Robotics and Autonomous Systems 75 (2016) 119–128

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

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

An algorithm for multi-robot collision-free navigation based on shortest distance

Abduladhem A. Ali[a,*], Abdulmuttalib T. Rashid[b], Mattia Frasca[c], Luigi Fortuna[c]

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

HIGHLIGHTS

- A new algorithm for collision-free multi-robot navigation is introduced.
- The new algorithm is based on shortest distance algorithm.
- It is particularly efficient and easy to implement.
- Comparison with previously discussed algorithms on different standard scenarios is presented.

ARTICLE INFO

Article history:
Received 21 March 2015
Received in revised form
15 October 2015
Accepted 23 October 2015
Available online 31 October 2015

Keywords:
Navigation
Multi-robots
Collision avoidance

ABSTRACT

This paper presents a new approach for multi-robot navigation in dynamic environments, called the shortest distance algorithm. This approach uses both the current position and orientation of other robots to compute the collision free trajectory. The algorithm suggested in this paper is based on the concept of reciprocal orientation that guarantees smooth trajectories and collision free paths. All the robots move either in a straight line or in a circular arc using the Bresenham algorithms. The current approach is tested on three simulation scenarios.

1. Introduction

This paper addresses the problem of motion planning of multiple mobile robots in dynamic environments. This problem, arising in the presence of mobile obstacles or multi-robot systems, is more difficult than the static problem, that is, the case arising when obstacles are fixed, since it requires the real-time solution of the path planning [1]. In fact, when multiple mobile robots share the same environment, collisions among them must be taken into account. Collision avoidance may be achieved by using a centralized approach to plan the trajectories of all the robots [2,3] or by planning each trajectory separately and using a centralized approach to coordinate these paths [4,5]. However, kinematic constraints are not taken into account in most of the centralized approaches. Only the earliest methods for path planning that use car like robots deal with kinematic constraints [6]. Such car like models are controlled to forward motions with a fixed speed and limited turning radius and modified by adding a reverse gear [7]. Their control is farther extended considering variable speed in any direction [8,9].

In this paper, we develop an approach for collision-free navigation of multiple robots with differential drive kinematics. Groups of coordinated differential-drive robots may be used for environmental monitoring, and rescue applications [10]. Hence, it is important to develop methods to investigate smooth and collision-free paths for these kinds of robots. Smooth trajectories are not guaranteed in most of the earlier methods and the investigation of collision-free paths is limited to single robots moving amongst dynamic obstacles.

Many works in robotics have addressed the problem of collision-free navigation of a robot in dynamic environments with moving obstacles [11–13]. Most approaches rely on the prediction of future locations of obstacles by extrapolating their current velocities, and let the robot avoid collisions accordingly. However, such approaches do not suffice when the robot encounters other robots, because treating the other robots as moving obstacles

* Corresponding author.
E-mail address: abduladem1@yahoo.com (A.A. Ali).

http://dx.doi.org/10.1016/j.robot.2015.10.010
0921-8890/© 2015 Elsevier B.V. All rights reserved.

overlooks the reciprocity principle. Hence, the future trajectories of other robots cannot be estimated by extrapolating their current velocities, and doing so could inherently cause undesirable oscillations in the motion of the robots [14,15].

Several variations of the velocity obstacle method have been proposed for multi-robot systems, generally attempting to incorporate the reactive behavior of the other robots in the environment. Variations such as reciprocal velocity obstacles [16], recursive probabilistic velocity obstacles [17], artificial bee colony algorithm [18], use various means to handle reciprocity, but each has its own shortcomings. Some of these methods are limited to manipulating only two robots, while others fail to bring robots together.

The concept of reciprocal orientation for collision avoidance has been introduced amongst robots that specifically consider this reciprocity [19]. Reciprocity lets a robot take half of the responsibility of avoiding collisions with another robot and assumes that the other robot takes care of the other half. The concept of reciprocal orientation guarantees both smooth and collision-free robot trajectories and avoid the oscillations in the motion of the robots. According to this method, each robot executes an independent continuous cycle of sensing and acting, in which a robot chooses its new direction based on observations of the positions and velocities of the other robots.

In this paper, the shortest distance algorithm (SDA) is suggested as a new algorithm to solve the problem of multi-robot navigation control taking into account the principle of reciprocity. The main idea of the algorithm is to move two eventually colliding robots in a straight line to the positions corresponding to the shortest distance they can have without bumps and, then, to start the maneuver for collision avoidance. In this way, the trajectory generated is at the same time simple and short as it will be discussed in the rest of the paper that is organized as follows. Section 2 describes the low level control of a differential-drive robot. Section 3 describes the reciprocal orientation algorithm (ROA). Section 4 introduces the shortest distance algorithm (SDA). In Section 5, simulations of our multi-robot navigation approach in four challenging scenarios are presented. Finally, Section 6 draws the conclusion of the paper.

2. Low level control of the robot

The low-level control aims at implementing the procedures needed to a robot to follow a given trajectory. This includes the kinematics of the differential-drive mobile robot and the Bresenham algorithms, which are used to implement robot motion along a straight line or an arc line.

Differential-drive robots use a simple drive mechanism that consists of two drive wheels mounted on a common axis. Each wheel can be independently driven in either forward or reverse direction. We also assume that no-slip happens between the robot's tire and the floor during motion. Most mobile robots in practical services like vacuum cleaners [20] and powered wheelchairs [21] have differential-drive mechanisms. In addition, differential-drive robots are often used to build up a distributed system of multiple mobile robots.

A schematic illustration of a differential-drive mobile robot is shown in Fig. 1. Let the coordinates of point $(x_s,\ y_s)$ define the position of the robot in a global frame [22]. Consider a line that is perpendicular to the wheel axis and goes through the point $(x_s,\ y_s)$ as an orientation reference for the robot. The angle that this line makes with the positive x-axis is θ. Assume that v_r and v_l are the right and the left velocities of the two wheels and L is the distance between two wheels. The speeds of the two wheels control the robot motion: when they are equal, straight-line motion is attained, while for different speeds the vehicle trajectory follows a circular arc.

