (left) and the final path after 12 iterations (right) [13].

3 Additional Optimization Criteria

The original spline-based algorithm uses cost function (Eq. 6) uses three criteria: topology, roughness and path length. This section introduces two additional criteria and explores their influence on the resulting path. A particular point visibility time (or line-of-sight time) criterion refers to the path length where the robot literally keeps the point within its direct line of sight so that no obstacle occludes the point from the robot. The two additional criteria, which are integrated into the new cost function, are the start point visibility time and the target point visibility time.

These two new criteria are important if a robot requires to maximize the time of a direct visual or radio contact with a monitoring device or a router that may support path planning, localization or some other functionality. The criterion considers the ratio of visible and invisible (from the start or target point) path segments. Thus, the optimization goal will be to maximize the time when the robot is visible from the start point while following the selected path before it disappears for the first time behind some obstacle. In other words, we minimize the time (which is measured as a path length assuming a constant speed of the robot - in our future work we also extend this to the cases of varying speed along the path) after the robot becomes occluded for the first time, which we refer as invisibility of the start point S:

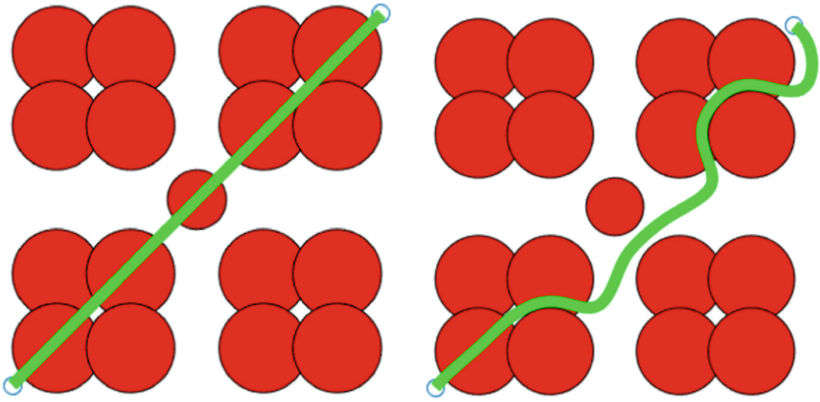

Fig. 5. Four simple concave obstacles with a convex obstacle in between: the first iteration path (left) and the final path after 18 iterations (right) [13].

$$I_S = 1 - \lim_{\delta t, \delta l(t) \to 0} \frac{\sum_{t=0}^{u} dist(A(t), A(t+\delta t))}{\int_{t=0}^{1} \delta l(t) \cdot dt} \tag{7}$$

such that

$$\forall t \in [0, u+\delta t] : [A(t), S] \cap (\cup_{j=1}^{N} Obs_j) = \emptyset \tag{8}$$

where the numerator in Eq. 7 reflects the path length of a visible from the start position S segment and the denominator reflects path length from Eq. 5. $A(t)$ denotes a position of the robot at timestamp t, and short segments of the path that were travelled between timestamp t and $t + \delta t$, which are denoted by $dist(A(t), A(t+\delta t)$, are accumulated. Equation 8 describes the visibility property, which means that a straight segment $[A(t), S]$ does not intersect any obstacle Obs_j where $j = 1..N$ and N is a number of obstacles in the environment. Thus, the last visible point from the start point while the robot follows the selected path before disappearing behind an obstacle for the first time is described with $A(u + \delta t)$. Similarly, we describe the criterion that shows the ratio of visible and invisible from the target point segments of a path, which we refer as invisibility of the target point T:

$$I_T = 1 - \lim_{\delta t, \delta l(t) \to 0} \frac{\sum_{t=w}^{1-\delta t} dist(A(t), A(t+\delta t))}{\int_{t=0}^{1} \delta l(t) \cdot dt} \tag{9}$$

such that

$$\forall t \in [w, 1] : [A(t), T] \cap (\cup_{j=1}^{N} Obs_j) = \emptyset \tag{10}$$

Equations 9 and 10 describe the first point $A(w)$ of a path segment $[A(w), A(1) = T]$ that marks the beginning of the last segment of the path, which is featured by a guaranteed constant visual contact between the robot and the target position T while the robot follows the selected path.

The new cost function combines the above five criteria as follows:

$$F(q) = \gamma_1 T(q) + \gamma_2 R(q) + \gamma_3 L(q) + \gamma_4 I_S + \gamma_5 I_T, \tag{11}$$

where γ_4 and γ_5 are the weight factors that set the line-of-sight criteria influence on a total cost of the path for start S and target T points respectively. Figure 6 demonstrates the two criteria influence on the path in the vicinity of start and target points: while with $\gamma_4 = 10$ and $\gamma_5 = 10$ these criteria do not contribute any significant influence (left sub-figure), with $\gamma_4 = 20$ and $\gamma_5 = 20$ the path changes are clearly visible, especially as the robot approaches the target position (right sub-figure); other parameters are defined as $\gamma_1 = 1$, $\gamma_2 = 1$, and $\gamma_3 = 0.5$ for both cases (both sub-figures).

