Robotics and Autonomous Systems 61 (2013) 1440–1449

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Path planning with obstacle avoidance based on visibility binary tree algorithm

Abdulmuttalib Turky Rashid [a,*], Abduladhem Abdulkareem Ali [b], Mattia Frasca [c], Luigi Fortuna [c]

[a] Electrical Engineering Department, University of Basrah, Basrah, Iraq
[b] Computer Engineering Department, University of Basrah, Basrah, Iraq
[c] DIEEI, Faculty of Engineering, University of Catania, Catania, Italy

HIGHLIGHTS

- A new algorithm for robot navigation, referred to as visibility binary tree algorithm is introduced.
- The construction of this algorithm is based on the visible tangents between robot and obstacles.
- The shortest path is run on top of the visibility binary tree.
- The performance is compared with three different algorithms for path planning.

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form
5 July 2013
Accepted 15 July 2013
Available online 23 July 2013

Keywords:
Path planning
Obstacle avoidance
Visibility graph
Bresenham algorithm

ABSTRACT

In this paper, a novel method for robot navigation in dynamic environments, referred to as *visibility binary tree algorithm*, is introduced. To plan the path of the robot, the algorithm relies on the construction of the set of all complete paths between robot and target taking into account inner and outer visible tangents between robot and circular obstacles. The paths are then used to create a visibility binary tree on top of which an algorithm for shortest path is run. The proposed algorithm is implemented on two simulation scenarios, one of them involving global knowledge of the environment, and the other based on local knowledge of the environment. The performance are compared with three different algorithms for path planning.

1. Introduction

Path planning and obstacle avoidance are two important aspects of autonomous mobile robot navigation. Based on the sensor information available, the approaches to path planning can be classified into global and local methods [1,2]. In global methods, the robot plans its trajectory on the basis of a global information on the environment [3]. This approach guarantees the convergence of the robot path to the target, and also indicates if the goal is reachable or unreachable. On the other side, planning approaches based on sensors providing limited (local) information, although of simpler implementation, do not guarantee the global convergence to the target [4], since the robot uses its sensors to locate nearby obstacles at each control cycle and to plan the next action to be executed.

Many techniques for path planning in the presence of obstacles employ a Voronoi diagram under the hypothesis either of a global [5] or a local knowledge scenario [6]. In the global knowledge scenario the assumption is that each robot has complete information about all obstacles in the environment, while in the local knowledge scenario the information of each robot is limited to its sensing range. In approaches based on the Voronoi diagram, the planned trajectory is either a piecewise linear trajectory or a smooth path. In the first case, the robots have to stop at each of the trajectory segments end, change its orientation according to the next segment and then restart again. This kind of motion is disagreeable and leads to additional waste of power. In order to get a smooth path, the use of different curves instead of linear segments has been proposed. An example is the use of Voronoi diagram for smooth path planning and better obstacle avoidance by iterative enhancement proposed in [7]. Another example is the use of Bezier curves. Smooths paths that are reliable with robot dynamics are generated by using Bezier curves of degree three [8]. Time optimality [9], navigation in presence of corridor constraints [10] and

* Corresponding author. Tel.: +964 7806325288.
E-mail addresses: muttalib_63@yahoo.com, abdturky@gmail.com (A.T. Rashid), Abduladem1@yahoo.com (A.A. Ali), mfrasca@diees.unict.it (M. Frasca).

0921-8890/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.robot.2013.07.010

curvature control [11,12] are other issues of navigation control addressed with approaches based on the use of Bezier curves. These approaches produce smooth paths but often require longer trajectories to the target.

Another class of methods for path planning is based on the use of graph search algorithms for shortest path. In such approaches, first the graph of possible paths is built and then a shortest path search algorithm is applied on such graph. The visibility graph can be used for this purpose: this is a graph of visible obstacle vertices (obstacles are assumed to be polygonal), where a vertex V is defined as visible from a vertex U if the segment VU does not intersect any obstacle edges in the environment [13,14]. A general limitation of methods based on graph is the computation time for shortest path calculation. For this reason, a simplification of the graph structure to be explored may result in an improvement of the efficiency of these methods. This is the idea underlying the introduction of other graphs instead of the visibility graph. Several works use, for instance, the so-called tangent visibility graph. The tangent visibility graph is defined as the set of possible trajectories obtained from visibility graph by retaining only the edges which are bi-tangent to convex obstacle vertices [15,16]. The tangent visibility graph has a lower number of vertices and edges than the visibility graph. This leads to a more efficient process of shortest path calculation for the graph. As concerns the algorithms for finding the shortest-path the Dijkstra's algorithm is used on a full visibility graph in [17]. The search can be optimized by running it on the tangent graph as in [18,19], instead on the visibility graph.

The aim of this paper is to introduce a new algorithm for path planning based on a further simplification of the graph structure used. The resulting graph, which we name *the visibility binary tree*, is derived from the tangent visibility graph. We assume that the robot and the obstacles have circular shapes, that the robot radius is R and that the obstacle radius is the sum of the physical space occupied by the obstacle and the radius of the robot. The visibility binary tree is built starting from all possible paths between the robot position and the target and optimizing the structure by reducing redundant edges. After this step, an ad hoc searching algorithm is run on this graph. In fact, thanks to the simplified structure, the searching phase is also optimized. A further contribution of this work is the use of a Bresenham algorithm for low level trajectory planning. This has the advantage of requiring few computational resources, so that the whole algorithm introduced in this paper is aimed at reducing the computational resources required for its implementation.

The rest of the paper is organized as follows. Section 2 describes the low level of trajectory planning of the robot. Section 3 develops the theoretical analysis, describing the visibility binary tree algorithm. In Section 4, the visibility binary tree algorithm has been tested in two scenarios. Finally, Section 5 draws the conclusions of the paper.

2. Low level of trajectory planning

In this section, we briefly discuss the low level trajectory planning of the robot. The low level trajectory planning aims at implementing the routines needed for a robot to follow a given trajectory, whereas this trajectory is the result of the high level path planning. In particular, in this section we briefly discuss the kinematics of the robot and the Bresenham algorithm used to implement robot motion along a straight line or an arc line.

Although Bresenham algorithms [20] were developed for drawing lines and circles on a pixelated display systems such as VGAs [21], in our work we apply them to implement the low level of trajectory planning. In pixelated display systems a line is defined as a set of points (pixels in the screen), in our approach we represent the trajectory of robot to be computed as a set of successive positions in the plane (points in the plane). So, we establish an

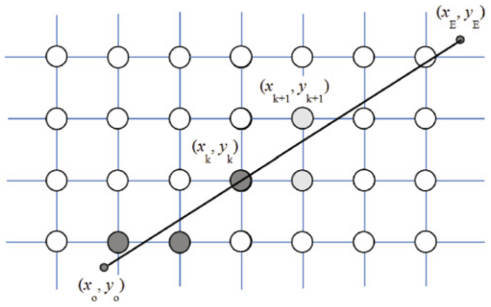

Fig. 1. Illustration of the Bresenham line algorithm.

analogy between pixels and points in the plane. To the best of our knowledge, this is the first time in which these algorithms are applied to trajectory planning. There are several useful characteristics in these algorithms motivating this choice: they are fast incremental algorithms and use only integer calculations; all multiplications are by 2 and thus can accomplished with a simple left shift instruction. These characteristics make these algorithms particularly suitable for implementation in hardware systems with limited available resources.