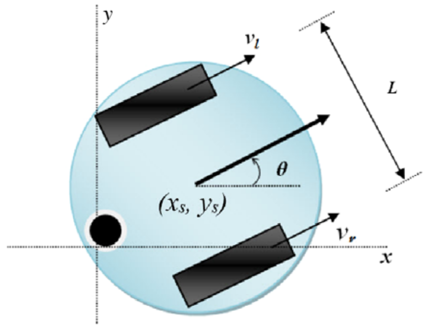


Fig. 1. Schematic illustration of the kinematics of a differential-drive mobile robot.

The Bresenham algorithms have been originally developed for drawing lines and circles on a pixelated display system [23], but then have been also used in low level control to represent the trajectory of a robot by a set of successive pixels in the display system. In our environment the set of successive pixels is replaced by a set of successive points. There are several useful characteristics in these algorithms that make them appealing for low level control of navigation. They are fast incremental algorithms and use only integer calculations. In all multiplications appearing in these algorithms one of the two terms is 2, meaning that the products can be accomplished with a simple shift left instruction. These characteristics imply that these algorithms need limited calculations and are suitable to be used with small mobile robots equipped with limited computational resources.

Let us first discuss the way to implement a straight line trajectory using the Bresenham line algorithm. Fig. 2 shows the space where the robot moves as a two-dimensional grid of points. Assume that the robot moves from source point $(x_S,\ y_S)$ to the target point $(x_T,\ y_T)$ following a straight line. The algorithm computes the sequence of points in the grid in which the robot has to move from source to target. At each step the algorithm chooses the next position along the x-axis (i.e., from x_k to $x_k + 1$ where k is an index labeling the points in the grid). Then the algorithm selects the y coordinate closest to the line between source and target (selecting either y_k or y_k+1). The following steps summarize the algorithm:

1. Choose the source point $(x_S,\ y_S)$ and target point $(x_T,\ y_T)$ and calculate Δx and Δy
$$\begin{aligned}\Delta x &= x_T - x_S\\ \Delta y &= y_T - y_S.\end{aligned} \tag{1}$$
2. Determine the first value of the decision parameter p_o as:
$$p_o = 2\Delta y - \Delta x. \tag{2}$$
3. For each x_k along the line from source point to target point, starting at $k = 0$, do the following test: if $p_k < 0$, the next point to choose is $(x_k + 1,\ y_k)$ and:
$$p_{k+1} = p_k + 2\Delta y \tag{3}$$
otherwise, the next point to plot is $(x_k + 1,\ y_k + 1)$ and:
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x. \tag{4}$$
4. Repeat step 3 until $(x_T,\ y_T)$ is reached.

The Bresenham circle algorithm is used to let the robot move along a circular trajectory. The algorithm assumes that this circular trajectory is centered on the origin $(x_c,\ y_c)$ and divided into eight symmetrical parts. The computation of robot positions is performed in one of the octants. At each step, in this algorithm, the new position $x_k + 1$ is selected with the y coordinate (y_k or $y_k - 1$) closest to the circle (see Fig. 3). The selection of the y coordinate depends on the value of the decision parameter p_k. This algorithm can be executed in an incremental way by the following steps:

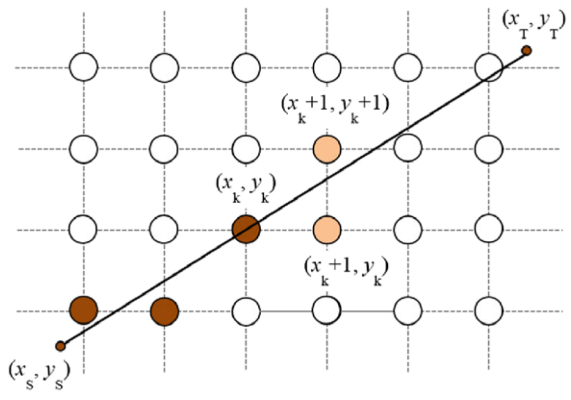


Fig. 2. Illustration of the Bresenham line drawing algorithm.

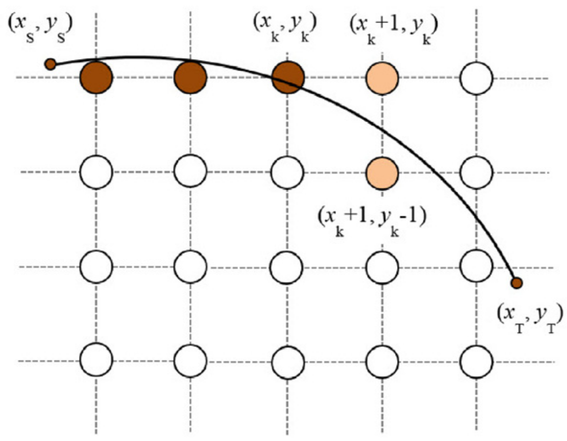


Fig. 3. Illustration of the Bresenham circular arc algorithm.

1. Input the radius r and the circle center $(x_c,\ y_c)$, and consider the coordinates for the left top point (where the robot starts) in the two-dimensional grid as:

$$(x_S,\ y_S) = (0, r). \tag{5}$$

2. Calculate the initial value of the decision parameter p_o as:

$$p_o = 5/4 - r. \tag{6}$$

3. At each position x_k, from source point to target point, perform the following test. If $p_k < 0$, the next point along the circle is $(x_{k+1},\ y_k)$ and:

$$p_{k+1} = p_k + 2x_{k+1} + 1. \tag{7}$$

Otherwise, the next point along the circle is $(x_k + 1,\ y_k - 1)$ and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}. \tag{8}$$

4. Determine symmetrical points in the other seven octants.
5. Calculate the coordinates with respect to the original reference frame.
6. Repeat steps 3–5 until $x_k \geq y_k$.