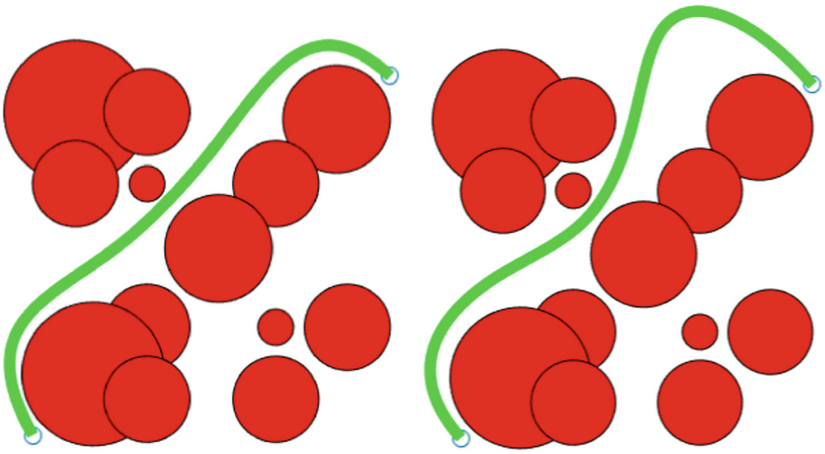

Fig. 6. Start and target points visibility time influence on the path quality: $\gamma_4 = 10$, $\gamma_5 = 10$ (left) and $\gamma_4 = 20$, $\gamma_5 = 20$ (right) [13].

As the path optimization with regard to Eq. 11 is performed only locally, the influence of the two additional parameters is also local. As a part of our future work we plan to apply the optimization at a global scale, which will allow the path to vary different homotopy sets in order to satisfy and emphasize a particular user-defined criterion influence.

4 Voronoi Graph Based Solution

Obviously, the local nature of the optimization procedure causes initial spline to be almost fully responsible for the original algorithm success or failure. Voronoi Graph approach [19] was selected for the integration with the original algorithm in order to provide a good initial spline, which could be iteratively improved locally with regard to user selection of the cost weights while avoiding potential function local minima troubles.

With the definition of obstacles and potential field from Sect. 2, prior to Voronoi graph construction, the two following preparation steps are performed:
1. Form Obstacles. A single obstacle may be formed with a single circle or with a set of intersecting circles. Register obstacles by grouping intersecting circles together. Initially, every circle is marked as idle and has its own index $i = 1..M$, where M is a (finite) number of circles in the environment. Start from an arbitrarily circle, assign it to obstacle O_1 and mark as an activated one. Next, iteratively grow O_1 by searching for all idle circles that intersect with O_1, assign them to obstacle O_1 and mark as activated. The iterative growth of O_1 continues until no more idle circles that intersect O_1 are detected in the environment. If the iterative growth of O_1 is completed but idle circles are still available, select another arbitrarily idle circle, assign it to obstacle O_2 and repeat the iterative growth procedure. Obstacles registration is completed when all circles of the environment become activated. For example, there are five obstacles that are formed by groups of circles in Fig. 7 and twelve obstacles in Fig. 8. While performing the registration, each pair of intersecting circles i and j provides their intersection point ω_{ij} in a case there is a single joint point (i.e., boundary touch) and a pair of points ω_{ij}, ω_{ji} in a case of joint two points (i.e., boundary intersection).
2. Form Boundaries. Find outer and inner boundaries of each obstacle of set $Obst = \{O_1, O_2, ..., O_k\}$, where k is a number of compound obstacle within the environment. Starting from an arbitrary circle within O_1 obstacle, boundaries of all circles that belong to O_1 are split into short segments of length σ, merged via intersection points ω_{ij} (or ω_{ji}) and labelled. Parameter σ is predefined in advance and correlates with a radius of a smallest circle of the environment in order to further match a shortest polygonal edge of contours. This way, two segments will receive the same label if there exists a continuous path between them, which is constructed from boundary segments. This procedure is repeated for each obstacle and upon its completion we receive a set of obstacles' boundaries. If the size of the latter set exceeds the size of *Obst*, it points out a presence of inner boundaries that were formed by internal contours. To get rid of such contours, for each particular obstacle O_i we encapsulated every contour of O_i into a convex hull, verify which one (of the resulting contours) forms the outer boundary of O_i and remove the rest. For example, this procedure has successfully removed four small diamond-shape internal contours inside complicated obstacles of environment in Fig. 7. Finally, we obtain several non-convex polygons, one for each obstacle within *Obst*, and tightly encapsulate them into a square map.

Next, we apply brushfire approach [3] to construct Voronoi graph as follows:

1. Parse free space with rays, which originate from edges of the obstacles and map boundaries. Figure 8 (the image on the right) demonstrates an example of Voronoi graph construction for the environment in the image on the left of the figure. Thick blue lines represent obstacle boundaries for *Obst* obstacles set. Thin blue lines are the rays that emerge outwards and inwards.