Let us first discuss the Bresenham line algorithm, which we use to implement planning of a straight line trajectory. Consider as in Fig. 1 a two-dimensional grid of points in the space where the robot moves. Assume that the robot initial position is (x_0, y_0), the direction to follow is given by the straight line shown in Fig. 1 and that the final end point is (x_E, y_E). The objective of the algorithm is to derive the sequence of positions in the grid in which the robot has to move to follow in an approximate way the depicted line. This is accomplished by moving at each step to the next position along the x axis (i.e., from x_k to x_{k+1}) and then by selecting which of y_k or y_{k+1} is the closest coordinate to the line (the points in the grid are indicated as (x_k, y_k) where k is an index labeling the points in the grid). Thus, the algorithm essentially has to choose at each step the value of the y coordinate. This is done by calculating at each time step a decision parameter p_k.

The algorithm can be described by the following steps:

1. start from the two line end points (x_0, y_0) and (x_E, y_E) and calculate the constants $\Delta x = x_E - x_0$ and $\Delta y = y_E - y_0$.
2. calculate the first value of the decision parameter as:
$$p_0 = 2\Delta y - \Delta x. \tag{1}$$
3. for each value of x_k along the line, perform the following test. If $p_k < 0$, the next point to be selected is (x_{k+1}, y_k) and:
$$p_{k+1} = p_k + 2\Delta y. \tag{2}$$
Otherwise, the next point to be selected is (x_{k+1}, y_{k+1}) and:
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x. \tag{3}$$
4. repeat step 4 until (x_E, y_E) is reached.

Let us now discuss the Bresenham circle algorithm we used to derive a circular trajectory of the robot. The algorithm assumes that the circle is centered at the origin, so that the circle has a eight-way symmetry and it suffices to calculate the locations of the robot in one of the octants. We will refer to the original coordinates as (X, Y) and to the coordinates in the reference frame where the circle is centered in the origin as (x, y). Analogously to the case of the Bresenham line algorithm, at each step the new position x_{k+1} is selected and the algorithm has to choice which coordinate y_k or y_{k-1} is closest to the circle boundary. This is done by testing if the mid point between y_k and y_{k-1} is inside the circle or not, i.e., if $f(x, y) \triangleq x^2 + y^2 - r^2$ calculated at the mid point is negative or positive. This idea can be accomplished in an incremental way by the following steps (see Fig. 2):

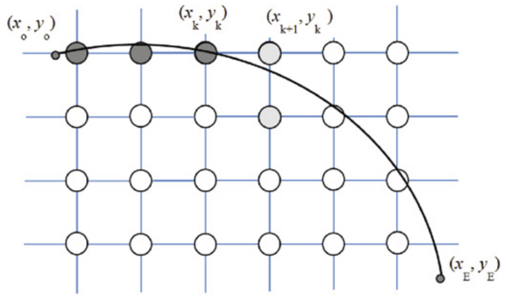

Fig. 2. Illustration of the Bresenham circle algorithm.

1. input the radius r and the circle center (x_c, y_c), and consider a left top point in the 2D grid with coordinates:
$$(x_0, y_0) = (0, r). \tag{4}$$
Assume that the robot starts from here, i.e., $(0, r)$.
2. calculate the initial value of the decision parameter as:
$$p_0 = \frac{5}{4}r. \tag{5}$$
3. for each point x_k, perform the following test.
if $p_k < 0$, the next point along the circle is $(x_k + 1, y_k)$ and:
$$p_{k+1} = p_k + 2x_{k+1} + 1. \tag{6}$$
Otherwise the next point along the circle is $(x_k + 1, y_k - 1)$ and:
$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}. \tag{7}$$
4. determine symmetrical points in the other seven octants, if needed.
5. calculate the coordinates in the original reference frame.
6. repeat steps 3–5 until $x_k \geq y_k$.

3. The visibility binary tree algorithm

In this section, the visibility binary tree algorithm for path planning of mobile robot with obstacle avoidance is introduced. We will first give an overview of the algorithm and then enter in the details of the substeps that constitute it. Although the algorithm is totally general, in the following we will assume that the robot kinematics is a differential-drive. In fact, our discussion about the way in which orientation of the robot towards the target is achieved explicitly makes this hypothesis for the robot kinematics.

The algorithm starts from the construction of the complete set of paths from source to target in the tangent visibility graph (Fig. 3(a) and (b)), that is, all the possible paths that are in the direction of the target and respect the visibility condition are considered (we recall that a vertex V is defined as visible from a vertex U if the segment VU does not intersect any obstacle edges in the environment). Then, this set is represented through a binary tree, called visibility binary tree (Fig. 3(c)). The next step is the optimization of this binary tree (Fig. 3(d)) by removing edges that do not belong to the shortest path between source and target (essentially by replacing sequences of two edges with a concave angle in any set of three vertices by a direct edge from the first to the third vertex). Finally, a searching algorithm is applied to this optimized binary tree to find the shortest path between source and target (Fig. 3(e)). This shortest path is the trajectory that the robot will follow.

The substeps mentioned above will be now described in details in the next subsections.

3.1. Construction of the complete set of paths

The first step in the visibility binary tree algorithm is the construction of the complete set of paths from source (i.e., from the robot starting position) to target by using the tangent visibility graph. The idea of this algorithm is to search for safe (i.e., without collisions) and as much as possible direct paths.

The different paths are built in an iterative manner. At each step a direct path between the robot position and target is first assumed (we refer in the following to it as 'standard' robot trajectory: in fact, this is composed by two tracts: an arc path to readjust orientation of the differential-drive robot towards the target and a straight line from the end of the arc path to the target point). Then, if a collision is detected along this path, this path is replaced by two possible paths. These paths are derived by drawing the two tangent lines from robot position to the safe circle of the colliding obstacle. This process is repeated until the last vertex of the path is the target point.

In particular, the implementation of this algorithm is accomplished according to the following steps.

Step 1: sensing the position, radius, orientation, and velocity of robot $(P_s(x_s, y_s), r_s, \theta_s, v_s)$; the position and radiuses of obstacles $(P_1, P_2, .., P_n, r_1, r_2, r_n)$; the target location $(P_g(x_g, y_g))$.

Step 2: computation of a simple 'standard' robot trajectory from source to target. This trajectory consists of two parts: an arc path, needed to orient the differential drive robot towards the target, and a straight line representing the tangent line between the circular trajectory of the robot and the target point, as shown in Fig. 4.

Step 3: check if there is any obstacle intersecting the path between robot and target. The intersection occurs when the distance between the center of the obstacle and the straight line from source to target is less than the radius of obstacle.

Step 4: if any intersection is detected, then the inner and outer tangent lines from the arc trajectory of the robot to the obstacle are drawn. Let us first focus on outer tangent lines. Four points have to be calculated: referring to Fig. 5 these points are indicated as (x_3, y_3), (x_4, y_4), (x_5, y_5) and (x_6, y_6), whereas (x_s, y_s), (x_g, y_g), indicate the source and target point, respectively. The derivations of the expression to calculate such point is omitted since it is based on quite simple geometrical constructions, we only mention that this involves the construction of third circle (indicated in Fig. 5 with dashed lines) having radius equal to the difference of the radii of the two circles and centered in (x_s, y_s). (x_1, y_1) and (x_2, y_2) represent the intersection points between tangent lines from the target to this third circle. Given that $D_1 = \sqrt{(x_g - x_1)^2 + (y_g - y_1)^2}$ and $D_2 = \sqrt{(x_g - x_2)^2 + (y_g - y_2)^2}$ (where D_1 and D_2 represent the distances between (x_g, y_g) and (x_1, y_1) and between (x_g, y_g) and (x_2, y_2)), the coordinates of the four points are calculated as follows:

$$\begin{aligned}
x_3 &= x_1 - r_g \cdot \frac{y_g - y_1}{D_1} \\
y_3 &= y_1 + r_g \cdot \frac{x_g - x_1}{D_1} \\
x_4 &= x_g - r_g \cdot \frac{y_g - y_1}{D_1} \\
y_4 &= y_g + r_g \cdot \frac{x_g - x_1}{D_1} \\
x_5 &= x_2 + r_g \cdot \frac{y_g - y_2}{D_2} \\
y_5 &= y_2 - r_g \cdot \frac{x_g - x_2}{D_2} \\
x_6 &= x_g + r_g \cdot \frac{y_g - y_2}{D_2} \\
y_6 &= y_g - r_g \cdot \frac{x_g - x_2}{D_2}.
\end{aligned} \tag{8}$$