3. Reciprocal orientation algorithm

The reciprocal orientation algorithm (ROA), introduced in [19], is an approach to obtain collision free trajectories in multi-robot scenarios based on the idea of reciprocal orientation. To better illustrate the characteristics of SDA, the approach underlying ROA is here summarized. In ROA reciprocal orientation is attained thanks to the following algorithms:

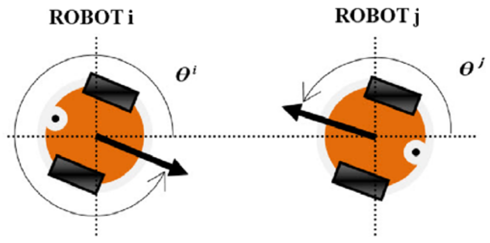


Fig. 4. Illustration of the rotation algorithm for two robots.

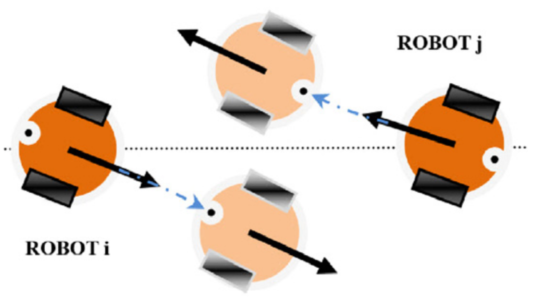


Fig. 5. Computation of robot trajectories.

1. Rotation algorithm: this algorithm helps to avoid the collision between two robots by choosing a suitable orientation, as shown in Fig. 4. In this algorithm we assume that the two robots are aligned to the horizontal axis of the environment. The movement directions of both robots depend on their current orientation. The new direction is chosen so that the two robots do not skew far from their targets. Each of two robots executes this process by taking into account the observed direction of the other robot.
2. Computation of robot trajectories: this algorithm is used to calculate the circular trajectories of two robots, as shown in Fig. 5. The information needed for the trajectory calculation is the initial positions of robots, their radiuses, their velocities and directions which are provided by the rotation algorithm. The next position of each robot and the distance between them is computed by using the following equations:

$$\begin{aligned} x^i_{h+1} &= x^i_h + v\,\Delta t\,\cos\theta^i \\ y^i_{h+1} &= y^i_h + v\,\Delta t\,\sin\theta^i \end{aligned} \tag{9}$$

$$d^{ij}_{h+1} = \left(\left(y^i_{h+1} - y^j_{h+1}\right)^2 + \left(x^i_{h+1} - x^j_{h+1}\right)^2\right)^{1/2} \tag{10}$$

where $(x^i_h,\ y^i_h)$ is the initial position of robot i, Δt is the time interval, v is the velocity of each robot, θ_i is the orientation of robot i, d^{ij}_{h+1} is the distance between two robots i and j at next time interval and $(x^i_{h+1},\ y^i_{h+1})$ is the position of robot i at next time interval.

Eqs. (9) and (10) are repeated for all the positions of two robots on their trajectories from sources to targets. At each time interval, we must avoid the case leading to a value of d^{ij}_{h+1} less than the sum of the radiuses of two robots. This constraint produces trajectories without collisions.
3. Two-dimensional transformation algorithm: since all the above algorithms rely on the assumption that the two robots are located in a horizontal line they are suitable only under this particular hypothesis. To make them effective in the more general case of arbitrary alignment, we must find the virtual locations of these robots with horizontal alignment. This process is accomplished by rotating the environment around the center of one robot using a transformation algorithm [24]. In this algorithm, the virtual locations for robots are found by rotating them until the horizontal

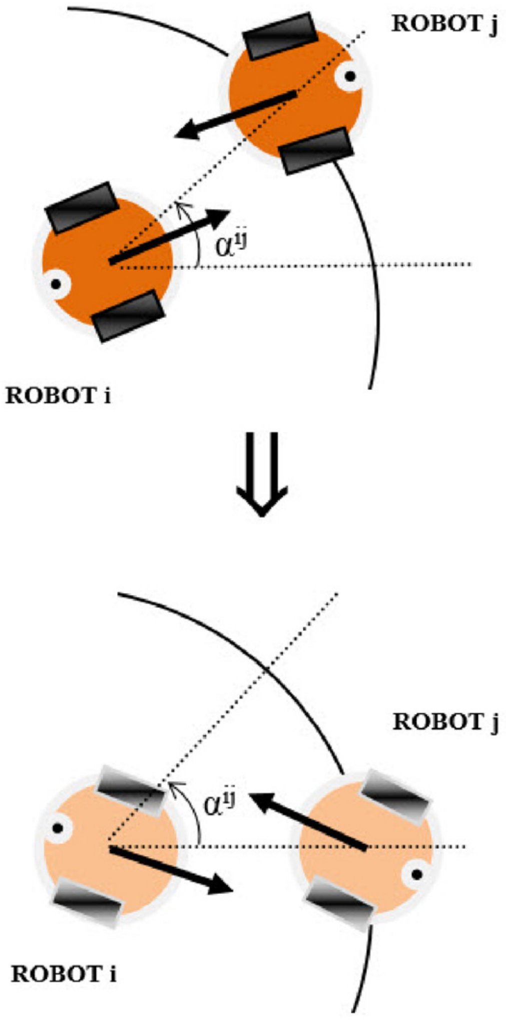


Fig. 6. Illustration of the two-dimensional transformation algorithm.

axis, as shown in Fig. 6. Now the reciprocal orientation algorithm can be applied on the virtual locations of robots to find the new virtual locations with no collision between them. After that, the virtual locations of robots are returned to original locations by applying the transformation algorithm in reverse direction. More in details the computation of the virtual locations is performed in the following way.