2. Calculate rays intersection points and connect them with segments for neighbouring ray. These segments are equidistant to boundaries of nearest obstacles (two or more obstacle boundaries including the map boundary), and all together they form Voronoi graph, which is depicted with a thin red line in Fig. 8 (the image on the right)

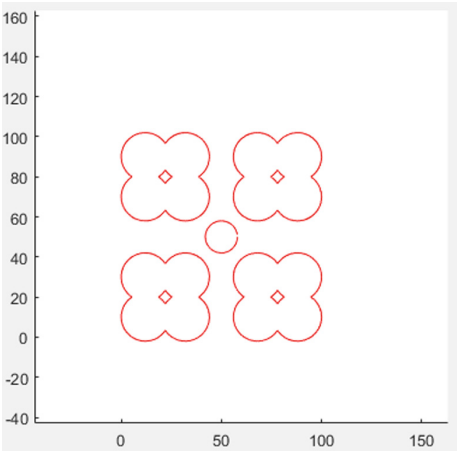

Fig. 7. Outer and inner boundaries of the obstacles [13].

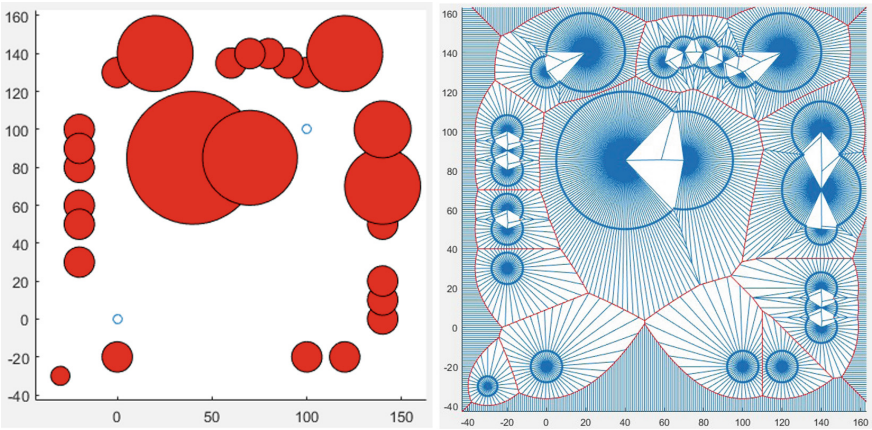

Fig. 8. Environment with obstacles (left) and Voronoi graph building procedure (right) [13].

After obtaining Voronoi graph VG, we select nearest to start position S and target position T points (S' and T' respectively) within VG so that segments $[S, S']$ and $[T, T']$ lie in the free space of the environment. Next, the shortest

path between S' and T' in VG is calculated with Dijkstra algorithm [5] (it is represented with thick red lines in Fig. 9). In general, any available path form S to T on Voronoi graph VG is guaranteed to be collision free and maximally safe with regard to distance from obstacle boundaries evaluation criteria. Therefore, any of these paths could serve as a good initial spline for the original spline-based method [12].

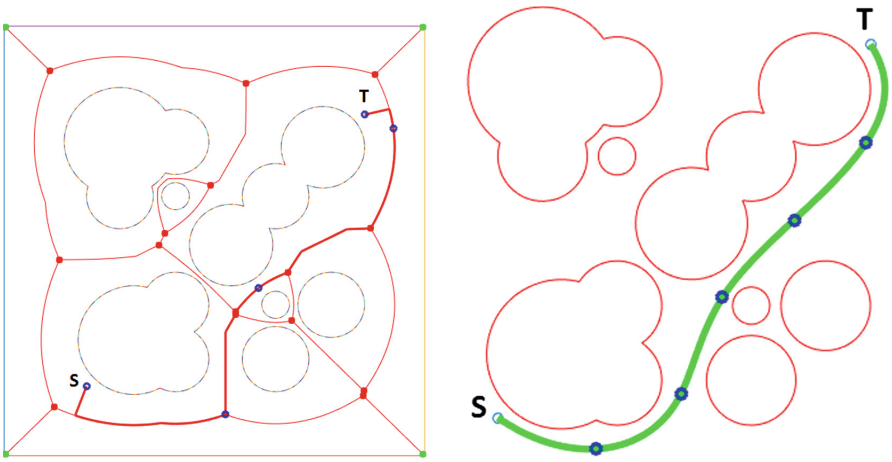

Fig. 9. The Voronoi graph VG of the environment (left): red thick dots are VG nodes and small blue circles are control via points of the initial spline proposal that approximates a selected path from start S to target T position; note that S and T are also control points and one of the VG nodes is selected by the algorithm as a via point. Final path from S to T is shown with a green curve (right) and small blue circles are control via points of the final spline, which is the selected locally optimal path after 2 iterations of the algorithm.

If we utilize S, T and those nodes Voronoi graph VG (thick red dots in Fig. 9), which form the selected path (S, T), in a role of via points for initial spline, such selection may fail to guarantee a good start of the spline-based method (the density of such points may be too low and they may become invisible to each other due to occlusions by obstacles). On the opposite, selecting all points that forms the selected path (S, T) will provide us with a dense and excess amount of via points, which in turn may cause an unfeasibly complicated spline. The proposed solution is to form a small set of special *control points* of the selected path, which could properly characterize features of the path while avoiding redundant complexity of its approximation spline.