In the case of inner tangent lines (Fig. 6), the following equations have to be used to calculate the intersection points (x_3, y_3), (x_4, y_4), (x_5, y_5) and (x_6, y_6) (in this case the dashed circle

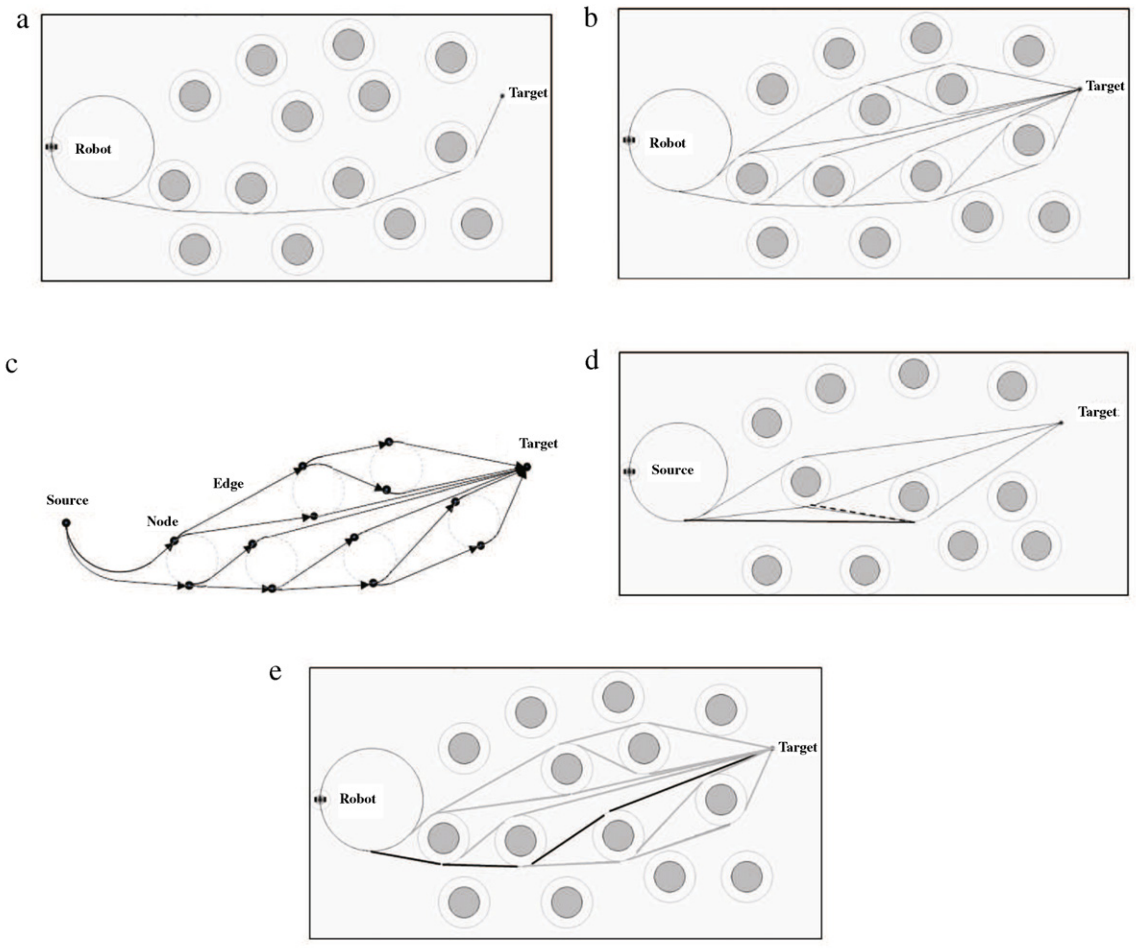

Fig. 3. (a) First path from source to target. (b) Set of complete paths. (c) Representation of the set of complete paths by the visibility binary tree. (d) Optimized binary tree. (e) The shortest path (in bold lines) in the optimized binary tree.

has radius equal to the sum of the radii of the other two circles):

$$\begin{aligned}
x_3 &= x_1 - r_g \cdot \frac{y_1 - y_g}{D_1}\\
y_3 &= y_1 - r_g \cdot \frac{x_g - x_1}{D_1}\\
x_4 &= x_g - r_g \cdot \frac{y_1 - y_g}{D_1}\\
y_4 &= y_g - r_g \cdot \frac{x_g - x_1}{D_1}\\
x_5 &= x_2 + r_g \cdot \frac{y_2 - y_g}{D_2}\\
y_5 &= y_2 + r_g \cdot \frac{x_g - x_2}{D_2}\\
x_6 &= x_g + r_g \cdot \frac{y_2 - y_g}{D_2}\\
y_6 &= y_g + r_g \cdot \frac{x_g - x_2}{D_2}.
\end{aligned} \tag{9}$$

Step 5: repeat steps 2–4 by assuming as starting point each of the intersection points of the inner and outer tangent lines calculated before. The process is repeated until the target is reached for each intersection point of each tangent line, as shown in Fig. 3(b).

3.2. Representation of the set of complete paths by a binary tree

The complete set of paths obtained is then represented as a binary tree. The root of the tree represents the source, i.e., the robot starting point. Each node then represents an intersection point in the robot trajectory. Each node thus has one incoming edge (from the last point visited by the robot) and two outcoming edges, towards the next intersection points to be reached by the robot (one related to the inner tangent line and the other to the outer tangent line). Finally, the only leaf of the graph represents the target.

3.3. Optimization of the binary tree

The binary tree is then optimized. The aim of this step is to reduce the number of links in the graph. In fact, the graph may contain sequences of edges with a concave angle between them. Since such sequences clearly do not represent the shortest path for the robot, the number of links in the graph can be reduced by replacing such sequences with one link. Consider, for instance, the dashed edge shown in Fig. 3(d). It forms a concave angle with another edge of the graph and, therefore, these two edges can be replaced by the solid bold line which is a shorter trajectory for the robot. The binary tree is optimized by looking at such sequences of edges and substituting them with a new edge when this latter has two visible end nodes.

3.4. The search algorithm

Given the simplified structure of the graph in which the shortest part has to be searched for, this latter step can be performed in a simple and computationally efficient way based on the following steps.

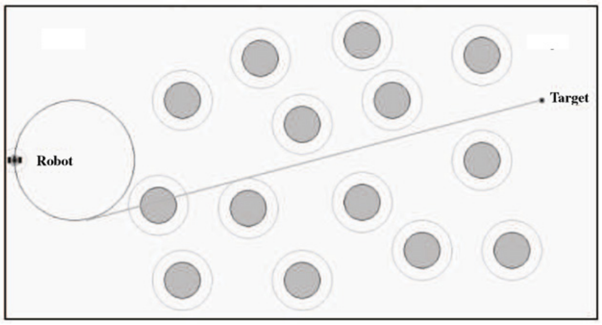

Fig. 4. A standard trajectory of the mobile robot.