According to Fig. 6, the alignment line between the centers of two robots has an angle α^{ij} with respect to the horizontal axis. The value of this angle is given by

$$\alpha^{ij} = \tan^{-1}\left(\left(y_h^i - y_h^j\right) \Big/ \left(x_h^i - x_h^j\right)\right). \tag{11}$$

Since, we assume that the rotation transformation algorithm is applied around the center of robot i (the origin of the new reference frame), the axes are translated according to following equations:

$$\begin{aligned} x &= x^j - x^i \\ y &= y^j - y^i. \end{aligned} \tag{12}$$

The equations of rotation are:

$$\begin{aligned} x_r^i &= x\cos\alpha^{ij} - y\sin\alpha^{ij} \\ y_r^i &= x\sin\alpha^{ij} + y\cos\alpha^{ij} \end{aligned} \tag{13}$$

where x_r^i and y_r^i represent the position for each point in robot i after rotation.

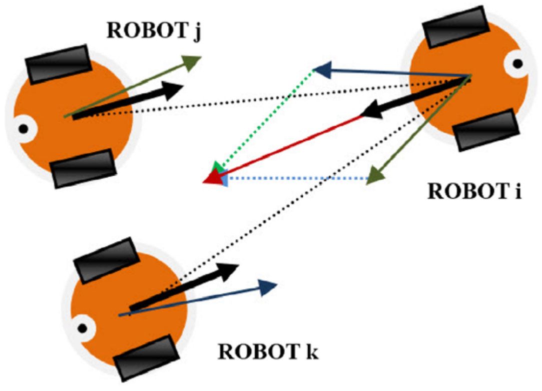


Fig. 7. Illustration of the combined reciprocal orientation algorithm.

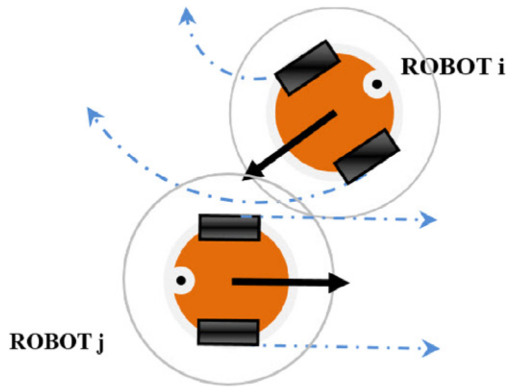


Fig. 8. Illustration of the resolution algorithm for deadlocks.

The virtual locations of each robot are obtained by using the following equations:

$$\begin{aligned} x_v^i &= x_r^i + x^j \\ y_v^i &= y_r^i + y^j \end{aligned} \tag{14}$$

where x_v^i and y_v^i represent the virtual location for each point in robot i.

Now we can compute the virtual trajectories with no collision between two robots.

Once this has been performed, to obtain the actual trajectories, we apply the inverse transformation using the following equations:

$$\begin{aligned} x &= x_r^i\cos\alpha^{ij} + y_r^i\sin\alpha^{ij} \\ y &= -x_r^i\sin\alpha^{ij} + y_r^i\cos\alpha^{ij}. \end{aligned} \tag{15}$$

4. Combined reciprocal orientation algorithm: the concept of the reciprocal orientation can be extended to multi-robots systems, by using the combined reciprocal orientation algorithm as graphically illustrated in Fig. 7. The idea is to apply the reciprocal orientation algorithm by pairing the robot i with all the other vehicles in its sensing range and, then, to sum the orientations obtained.
5. Resolution algorithm for deadlocks: deadlocks among two or more robots occur if a number of robots blocks each other in a way such that none of them is able to continue along its trajectory without causing a collision (Fig. 8). After a deadlock is detected, an alternative trajectory is obtained by increasing the angular velocity of the robot. With this simple strategy, the deadlock is solved if it is detected at a sufficiently large distance.

4. Shortest distance algorithm

In this section, the new approach for collision free multi-robot navigation, called shortest distance algorithm (SDA), is

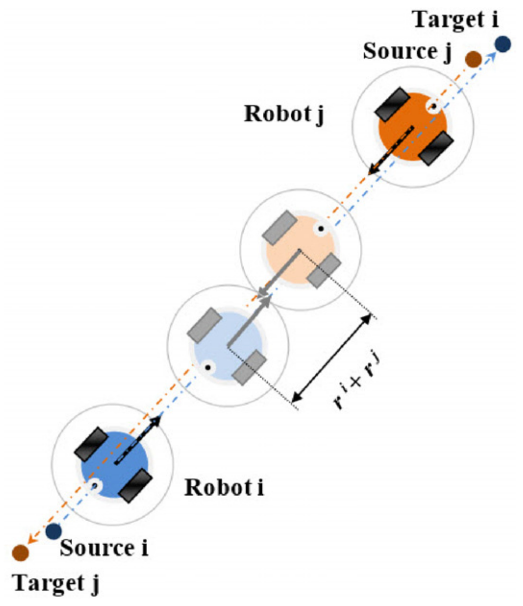


Fig. 9. Illustration of step 2.

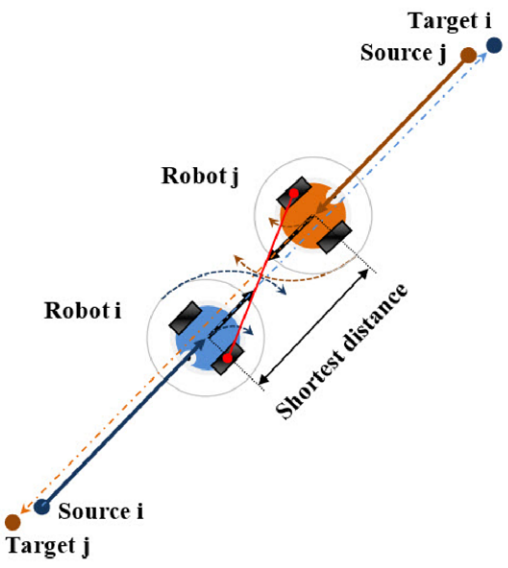


Fig. 10. Illustration of calculation of the shortest distance. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

introduced. Similarly to ROA, this approach also takes into account the observed direction of the other robots in order to avoid collisions with them. However, in SDA a different approach for trajectory planning is used. The main idea of the algorithm can be illustrated by taking into account the case in which two robots lie in a trajectory which will lead them to collide. The idea is to let the robots move in a straight line to the positions corresponding to the shortest distance they can have. At this time, they start a maneuver for collision avoidance, which can be simply accomplished by using a circular arc trajectory. In this way, two advantages are obtained: (i) two colliding robots travel the maximum allowed distance in a straight direction before starting their collision avoidance maneuver; (ii) the approach makes use only of trajectories generated by Bresenham algorithms and, in doing so, is particularly efficient in terms of computational efforts. The algorithm consists of the following steps:

Step 1. Measurement of the actual values of the variables: position of robot i and robot j $((x^i, y^i), (x^j, y^j))$, velocities of robot i and robot j (v^i, v^j), directions of robot i and robot j (θ^i, θ^j). Further inputs are the following constants: position of source i and source j $((x^i_S, y^i_S), (x^j_S, y^j_S))$, position of target i and target j $((x^i_T, y^i_T), (x^j_T, y^j_T))$, radiuses of robot i and robot j (r^i, r^j) and the distance between two wheels for each robot (L).

Step 2. Calculation of the straight line trajectories up to positions corresponding to a distance equal to $2r$: given the variables measured in step 1, the future positions of robots i and j in a straight line up to positions at a distance equal to two times the radius are computed. This calculation is done by using the Bresenham straight line algorithm. The robots are not moved in this step and this position is calculated only to facilitate the next step (in Fig. 9 these positions are shadowed). In fact, robots need a larger space to perform their maneuver to avoid collision. This is the task of step 3. However, knowing this virtual position facilitates the choice of the direction of rotation making the robot not far from target and the selection of the tire around which the robot turns.

According to Fig. 9 the straight line trajectories of robots are computed by using Bresenham straight line algorithm as follows:

1. Compute Δx, Δy and the initial value of the decision parameter p_o using Eqs. (1) and (2).
2. Use Eqs. (3) and (4) to calculate the next positions of the two robots $((x^i_{h+1}, y^i_{h+1}), (x^j_{h+1}, y^j_{h+1}))$, starting from $h = 1$.
3. Calculate the distance between the two robots d^{ij}_{h+1} using Eq. (10).
4. If $d^{ij}_{h+1} > (r^i + r^j)$ then increase h by 1 and go to 2, else the distance $(r^i + r^j)$ is found.

Step 3. Calculation of the shortest distance and of the orientation to avoid collision: to estimate the shortest distance between robots to avoid the collision, we must choose a suitable direction for each robot. To select a suitable orientation to each robot we use the direction algorithm in Section 3. This algorithm prevents these robots from skewing far from their targets. Since rotations are performed around one of the tires, to estimate the shortest distance, we have to know which tire will be used. In Fig. 10 the red points represent the center of rotation for two robots and the red line represents the distance between these two centers. The shortest distance and the value of the orientation to avoid the collision are calculated simultaneously. Once this is done, the robots are let to move in a straight line trajectory up to the position corresponding to the shortest distance. The step is summarized as follows:

1. According to the direction algorithm in Section 3, choose a suitable orientation for the robots as shown in Fig. 10. This orientation is selected to avoid the collision and to prevent the robots from skewing far from their targets.
2. From Fig. 11 we find that the rotation is implemented around the right tire of each robot. The two robots are at the actual shortest distance when the distance between the right tires of robots equals the sum of the radiuses of robots and the distance L.
3. To calculate the distance R, at first, we must compute the coordinate axis to the right tires of both robots as in Fig. 11. For robot i:

$$\begin{aligned} x_{R}{}^{i}{}_{h} &= x^i_h + L/2 \cos\theta^i \\ y_{R}{}^{i}{}_{h} &= y^i_h + L/2 \sin\theta^i \end{aligned} \tag{16}$$

and, similarly, for robot j:

$$\begin{aligned} x_{R}{}^{j}{}_{h} &= x^j_h + L/2 \cos\theta^j \\ y_{R}{}^{j}{}_{h} &= y^j_h + L/2 \sin\theta^j. \end{aligned} \tag{17}$$

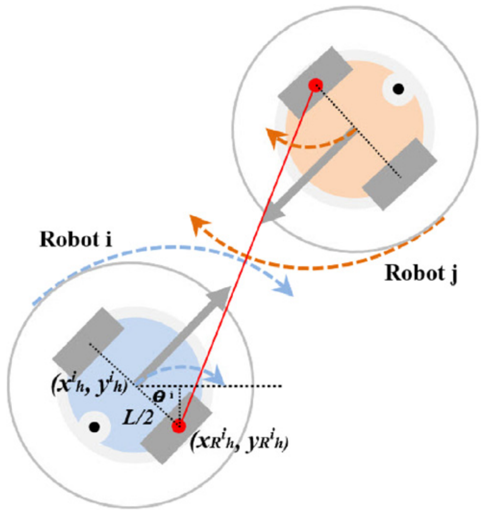


Fig. 11. Illustration of the computation of the distance between right tires of two robots.

4. Compute the distance R between the right tires of two robots using the following equation.

$$R = ((y_R{}^j{}_h - y_R{}^i{}_h)^2 + (x_R{}^j{}_h - x_R{}^i{}_h)^2)^{1/2}. \tag{18}$$

5. If $R < (r^i + r^j + L)$ then reduce h by 1 and go to 3.
6. Move the robots from their source to the calculated locations at shortest distance, that is, (x^i_h, y^i_h) and (x^j_h, y^j_h).