We start with an empty list of control points $\{L\}$ and start point S is selected as a active point. Next, a farthest visible from S point of the path, VP_1, is calculated. S is added to $\{L\}$, while VP_1 receives a status of a next active point and again a farthest visible from VP_1 point of the path, VP_2, is calculated. VP_1 is included into $\{L\}$, VP_2 becomes a next active point and the process continues

until target point T becomes visible. Here, point VP_{i+1} is visible from point VP_i if they could be connected with a straight segment that does not collide with any obstacle of the environment. After the process finishes, the points of $\{L\}$ are utilized as via points for initial spline of the spline-based method [12]. Figure 9 demonstrates an example of control points list $\{L\}$, which are depicted as thick blue dots on Voronoi graph. It is important to emphasize that the amount of control points does not exceed the amount of VG nodes (thick red dots) of the selected within VG path (S, T). Thus, after these five control points form the via points of an initial spline, the algorithm required only two iterations (which added two new via points to the spline) in order to provide locally optimal path with regard to cost function (Eq. 11).

Using of Voronoi graph, we can find different paths from a start position to a target position that belong to different homotopy classes. The more distinct obstacles appear within the map, the larger is the amount of homotopy classes. For example, Fig. 11 demonstrates two different paths (in the middle and on the right) that belong to different homotopy classes. Figure 9 demon-

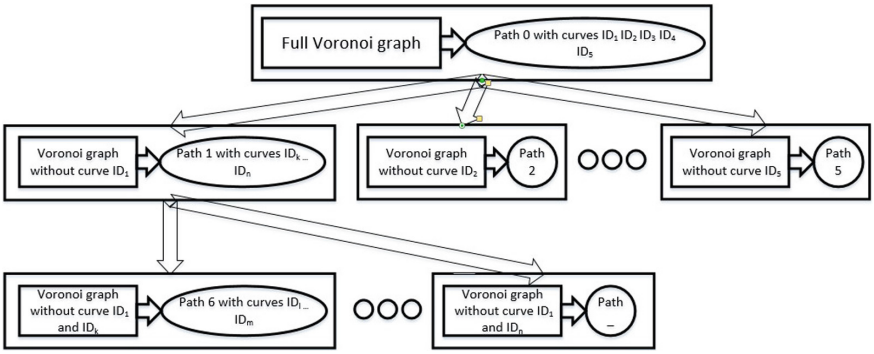

Fig. 10. Approach for path search in different homotopies.

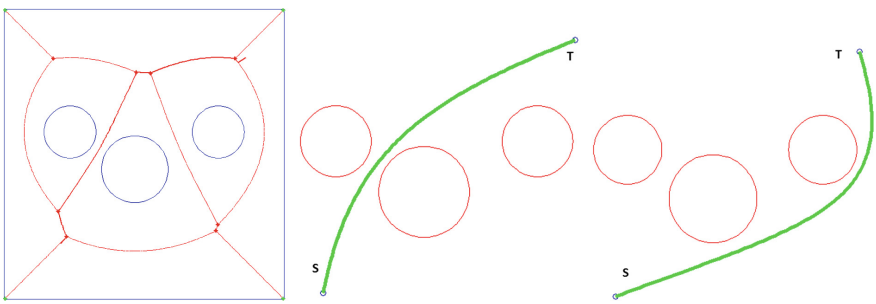

Fig. 11. Path planning results from start(S) to target(T): full Voronoi graph VG (left), a spline-based optimal path within homotopy class HC_1(middle), an optimal path within another homotopy class HC_2(right).

strates an environment with a large number of possible paths from S to T over more that ten homotopy classes. Having multiple options for initial spline selection within various homotopies is of special interest for urban search and rescue operations (USAR) that apply autonomous or semi-autonomous mobile ground and aerial vehicles. For example, USAR center may have an environment map before an anthropogenic catastrophe or terrorist attack and thus could create a Voronoi graph VG in the off-line mode for further use with a rescue vehicle. Number of ways to avoid obstacles and accordingly the potential paths could be overwhelming and calculation of all optional paths would take unfeasibly long time. However, off-line stage does not limit us with computation resources as such stage could be performed far in advance using high performance computational clusters. Calculation of multiple paths within different homotopies may be achieved using the following algorithm:

1. Calculate a number of Voronoi graph VG nodes. For example, in Fig. 9 there are 20 nodes in VG: 16 regular nodes that are marked with thick red dots (two pairs of nodes are located very close to each other but are still distinct nodes) and 4 corner nodes of the environment that are marked with thick green dots. Note, that S, S', T and T' are not considered as VG nodes because they are task dependent and would differ for different searches.
2. Calculate a number of of Voronoi graph VG edges and set a unique ID for each edge. For example, in Fig. 9 there are 26 edges in VG that are shown as red curves. Note that we do not add the edges, which form the environment boundary.
3. When a path from S to T is found, the IDs of all edges that form this path are stored. For example, in Fig. 9 path P_0, which is denoted in VG with thick red curves, contains five curves with ID_i, $i \in 1..5$. A complete Voronoi graph VG will serve as a root for the homotopic paths search tree and a sub-graph P_0 appears as a top node of the search tree, which we refer as level zero (Fig. 10).
4. The first level of the homotopic paths search tree is formed based on the size of P_0 and will contain the number of nodes that is equal to the number of edges in P_0. Each node of this level contains original Voronoi graph VG with an exclusion (deletion) of a single unique edge of the sub-graph P_0. Next, a new optimal path within its own homotopy class is found for each node sub-graph: P_{0i}, where $i \in 1..5$ and each new sub-graph corresponds to the deletion of a particular edge i. In the example (Fig. 10) we delete edge ID_1 from the first (from left to right) node sub-graph of VG, edge ID_2 from the second (from left to right) node sub-graph of VG etc. and then we obtain five new optional paths in five different homotopies.
5. The second level of the homotopic paths search tree is formed in a similar way as the first one, but now for each node of the second level two appropriate edges are excluded from VG. The process of tree building continues iteratively so that at each next level a sub-graph for path planning is reduced by one edge relatively to the original VG. Tree building process resembles a classical Breadth-first search (BFS) algorithm [4] and stops when a stop-condition, which is explained below, is met.

Before starting a construction of a homotopic paths search tree, a user should set a particular number of paths $\#HP$ - each within a distinct homotopy class - that he(she) is interested to obtain. $\#HP$ will serve as a stop-condition for BFS-based construction process: the process stops when the tree completed $\#HP$ distinct paths. To avoid homotopy class duplication, which directly lead to optimal path duplications, every new node of a new level is compared to already existing nodes within the same level and only a unique node is included into the construction.

5 Simulations and Algorithm Comparison

In order to verify our approach the new smart spline-based algorithm was implemented in Matlab environment and an exhaustive set of simulations was performed. Particular attention was paid to the cases where the original algorithm failed [12]. The cost function of Eq. 11 was applied with the following empirical parameter selection: $\gamma_1 = 1$, $\gamma_2 = 1$, $\gamma_3 = 0.5$, $\gamma_4 = 5$, and $\gamma_5 = 5$. The algorithm succeeded to provide collision-free paths in all cases, which was a natural consequence of applying initial Voronoi-based path as an initial spline input for the iterative algorithm.

Figure 12 demonstrates two environments, where the original spline-based algorithm had failed (e.g., the example in sub-figure (b) corresponds to Fig. 5). Voronoi graph always provides safe paths without obstacle collision. Therefore, a good initial path, being a sub-graph of VG, ensures that the modified spline-based algorithm calculates a final path in a significantly smaller number of iterations. Our preliminary concern about the time complexity of Voronoi graph construction turned out to be obsolete as the simulations empirically demonstrated that Voronoi graph calculations take acceptably small amount of time (at least for our reasonably simple cases, while more simulations in complicated large-size environments are scheduled as a part of the future work).

For example, for the environment of Fig. 12(a) the Voronoi graph construction and initial path calculation took only 2 s, while for Fig. 12(c) - 6 s. The total running time of the new algorithm decreased in three times in average with regard to the original algorithm. For example, the final path in Fig. 12(b) was calculated in 2 iterations within 2.5 min in Matlab, whereas the original algorithm [12] had spent 18 iterations and 38 min to conclude on its failure to provide an acceptable path between start to target points. Similarly, the original algorithm required 9 iterations and 15 min to provide a good path within Fig. 3 environment, whereas the new algorithm took 4 iterations and 5 min. In another case of Fig. 4 the original algorithm failed to find a path, while a Voronoi-based algorithm successfully completed the task within 3 iteration and 1 min (Fig. 11).

Initially our pilot simulations were performed manually. However, to obtained statistically reasonable comparison results, the algorithms should be verified with a large amount of data and in various environments for different selection of start and target points, which requires test automation. To automate the tests, we first created 18 different environments filled with obstacles, which were encapsulated