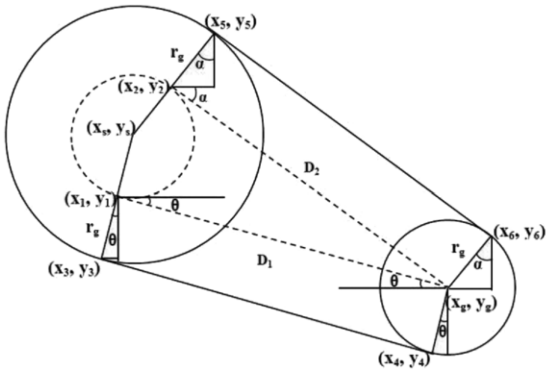

Fig. 5. Calculation of the outer tangent lines.

Step 1: mark the distance to every node on the graph as infinity.

Step 2: for the first iteration, choose the starting point as the current node and fix the distance to it to zero. Select one path and update the distances to every unvisited neighbor node, by determining the sum of the distance between an unvisited node and the current node.

Step 3: mark the current node as visited (in Fig. 7 the edge is colored as a gray solid line), and select the next unvisited node along the same path until the destination node is reached.

Step 4: repeat step 3 for the other paths. In Fig. 7 the subtrees arising from the outer tangent to the first obstacle are drawn as dotted lines, and the subtrees from the inner tangent to the first obstacle are drawn as dashed lines.

Step 5: compare the current path with the best path obtained so far and choose the shortest. The final result of this process is the shortest path shown in Fig. 3(e).

4. Simulation results

Numerical simulations have been implemented in Visual basic programming language and tested in Windows environment using an Intel core i5 CPU 2.53 GHz processor. In our simulations we assumed simple small differential-drive robots, although the algorithm is totally general. The introduced algorithm has been tested on two different scenarios:

- local knowledge scenario. In this case the robot has limited sensing capabilities and is therefore able to locate obstacles inside a *sensing area*;
- global knowledge scenario. In this case the robot has unlimited sensing capabilities for obstacles detection.

The results are compared with three path planning algorithms: the algorithm based on Voronoi diagrams and Bezier curves [12], the VisBug algorithm [22] and the TangentBug algorithm [23].

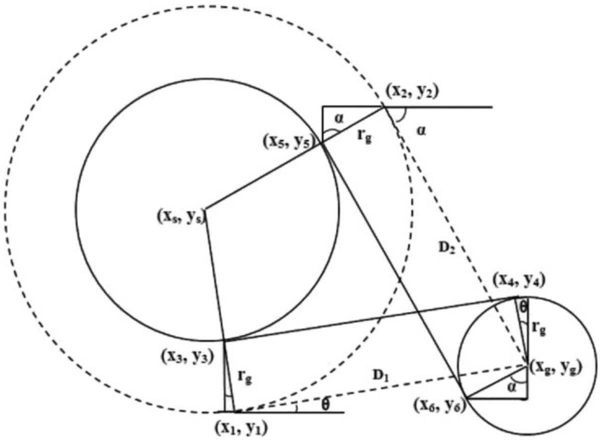

Fig. 6. Calculation of the inner tangent lines.

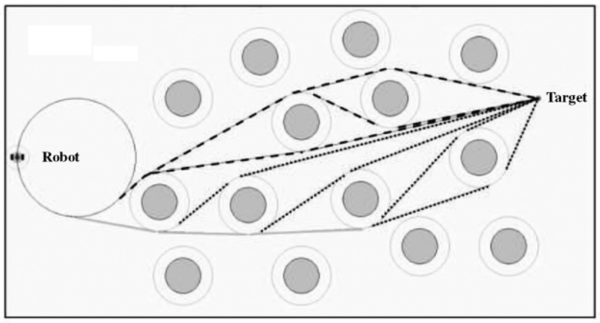

Fig. 7. Schematic illustration of step 3 of the shortest path search in the optimized binary tree.

These three algorithms are representative examples of path planning approaches based on Voronoi diagram (the first approach, which is actually already a version of the basic algorithm optimized to generate smooth trajectories) and on navigation based on graph search (VisBug and TangentBug). In particular, in the cases of VisBug and TangentBug, global and local information are combined together. In fact, the information on the target location is global, in the sense that it is known since the start of the navigation and independently of the initial robot position, while obstacles are detected only in their proximity.

The VisBug algorithm [22] has of two modes of motion: motion towards the target and obstacle boundary following. When the robot hits an obstacle it changes direction of movement from target towards boundary of obstacle. It then leaves the obstacle boundary when this ensures that the distance to the target decreases. Instead, the TangentBug algorithm [23] makes use of local data to compute a shortest path based on the so-called local tangent graph [24]. The local tangent graph is a graph including only the obstacles which are located within the robot sensing range. Additionally, this algorithm makes use of the same types of motion implemented in the VisBug algorithm: motion towards the target and obstacle boundary following. During motion towards the target, the robot moves along the shortest path computed through the local tangent graph. During obstacle boundary following, the TangentBug algorithm checks the leaving condition to decide when return to the motion-towards-target mode. In local knowledge scenarios, both VisBug and TangentBug algorithms produce near shortest trajectories to the target, with slightly better performance of TangentBug algorithm.

The visibility binary tree algorithm discussed in this paper shares with the VisBug and TangentBug algorithms the combination of global information (for the target) and local information for

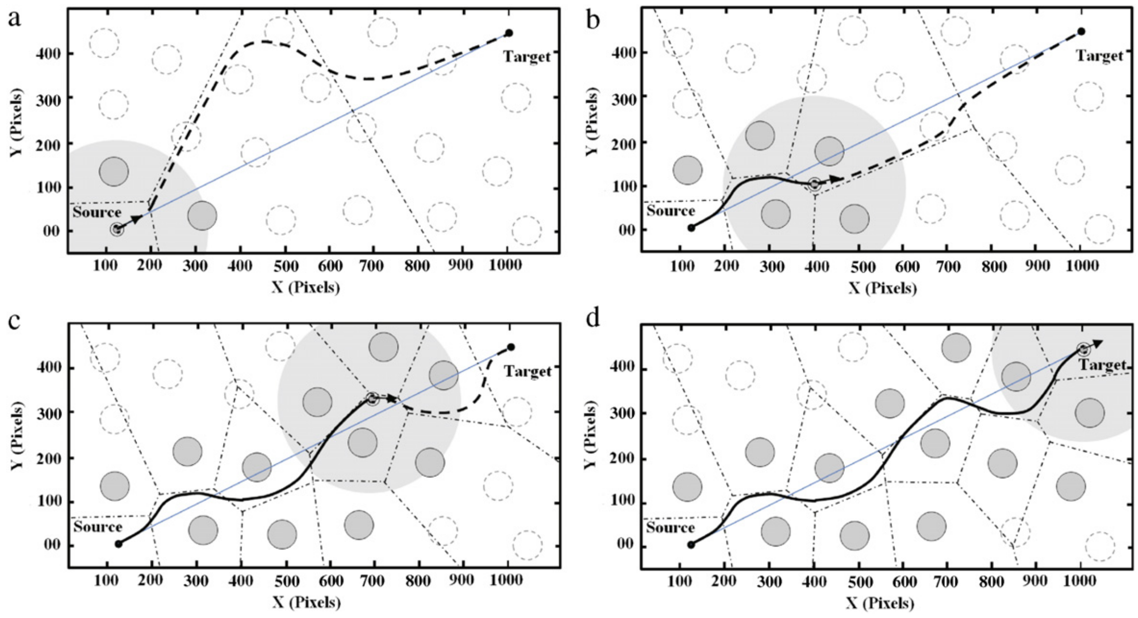

Fig. 8. Trajectory obtained by path planning using the algorithm based on Voronoi diagrams and Bezier curves [12] at different times: (a) $t = 0$ s; (b) $t = 3.31$ s; (c) $t = 7.17$ s; (d) $t = 11.03$ s.