Step 4. Calculation of the circular arc trajectories: by using the Bresenham circle drawing algorithm, we compute the near future positions located on circular trajectories centered at the right tires of robots. This process is done by using the Bresenham circular arc algorithm to find the circular arc trajectories without collision. As shown in Fig. 12, the coordinate axis to the end points $((x^i_E, y^i_E)$ and $(x^j_E, y^j_E))$ lies on the straight line between the right tires of robots. The calculation of these trajectories is performed as follows:

1. Calculate the inclination angle of the straight line connecting the right tires centers using the following equation:

$$\phi = \tan^{-1}((y_R{}^j{}_h - y_R{}^i{}_h)/(x_R{}^j{}_h - x_R{}^i{}_h)). \tag{19}$$

2. According to Fig. 12, the coordinates of the endpoints are calculated as follows. For robot i:

$$\begin{aligned} x^i_E &= x_R{}^i{}_h + L/2\ \cos\phi \\ y^i_E &= y_R{}^i{}_h + L/2\ \sin\phi \end{aligned} \tag{20}$$

and for robot j:

$$\begin{aligned} x^j_E &= x_R{}^j{}_h - L/2\ \cos\phi \\ y^j_E &= y_R{}^j{}_h - L/2\ \sin\phi. \end{aligned} \tag{21}$$

3. Use the Bresenham circular arc algorithm in Section 2 to calculate the future locations of the robots on the circular paths, starting from the current sites $((x^i_h, y^i_h), (x^j_h, y^j_h))$ and ending at the final points $((x^i_E, y^i_E), (x^j_E, y^j_E))$. The centers of rotation for the robots are represented by the coordinates of the right tires $((x_R{}^i{}_h, y_R{}^i{}_h), (x_R{}^j{}_h, y_R{}^j{}_h))$ and the radiuses of rotation are $((r^i + L/2), (r^j + L/2))$.
4. Complete the movement of robots from current locations to the end points as shown in Fig. 13.

Step 5. Re-orientation of the robots to targets: in the last step robots perform a maneuver to avoid collision that eventually lead

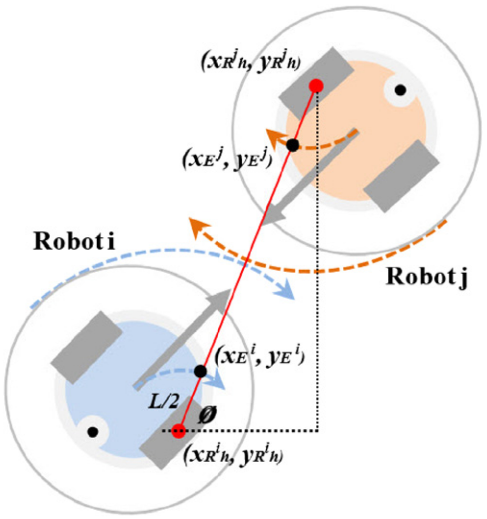


Fig. 12. Illustration of the computation of the circular arc trajectories.

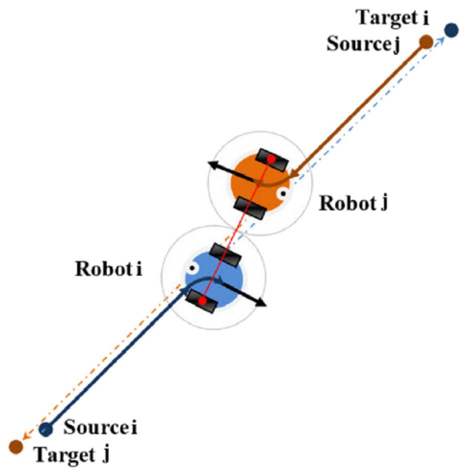


Fig. 13. The robots positions after the circular arc movement.

them to not point to the direction of the target. The purpose of this step is to re-orient them to their targets. The orientation is achieved by calculating circular arc trajectories around each other. The Bresenham circular arc algorithm in Section 2 is used to calculate the future positions for both robots on these trajectories. Each circular arc has a center at the current location of other robot and has a radius equal to the sum of the radiuses of both robots as shown in Fig. 14. Each robot trajectory starts from current location of the robot and ends at the tangent point between the computed circular arc and the straight line from the target (Fig. 14). To calculate the coordinates of tangent points we need the rules used with the intersection of two circles and reported in [24]. Step 5 is summarized as follows.

1. Use the following equations to calculate the tangent points between the circular arcs and the straight lines to the targets (Fig. 14). For robot i:

$$h^i_0 = ((y^i_p - y^j_h)^2 + (x^i_p - x^j_h)^2)^{1/2} \tag{22}$$

$$m^i = ((y^i_T - y^j_h)^2 + (x^i_T - x^j_h)^2)^{1/2} \tag{23}$$

with:

$$h^i_2 = ((m^i)^2 - (h^i_0)^2)^{1/2} \tag{24}$$

$$w^i = ((h^i_2)^2 - (h^i_0)^2 + (m^i)^2)/(2m^i) \tag{25}$$

$$h_1^i = ((h_2^i)^2 - (w^i)^2)^{1/2} \tag{26}$$

$$x_v^i = x_T^i - w^i(x_T^i - x_h^j)/m^i$$
$$y_v^i = y_T^i - w^i(y_T^i - y_h^j)/m^i \tag{27}$$

$$x_p^i = x_v^i + h_1^i(y_T^i - y_h^j)/m^i$$
$$y_p^i = y_v^i + h_1^i(x_T^i - x_h^j)/m_i \tag{28}$$

and for robot *j*:

$$h_0^j = ((y_p^j - y_h^i)^2 + (x_p^j - x_h^i)^2)^{1/2} \tag{29}$$

$$m_j = ((y_T^j - y^i)^2 + (x_T^j - x_h^i)^2)^{1/2} \tag{30}$$

$$h_2^j = ((m^j)^2 - (h_0^j)^2)^{1/2} \tag{31}$$

$$w^j = ((h_2^j)^2 - (h_0^j)^2 + (m^j)^2)/(2m^j) \tag{32}$$

$$h_1^j = ((h_2^j)^2 - (w^j)^2)^{1/2} \tag{33}$$

$$x_v^j = x_T^j - w^j(x_T^j - x_h^i)/m^j$$
$$y_v^j = y_T^j - w^j(y_T^j - y_h^i)/m^j \tag{34}$$

$$x_p^j = x_v^j + h_1^j(y_T^j - y_h^i)/m^j$$
$$y_p^j = y_v^j + h_1^j(x_T^j - x_h^i)/m^j. \tag{35}$$

2. Starting from the current sites $((x_h^i, y_h^i), (x_h^j, y_h^j))$ to the end points of the movement $((x_p^i, y_p^i), (x_p^j, y_p^j))$, use the Bresenham circular arc algorithm to calculate the future locations to the robots on the circular paths. The centers of rotation are represented by the starting coordinates of the robots $((x_h^i, y_h^i), (x_h^j, y_h^j))$ and the radiuses of rotation are $(r^i + r^j)$.
3. Complete the movement of robots from current locations to the end points as shown in Fig. 15.