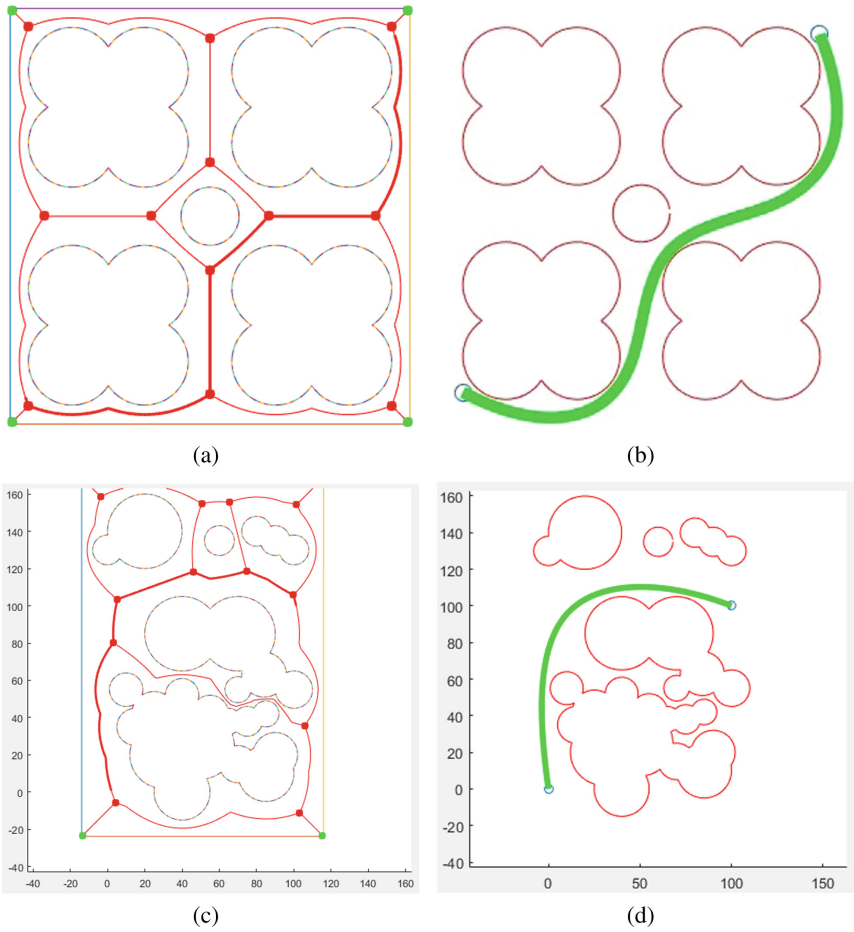

Fig. 12. Path planning results: full Voronoi graph VG (a, c) and corresponding spline-based optimal paths (b, d) [13].

into tight rectangular bounding boxes; to provide additional free space on the perimeter of a bounding box, each box was increased by 20% free space in all directions (up, down, right and left). A random number of points (from 10 to 500 points) were randomly distributed within each map - these points formed a list of candidates CPL for start and target points. From CPL we removed all points that were located inside obstacles or on obstacles boundary. Next, we formed a random number of pairs with CPL points and excluded the pairs, which could be connected with a straight segment that does not intersect any obstacle. The number of (S,T)-pairs was different for each of 18 environments and appear in the second column of Table 1. The idea of building a list originates from our previous publication on path planning [14].

Table 1. The performance of the original and the new algorithm in 18 different environments.

N	Pairs	T_{VC}	AIN^O	AIN^N	AT^O	AT^N	S^O	S^N
1	932	0.34	4.32	3.79	52.05	59.95	699	932
2	262	0.14	4.65	4.25	64.08	75.99	156	262
3	221	0.07	4.90	4.65	36.09	41.77	152	221
4	120	0.14	4.73	4.12	81.72	70.73	100	120
5	104	0.39	4.39	4.23	62.93	61.15	64	104
6	102	0.26	4.86	4.31	72.08	80.59	49	102
7	72	0.20	5.64	4.88	67.49	64.38	31	72
8	52	0.11	4.57	4.48	43.42	59.68	49	52
9	48	0.02	5.25	5.21	62.78	73.32	35	48
10	41	0.03	5.00	4.39	72.98	77.41	30	41
11	40	0.07	6.05	3.35	47.27	50.30	20	40
12	39	0.02	4.52	4.15	64.05	80.30	19	39
13	34	0.04	4.52	4.33	68.75	78.71	27	34
14	33	0.04	4.27	4.23	59.25	55.38	26	33
15	23	0.36	3.00	2.87	30.44	34.06	17	23
16	17	0.02	5.67	4.81	79.21	74.06	9	17
17	13	0.01	4.38	4.01	42.89	45.38	8	13
18	11	0.35	5.66	5.18	61.42	58.87	6	11

First, we evaluated the time, which is required by the new algorithm to perform the following actions in each environment:

1. Find intersection points of different circles, that form each obstacle.
2. Find circles unions for each compound obstacle.
3. Find and order boundary points of circles, which are not inside other circles.
4. Join the chains of boundary points for intersecting circles.
5. Delete inner boundaries that are formed by circle intersections inside each compound obstacle.
6. Construct Voronoi graph.

These actions are performed only once for each environment in order to construct a Voronoi graph. This time appears in the third column of Table 1 as T_{VC} - time for Voronoi graph construction.