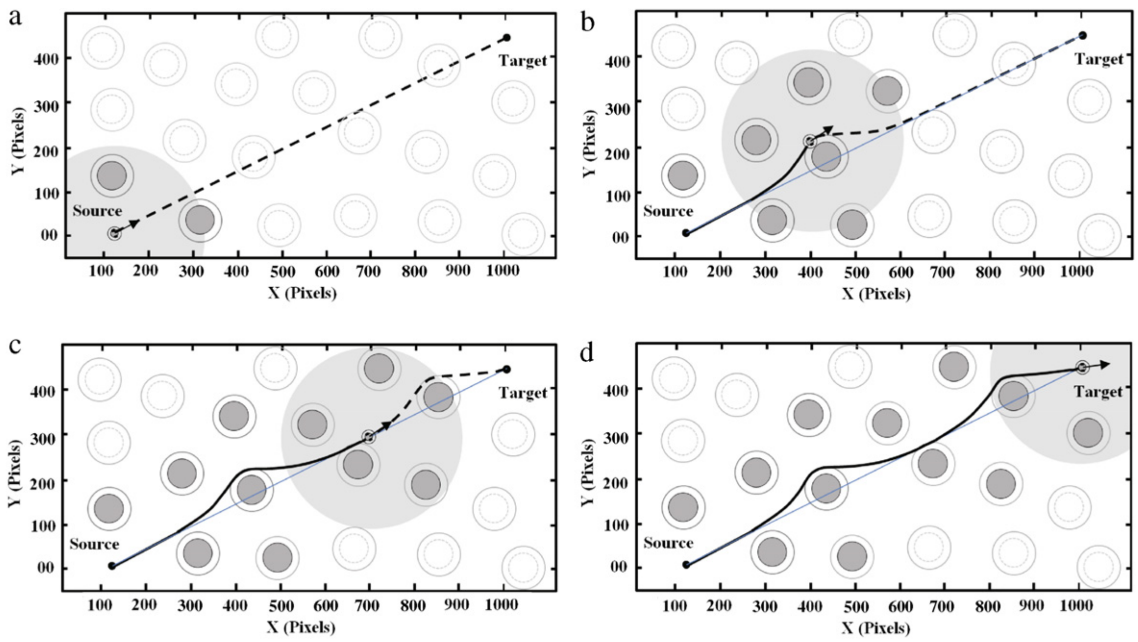

Fig. 9. Trajectory obtained by path planning using the VisBug algorithm at different times: (a) $t = 0$ s; (b) $t = 3.59$ s; (c) $t = 6.7$ s; (d) $t = 10.7$ s.

obstacles (this information then becomes global for large values of the robot sensing radius). The visibility binary tree algorithm makes use of the graphs like TangentBug to construct the path towards the target which avoids obstacles. The main differences of our algorithm with TangentBug rely in the way in which this graph is built, a way which produces benefits when the information on target location is limited. In fact, for scenarios in which the robot sensing radius is large (at the limit, scenarios with global knowledge of obstacle position), as we will show below, the two algorithms show similar performance.

We first discuss the local knowledge scenario and then, as a limit case obtained for large values of the sensing radius, the global knowledge scenario. We considered simulations under different conditions related to the number of obstacles, their dimension, and the robot sensing radius. Let us first consider, for illustration purpose, a scenario with 20 obstacles of radius equal to 60 pixels. For a qualitative analysis we report in Figs. 8–11 several snapshots of the trajectories generated by the four different algorithms on the same scenario. The robot speed has been fixed to 100 pixels/s and the sensing radius to 200 pixels. The visibility binary tree algorithm provides the trajectory with shorter length when compared to the result of the other three algorithms. The trajectory generated by the algorithm based on the Voronoi diagram is the longest, since it tries to keep the algorithm into a safe trajectory while the other algorithms drive the robot along trajectories which may be closer to the obstacles.

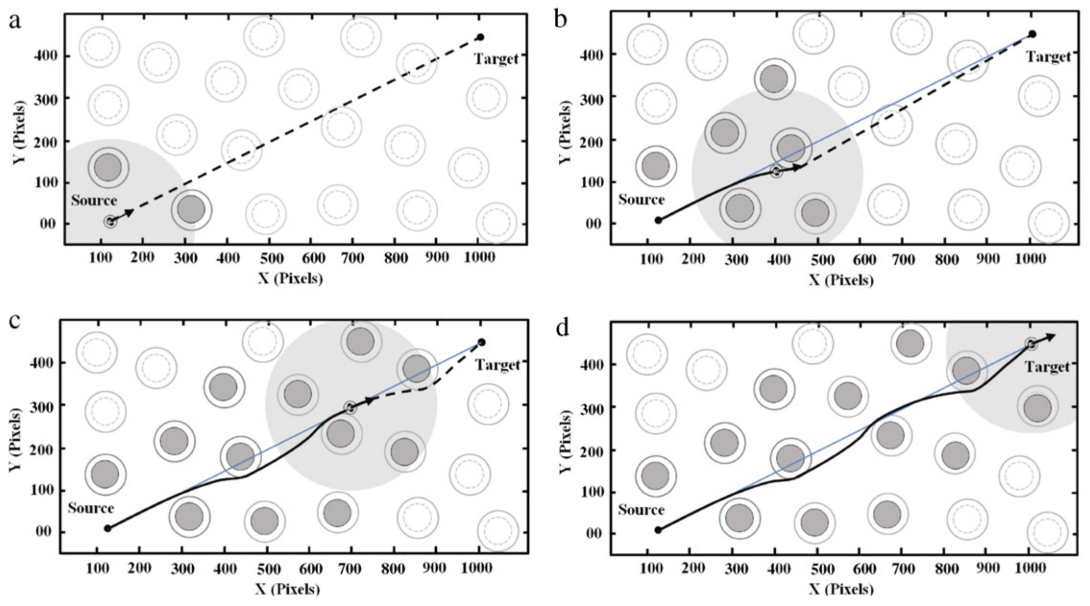

Fig. 10. Trajectory obtained by path planning using the TangentBug algorithm at different times: (a) $t = 0$ s; (b) $t = 3.09$ s; (c) $t = 6.68$ s; (d) $t = 10.48$ s.

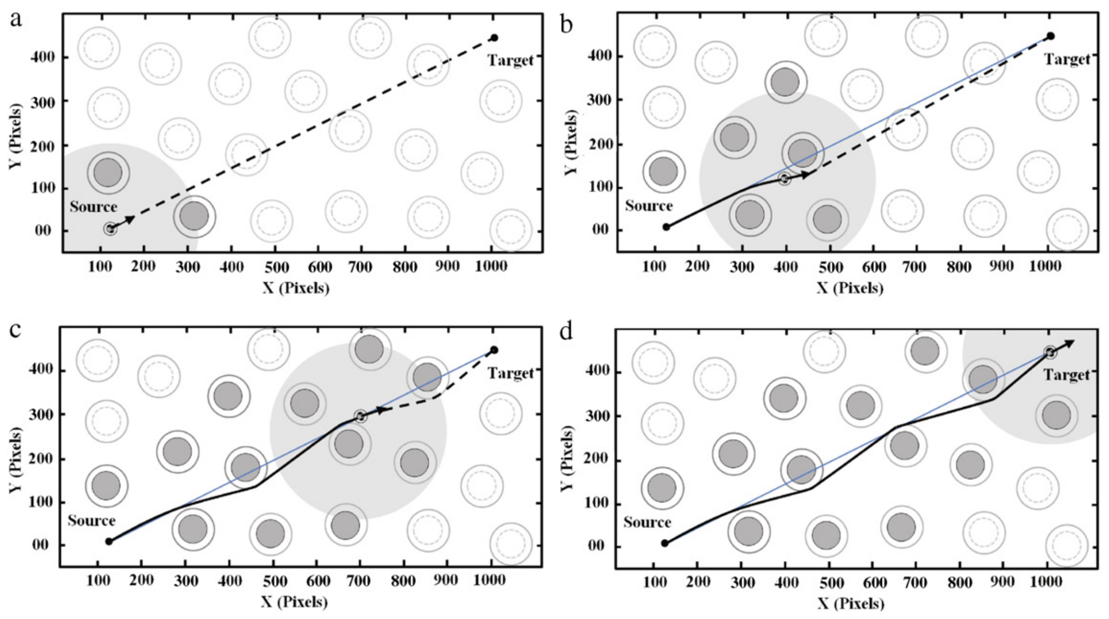

Fig. 11. Trajectory obtained by path planning using the visibility binary tree algorithm at different times: (a) $t = 0$ s; (b) $t = 3.03$ s; (c) $t = 6.66$ s; (d) $t = 10.37$ s.