Step 6. Calculation of the straight final movement to the targets: once performed the maneuver to avoid the other robots and having re-oriented the vehicle, a straight trajectory towards the target can be planned (Fig. 16). To do this, the Bresenham straight line algorithm is used. Clearly, if other collisions are predicted, the previous steps are repeated. Each robot trajectory starts from the current location and ends at the target. Step 6 consists of the following items.

1. Starting from the current sites $((x_h^i, y_h^i), (x_h^j, y_h^j))$ until the target points $((x_T^i, y_T^i), (x_T^j, y_T^j))$, use the Bresenham straight line algorithm to calculate the future locations to the robots on the straight paths.
2. If the future locations of robots may lead to collision then repeat steps 3–5.
3. Complete the movement of robots from current locations to the tangent points as shown in Fig. 17.

5. Simulation results

The new multi-robot navigation approach is implemented and tested on three challenging scenarios, which are usually considered as a benchmark for multi-robot navigation [19]:

- Chicken scenario: two robots are initialized at opposite ends of the workspace [13]. They must pass each other to exchange positions. Each robot has the same preferred speed of 80 pixel/s.
- Four corners scenario: a robot is placed at each corner of a rectangular environment, and its goal is to navigate to the opposite corner of the environment on the diagonal [25]. The robots will meet and have to negotiate around each other in the middle. Each robot has the same preferred speed of 80 pixel/s.
- Circle scenario: 12 robots are distributed equally in a circle, and each one has a target on the other side of the circle [12]. All robots have the same speed (80 pixel/s).

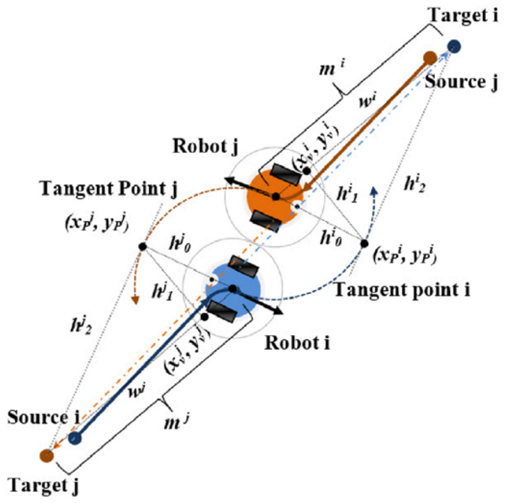


Fig. 14. Illustration of step 5 of SDA.

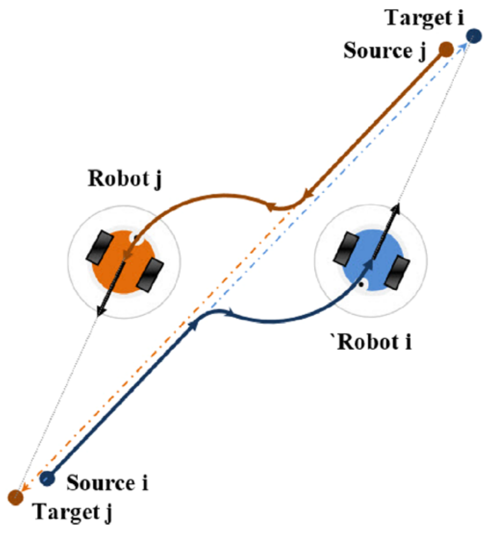


Fig. 15. Illustration of the trajectory obtained at step 5.

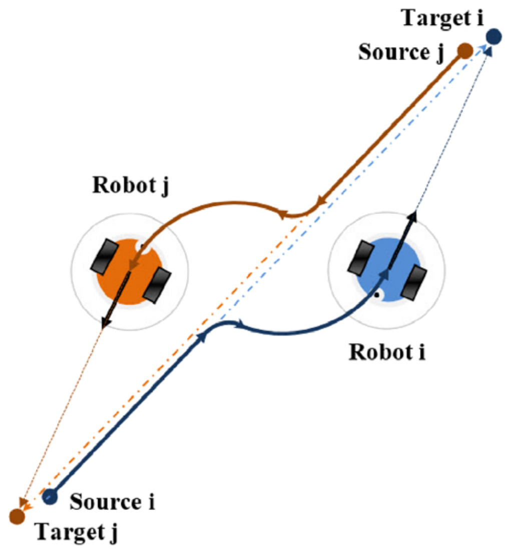


Fig. 16. Illustration of step 6 of the SDA.

In addition, simulations in the last scenarios have been repeated for different values of the radius and of the number of robots.

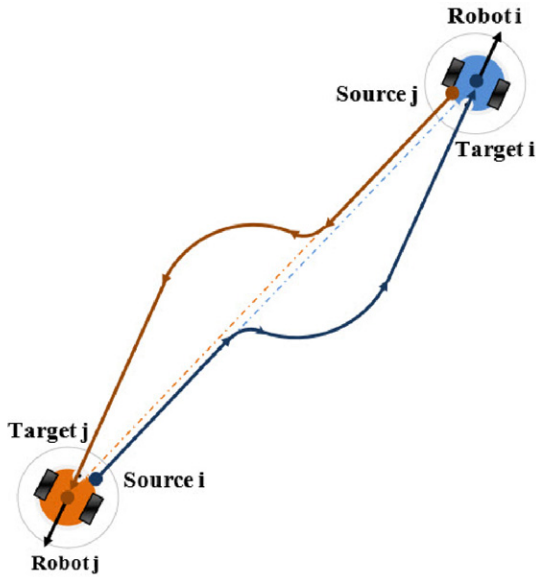


Fig. 17. Illustration of the whole trajectory obtained by applying SDA.