Next, for each environment a number of automated tests were performed with the original and the new algorithms. The performance results are summarized in Table 1. The first column contains IDs of particular environments, and thus each line of the table corresponds to a particular environment. The second column contains a number of performed path searches within these environments (the number of (S,T) pairs). As was already mentioned, the third column T_{VC} shows

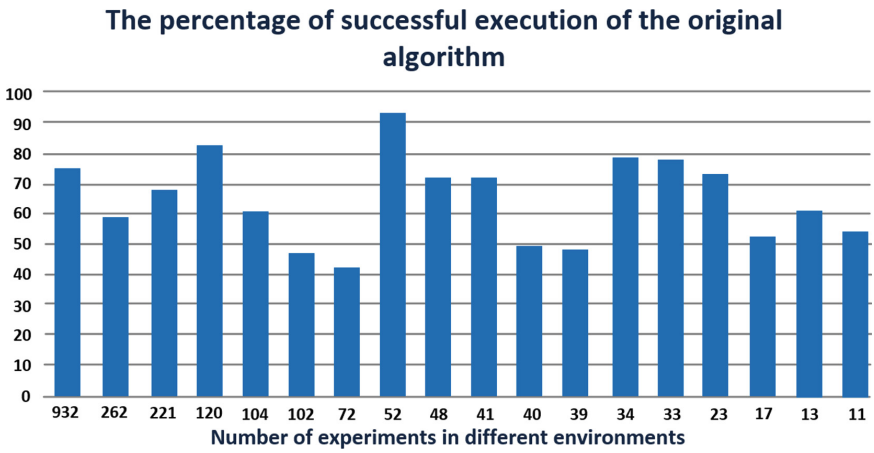

Fig. 13. The percentage of successful execution of the original algorithm.

the time that the new algorithm spent for a one-time Voronoi graph construction for each environment. The forth and fifth columns show average number of iterations to obtain an obstacle-free locally optimal path in the corresponding environment with the original algorithm (AIN^O) and the new algorithm (AIN^N) respectively. The sixth and seventh columns show average time of an obstacle-free locally optimal path calculation (in seconds) in the corresponding environment with the original algorithm (AT^O) and the new algorithm (AT^N) respectively. Finally, two last columns summarize the success rate of the original (S^O) and new algorithms (S^N) by listing a number of successfully found obstacle-free locally optimal paths. Before analysing the performance, an important remark should be made. For the original algorithm Table 1 contains only the results of successful searches, while all failure reports, that usually require at least 15 min of execution time in order to conclude on its failure, were not included into average time and iteration numbers statistics.

Analysis of Table 1 in the term of algorithm success rates (which is successful path search or failure report) is demonstrated in the graph of Fig. 13. Y-axis shows the success rate, while X-axis is responsible for a particular environment and the number below each bar reflects a number of total path searches within the environment.

The graph concentrates only on the success rates of the original algorithm execution, because the new algorithm succeeded in all executions (compare the second and the last columns of Table 1). Unfortunately, while in some environments success rate is 94%, in the worst scenario it succeeded only in 43% of the executions. Figure 14 compares the algorithms in terms of average iteration numbers, which are required for the algorithm to successfully calculate a locally optimal path. Here the new algorithm outperformed the original algorithm in all environments. Worth noticing that a reduced number of iteration for the new algorithm is caused by initial spline complexity: while the original algorithm

starts from a straight line, the new algorithm starts from an obstacle-free path and only needs to iteratively optimize it locally. Again, the number of average iterations for the original algorithm does not account for failure report cases that would significantly increase the number of its iterations and time. Moreover, as the two algorithms use slightly different cost evaluation functions - Eq. 6 for the original algorithm and Eq. 11 for the new algorithm - calculation time within optimization procedure of the new algorithm increases.

Note, that for the new algorithm the time spent for Voronoi graph construction differs from average path search time in several magnitudes, which means that it is neglectable at least for simple environments, while the graph could be reused for multiple searches within the same environment and in some cases even calculated off-line, as was proposed in the previous section.

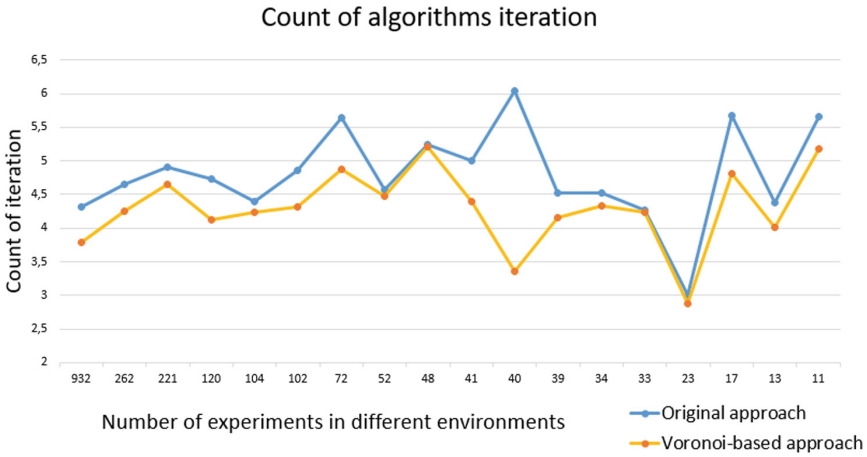

Fig. 14. Comparison of the average number of iterations of algorithms.

6 Conclusions

In this paper we have presented a combined method for calculating a smooth and safe path for a mobile car-type robot in static planar environments. The introduced Voronoi-based modification of our original spline-based path planning algorithm helped avoiding local minima problem and added more flexibility for path optimization. The cost function was modified by introducing additional criteria that maximized the time of keeping a robot within direct line of sight from start and target points while the robot follows a path. The new smart spline-based method algorithm was implemented in Matlab environment and its results were explicitly compared with our original algorithm. The new algorithm demonstrated significantly better results than the original approach. The comparison was based on success rate, number of iterations and running time of

the algorithms. In total, several thousands tests were performed in 18 different simulated environments.