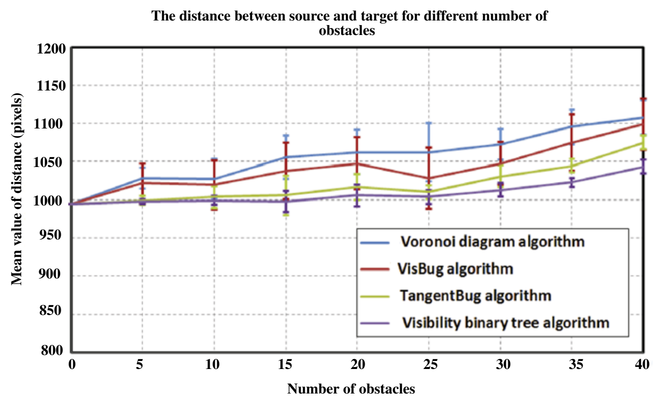

Fig. 12. Comparison of the performance of the introduced path planning algorithm with three other path planning algorithms for different values of the number of obstacles in a local knowledge scenario. Error bars represent standard deviation.

For a more quantitative comparison, simulations in this scenario have been repeated for a different number of obstacles, ranging from 0 to 40 at steps of 5. For each value of the number of obstacles, 10 different realizations have been considered, where at each realization a different distribution of the obstacles in the plane has been taken into account. For each path planning algorithm, average value and standard deviation of the distance between source and target have been then computed and reported in Fig. 12. The introduced algorithm has better performances for any value of the number of obstacles.

A second set of experiments in the local knowledge scenario refers to a scenario with a fixed number of obstacles (equal to 50) with different size (ranging from 50 to 140 pixels). Simulations in this scenario have been repeated for four different locations of the target. Results are summarized in Table 1 and in Fig. 13. From Table 1 we notice that the visibility binary tree algorithm has the best performance for all four target locations. The trajectories generated by the four algorithms are reported in Fig. 13.

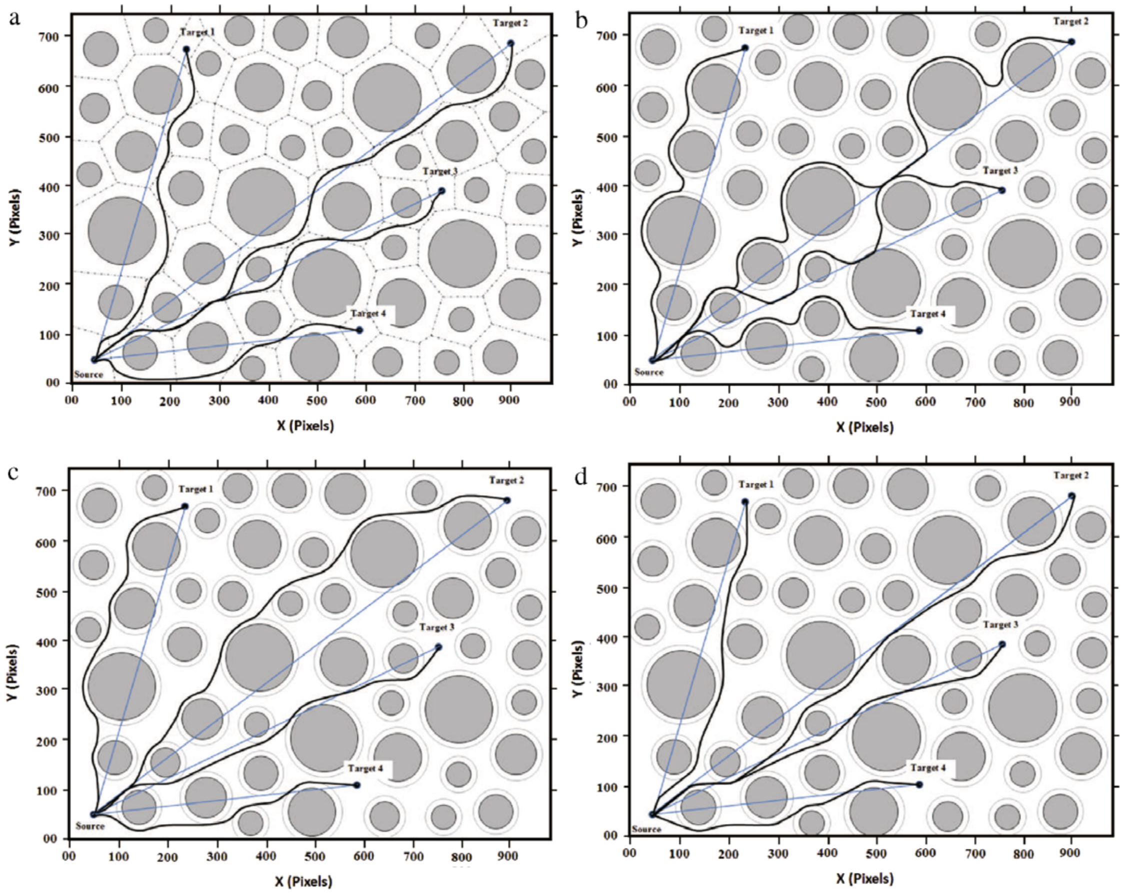

Fig. 13. Comparison of the performance in a local knowledge scenario with 50 obstacles with different sizes and four different target locations: (a) algorithm based on the Voronoi diagram; (b) VisBug algorithm; (c) TangentBug algorithm; (d) visibility binary tree algorithm.

Table 1
Performance on a local knowledge scenario with 50 obstacles with different sizes and four different target locations.

Target location	Voronoi diagram algorithm	VisBug algorithm	TangentBug algorithm	Visibility binary tree algorithm
(230, 660)	711	781	738	684
(900, 685)	1142	1454	1131	1104
(760, 390)	840	980	813	802
(590, 110)	598	662	566	555

The third set of simulations was aimed at evaluating the performance when the robot sensing radius is changed from 0 pixels (thus emulating a contact sensor) to unlimited values (thus emulating a scenario of global knowledge). Simulations have been performed on an environment with 30 obstacles with different radius values ranging from 50 to 140 pixels. For each value of the robot sensing radius analyzed, 10 different realizations have been considered. At each realization a different distribution of obstacles in the plane has been taken into account. An example of the trajectories generated by the four different algorithms in one of the realizations is illustrated in Fig. 14. The average value and the standard deviation of the distance between source and target in the 10 realizations have been then computed and reported in Fig. 15. The comparison shows that in most of the cases, which have taken into account a wide range of values of the sensing radius including the case of global knowledge, the visibility binary tree algorithm performs better (showing both a lower average value and a smaller standard deviation) than the other three algorithms implemented; it shows performance very similar to the TangentBug algorithm for large values of the sensing radius.