Experimental comparison demonstrated that the new algorithm requires in average 10% fewer iterations than the original algorithm and succeeds finding all existing paths in all environments, while failure rate of the original algorithm was about 34% in average and up to 67% in the worst-case environment. Average search time of the new algorithm increased by 6% relatively to the original algorithm and is caused by our statistical evaluation approach: for the original algorithm we do not accumulate unsuccessful attempts that report on failure to find a path, and such attempts typically take significantly long time than successful attempts (in order of two magnitudes).

As a part of our future work we plan to introduce several new additional parameters into the cost function, including varying speed of a robot along a path. A large number of simulated tests will be carried in order to analyse path search time on several homotopies and to investigate the influence of the most suitable homotopy selection criteria on a final path. We consider extending the new algorithm for 3D environment and adding new parameters of cost function that are associated with 3D surfaces. The algorithm will be organized as a ROS package with C++ implementation and further verified with real navigation of a heterogeneous robotic team operating in an urban search and rescue scenario.

Acknowledgement. This work was partially supported by the Russian Foundation for Basic Research (RFBR) and Ministry of Science Technology & Space State of Israel (joint project ID 15-57-06010). Part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

1. Andrews, J.R., Hogan, N.: Impedance control as a framework for implementing obstacle avoidance in a manipulator. Master's thesis, Department of Mechanical Engineering, M.I.T. (1983)
2. Buyval, A., Afanasyev, I., Magid, E.: Comparative analysis of ROS-based monocular SLAM methods for indoor navigation. In: Proceedings of 9th International Conference on Machine Vision (2016)
3. Choset, H.M.: Principles of Robot Motion: Theory, Algorithms, and Implementation. MIT Press, Cambridge (2005)
4. Cormen, T.H.: Introduction to Algorithms. MIT Press, Cambridge (2009)
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
6. Elbanhawi, M., Simic, M., Jazar, R.N.: Continuous path smoothing for car-like robots using b-spline curves. J. Intell. Robot. Syst. **80**(1), 23–56 (2015)
7. Fleury, S., Soueres, P., Laumond, J.-P., Chatila, R.: Primitives for smoothing mobile robot trajectories. IEEE Trans. Robot. Autom. **11**(3), 441–448 (1995)
8. Khatib, O., Siciliano, B.: Springer Handbook of Robotics. Springer, Berlin (2016)
9. Lagarias, J.C., Reeds, J.A., Wright, M.H., Wright, P.E.: Convergence properties of the nelder-mead simplex method in low dimensions. SIAM J. Optim. **9**(1), 112–147 (1998)

10. Latombe, J.-C.: Robot Motion Planning, vol. 124. Springer Science & Business Media, Berlin (2012)
11. Tang, L., Dian, S., Gu, G., Zhou, K., Wang, S., Feng, X.: A novel potential field method for obstacle avoidance and path planning of mobile robot. In: Proceedings of 3rd IEEE International Conference on Computer Science and Information Technology, vol. 9, pp. 633–637. IEEE (2010)
12. Magid, E., Keren, D., Rivlin, E., Yavneh, I.: Spline-based robot navigation. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2296–2301. IEEE (2006)
13. Magid, E., Lavrenov, R., Khasianov, A.: Modified spline-based path planning for autonomous ground vehicle. In: Proceedings of 14th International Conference on Informatics in Control, Automation and Robotics (2017)
14. Magid, E., Rivlin, E.: CAUTIOUSBUG: a competitive algorithm for sensory-based robot navigation. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2757–2762 (2004)
15. Magid, E., Tsubouchi, T., Koyanagi, E., Yoshida, T.: Building a search tree for a pilot system of a rescue search robot in a discretized random step environment. J. Robot. Mech. **23**(4), 567 (2011)
16. Pipe, A., Dailami, F., Melhuish, C.: Crucial challenges and groundbreaking opportunities for advanced HRI. In: Proceedings of IEEE/SICE International Symposium on System Integration, pp. 12–15. IEEE (2014)
17. Rosenfeld, A., Agmon, N., Maksimov, O., Azaria, A., Kraus, S.: Intelligent agent supporting human-multi-robot team collaboration. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 1902–1908. AAAI Press (2015)
18. Seraji, H.: Traversability index: a new concept for planetary rovers. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 3, pp. 2006–2013. IEEE (1999)
19. Toth, C.D., O'Rourke, J., Goodman, J.E.: Handbook of Discrete and Computational Geometry. CRC Press, Boca Raton (2004)
20. Yakovlev, K., Khithov, V., Loginov, M., Petrov, A.: Distributed control and navigation system for quadrotor UAVs in GPS-denied environments. In: Proceedings of the 7th IEEE International Conference on Intelligent Systems, pp. 49–56. Springer International Publishing, Berlin (2015)