Concerning the computational time required by these algorithms, our algorithm has better performance than the approach based on Voronoi diagram, but poorer performance with respect to VisBug and TangentBug. The bottleneck of algorithms based on graph construction is, in fact, the search for the shortest path in the graph of possible trajectories to the target. Therefore, our algorithm benefits from the reduced computational time due to the simplification of the graph structure with respect to approaches considering the whole tangent visibility graph, but does not reach the time performance of VisBug and TangentBug, in which on the contrary the graph structure is so simplified that the presence of the more convenient path to the target is not guaranteed, so that they often result in a non-optimal choice of the trajectory to the target. As an example, in a scenario with 50 obstacles with sizes ranging from 50 to 140 pixels, robot speed fixed to 100 pixels/s, distance to the target equal to 1080 and robot sensing radius equal to 200 pixels, on an Intel core i5 CPU 2.53 GHz processor the approach based on Voronoi diagrams requires 14 min 7 s, VisBug 1 min 28 s, TangentBug 1 min 26 s and the visibility binary tree algorithm 4 min 47 s.

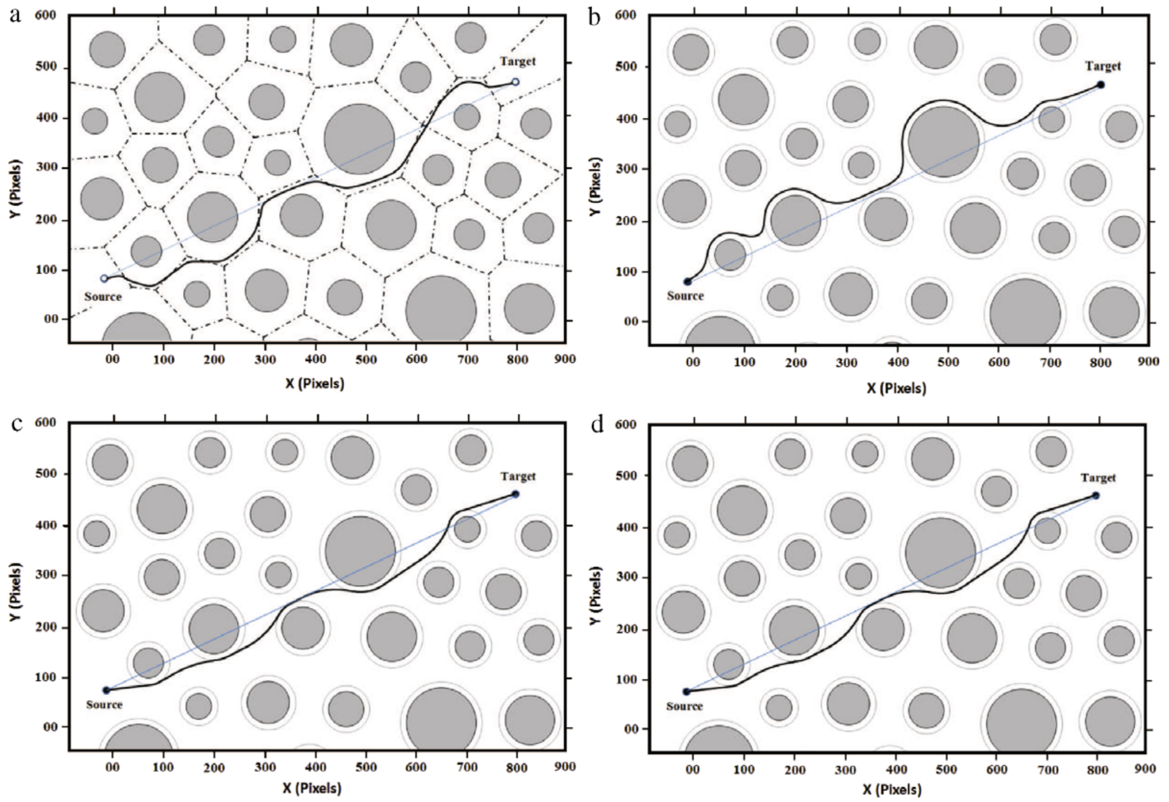

Fig. 14. Environment used for the study of the effect of different values of the robot sensing radius on path planning performances and trajectory obtained for a sensing radius equal to 100 pixels: (a) algorithm based on the Voronoi diagram; (b) VisBug algorithm; (c) TangentBug algorithm; (d) visibility binary tree algorithm.

5. Conclusions

In this paper, an algorithm for path planning of mobile robots in the presence of obstacles, named visibility binary tree algorithm, has been introduced in local and global knowledge scenarios. In the description of the algorithm for path planning, we have described both the low and the high level of path planning. The first part of the paper is devoted to introduce the use of Bresenham algorithms for low level path planning, where the trajectory of robot to be computed is represented as a set of successive points in the plane, while the second part focuses on the graph construction, optimization and search for planning the trajectory of the robot. The use of Bresenham algorithms for this purpose is motivated by the fact that they offer the advantage of being a fast incremental algorithm, using only integer calculations. This is a characteristic making this algorithm particularly suitable for implementation in hardware systems with limited available resources. In illustrating both levels of path planning, we have thus shown how the whole algorithm is suitable for implementation with limited available resources.

The introduced algorithm has been described in details and simulation results comparing the proposed algorithm with three particularly important different approaches to the problem of path planning have been presented. In particular, since the literature on the subject is vast and rapidly growing, we have chosen to compare the proposed algorithm with one method that can be considered as a benchmark for path planning and two other algorithms sharing some important features with the one we have proposed. The main result is that the proposed algorithm has good performance in all the scenarios considered (which include examples with local and global knowledge, different number of obstacles, different sizes of obstacles and different values of the robot sensing radius). In particular, in terms of optimality of the trajectory it performs better than the approach based on the Voronoi diagram and the VisBug algorithm. Compared with the TangentBug algorithm, it has the same performance in the global knowledge scenario and better performance when information on the obstacle locations is limited. In terms of computational time, the approach performs much better that the approach based on Voronoi diagrams, but does not reach the performance of VisBug and TangentBug algorithms.

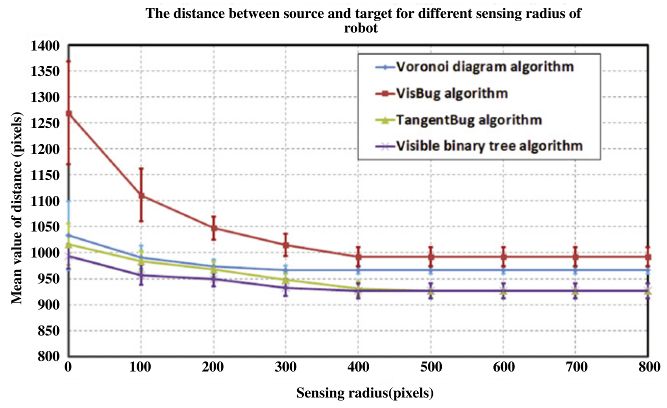

Fig. 15. Comparison of the performance of the introduced path planning algorithm with three other path planning algorithms for different values of the robot sensing radius. Error bars represent standard deviation.

References

[1] M. De Berg, O. Cheon, M. va Krevel, Computational Geometry, Springer, Berlin, Germany, 2000.
[2] S. Ghosh, J. Burdick, A. Bhattacharya, S. Sarkar, Algorithms with discrete visibility-exploring unknown polygonal environments, IEEE Robotics and Automation Magazine 15 (698) (2008) 67–76.
[3] G. Foux, M. Heymann, A. Bruckstein, Two dimensional robot navigation among unknown stationary polygonal obstacles, IEEE Transactions on Robotics and Automation (1994) 1622–1627.
[4] W.D. Rencken, Concurrent localization and map building for mobile robots using ultrasonic sensors, in: Proceeding of the IEEE/RSJ Conference on Intelligent Robots and Systems, IROS, 1993, pp. 2192–2197.

[5] P. Bhattacharya, M.L. Grvrilova, Voronoi diagram in optimal path planning, in: 4th IEEE International Symposium on Voronoi Diagrams in Science and Engineering, 2007, pp. 38–47.
[6] S. Mohammadi, N. Hazar, A Voronoi-based reactive approach for mobile robot navigation, in: Advances in Computer Science and Engineering, Vol. 6, Springer, Berlin, Heidelberg, 2009, pp. 901–904.
[7] P. Bhattacharya, M. Gavrilova, Roadmap-based path planning- using the Voronoi diagram for a clearance-based shortest path, IEEE Robotics and Automation Magazine 15 (2) (2008) 58.66.
[8] The 2005 DARPA Grand Challenge, Vol. 36, Springer, Berlin, Heidelberg, 2007, pp. 363–405.
[9] A. Sahraei, M.T. Manzuri, M.R. Razvan, M. Tajfard, S. Khoshbakht, Real-time trajectory generation for mobile robots, in: The 10th Congress of the Italian Association for Artificial Intelligence, AIIA 2007, September, 2007, pp. 10–13.
[10] J. Choi, R.E. Curry, G.H. Elkaim, Path planning based on bezier curve for autonomous ground vehicles, in: IAENG Transactions on Electrical and Electronics Engineering Volume I—Special Edition of the World Congress on Engineering and Computer Science 2008, IEEE Computer Society, 2009, pp. 158–166.
[11] J.W. Choi, R.E. Curry, G.H. Elkaim, Continuous curvature path generation based on bezier curves for autonomous vehicles, IAENG International Journal of Applied Mathematics 40 (2010).
[12] J.W. Choi, R.E. Curry, G.H. Elkaim, Real-time obstacle avoiding path planning for mobile robots, in: Proceedings of the AIAA Guidance, Navigation and Control Conference, AIAA GNC 2010, Toronto, Ontario, Canada, 2010.
[13] T. Lizano-Perez, M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, Communications of the ACM 22 (10) (1979).
[14] T. Lizano-Perez, Automatic planning of manipulator. transfer movements, IEEE Transactions on Systems, Man, and Cybernetics SMC-11 (1981).
[15] Y.H. Liu, S. Arimoto, Proposal of tangent graph and extended tangent graph for path planning of mobile robots, in: Proc. IEEE Inr. Con. Robot. Automat., 1991, pp. 312–317.
[16] Y.-H. Liu, S. Arimoto, Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph, IEEE Transactions on Robotics and Automation 11 (1995) 682–691.
[17] T. Asano, L. Guibas, J. Hershberger, H. Imai, Visibility of disjoint polygons, Algorithmica 1 (1) (1986).
[18] L. Cesari, Optimization. Theory and Applications: Problems with Ordinary Differential Equations, Springer-Verlag, New York, 1983.
[19] C. Alexopolous, P.M. Griffin, Path planning for a mobile robot, IEEE Transactions on Systems, Man, and Cybernetics 22 (1992) 318–322.
[20] J.E. Bresenham, Algorithm for computer control of a digital plotter, IBM Systems Journal 4 (1) (1965) 25–30.
[21] J.E. Bresenham, Ambiguities in incremental line rastering, IEEE Computer Graphics and Applications 7 (5) (1987).
[22] V.J. Lumelsky, T. Skewis, Incorporating range sensing in the robot navigation function, IEEE Transactions on Systems, Man, and Cybernetics 20 (5) (1990) 1058–1068.
[23] I. Kamon, E. Rivlin, A new range-sensor based globally algorithm for mobile, in: Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, Minnesota—April 1996.
[24] Y.H. Liu, S. Arimoto, Path planning using a tangent graph for mobile robots among polygonal and curved obstacles, International Journal of Robotics Research 11 (4) (1992) 376–382.

Abdulmuttalib Turky Rashid was born in Iraq. He received the B.S. degree in electrical engineering from Basrah University at Basrah, Iraq in 1986. He received the M.Sc. Degree from the same University at 1992. He worked as Assistant Lecturer and Lecturer at the Department of Electrical Engineering University of Omer Al Mukhtar, Lybia at 1997–2007; then at the Department of Electrical Engineering, University of Basrah, Iraq at 2007 up to now. His field of Interest is Robotics and Industrial control. Currently, he is pursuing Ph.D. degree in Electrical engineering in University of Basrah, since 2009. His research interests are in motion planning and control of multi mobile robots.

Abduladhem Abdulkareem Ali: He received his M.Sc. and Ph.D. Degrees from the Department of Electrical Engineering University of Basrah, Iraq at 1983 and 1996. Worked as Assistant Lecturer, Lecturer, and Assistant professor at the same Department at 1984, 1987 and 1981 respectively. Then as Assistant professor and professor at the Department of Computer Engineering at 1997 and 2004 respectively. Worked as consultant to many industrial firms to design industrial control systems. Have more than 70 published papers, one patent, supervised many M.Sc. and Ph.D. Dissertations. He is an Editor chair for the Iraqi Journal for Electrical and Electronic Engineering and a member of the editorial board for many Journals. Chairman of the first IEEE International conference on Energy, Power and Control (EPC-IQ01). His field of Interest is Robotics, Industrial control and Intelligent systems. He is currently the Director of Avicenna E-learning center at University of Basrah, Iraq.

Mattia Frasca was born in Siracusa, Italy, in 1976. He graduated in Electronics Engineering in 2000 and received the Ph.D. in Electronics and Automation Engineering in 2003, from the University of Catania, Italy. Currently, he is research associate at the University of Catania. His scientific interests include nonlinear systems and chaos, Cellular Neural Networks, complex systems and bio-inspired robotics. He is involved in many research projects and collaborations with industries and academic centers. He is referee for many international journals and conferences. He was in the organizing committee of the 10th ''Experimental Chaos Conference'' and co-chair of the ''4th International Conference on Physics and Control''. He is coauthor of three research monographs (with World Scientific): one on locomotion control of bio-inspired robots, one on self-organizing systems and one on the Chua's Circuit. He published more than 150 papers on refereed international journals and international conference proceedings and is coauthor of two international patents. He is IEEE Senior.

Luigi Fortuna (M'90-SM'99-F'00) was born in Siracuse, Italy, in 1953. He received the degree of electrical engineering (cum laude) from the University of Catania, Catania, Italy, in 1977. He is a Full Professor of system theory with the Università degli Studi di Catania, Catania, Italy. He was the Coordinator of the courses in electronic engineering and the Head of the Dipartimento di Ingegneria Elettrica Elettronica e dei Sistemi (DIEES). Since 2005, he has been the Dean of the Engineering Faculty, Catania. He currently teaches complex adaptive systems and robust control. He has published more than 450 technical papers and is the coauthor of ten scientific books, among which: Chua's Circuit Implementations (World Scientific, 2009), Bio-Inspired Emergent Control of Locomotion Systems (World Scientific, 2004), Soft-Computing (Springer 2001), Nonlinear Non Integer Order Circuits and Systems (World Scientific 2001), Cellular Neural Networks (Springer 1999), Neural Networks in Multidimensional Domains (Springer 1998), Model Order Reduction in Electrical Engineering (Springer 1994), and Robust Control—An Introduction (Springer 1993). His scientific interests include robust control, nonlinear science and complexity, chaos, cellular neural networks, soft computing strategies for control, Robotics, micro–nanosensor and smart devices for control, and nanocellular neural networks modeling.

Dr. Fortuna was the IEEE Circuits and Systems (CAS) Chairman of the CNN Technical Committee, IEEE CAS Distinguished Lecturer from 2001 to 2002, and IEEE Chairman of the IEEE CAS Chapter, Central-South Italy.