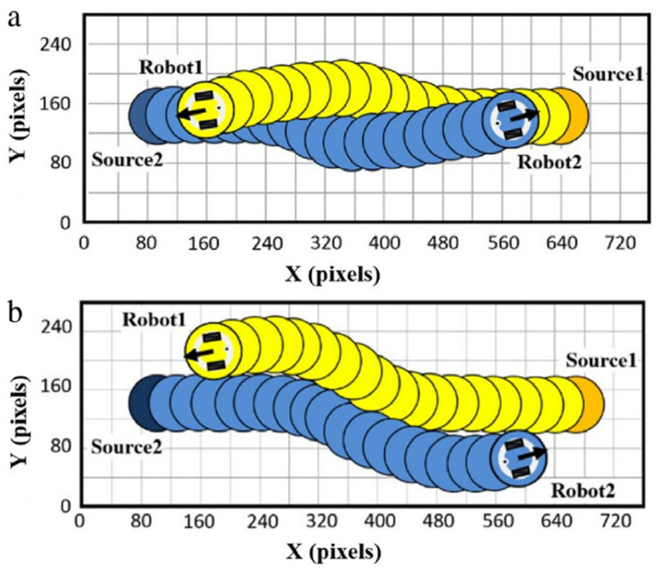


Fig. 18. The robots' motion in the chicken scenario (speed = 80 pixel/s): (a) ROA; (b) SDA.

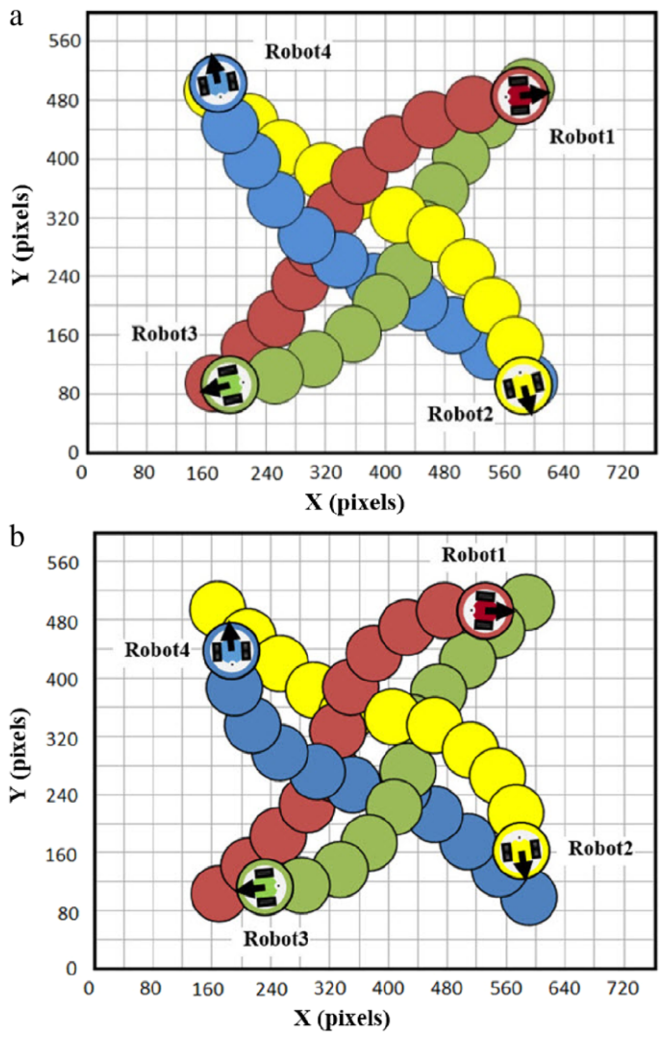


Fig. 19. Trajectories of the robots in the four corner scenario (speed = 80 pixel/s): (a) ROA; (b) SDA.

For each case the length of the trajectory obtained is measured as indicator of the algorithm performance.

Robot trajectories are shown in Fig. 18 for the chicken scenario. Later positions are drawn on top of earlier. Fig. 18(a) shows the trajectory obtained by using ROA [19] while Fig. 18(b) shows the result of SDA. By comparing the two algorithms in scenario one, it appears that SDA is collision free and reduces the distance to avoid the collision with other robots, but on the other hand it increases the angle of turn which may orient the robot far from his goal. On the other hand, the ROA improves the angle of orientation and makes the robot not far from the direction of his goal, but it requires an increased range to take the decision for avoidance.

The trajectories of the robots in the four corners scenario are shown in Fig. 19. Fig. 19(a) shows the result of ROA [19], and Fig. 19(b), that of SDA. The two algorithms generate trajectories, which, in both cases, are smooth, collision free, and without oscillations. However, the SDA paths contain only straight and circular arc trajectories. This makes it easier to implement.

The third scenario represents the circle formation of a team of robots. Fig. 20(a) shows the trajectories obtained using the ROA [19], while Fig. 20(b) shows the trajectories obtained by using the SDA. There are no collisions among robots and both algorithms produce smooth and regular movements. The SDA gives a shorter path to target and it is implemented with simple computation since it only uses Bresenham algorithms.

For a more detailed comparison, simulations in the last scenario have been repeated for a different number of robots, ranging from 2 to 20 at steps of 2. For each value of the number of robots, 3 different values of the robot radius have been considered. For each radius, the length of the trajectory between sources and targets for both algorithms have been then computed and reported in separate figures (Figs. 21–23). For all the figures, we have found that the introduced algorithm produces a shorter trajectory from source to target for any value of the number of robots and any value of robot radius.

6. Conclusion

In this paper, a new approach, called the shortest distance algorithm (SDA), is introduced for smooth and collision free navigation of multiple robots and compared with the reciprocal orientation algorithm (ROA). The kinematics of the robot is incorporated, and implemented on mobile robots. From the simulation results we found that all the algorithms discussed in this paper are collision free, prevent oscillatory motions and straight forward. The simulation results show that the ROA

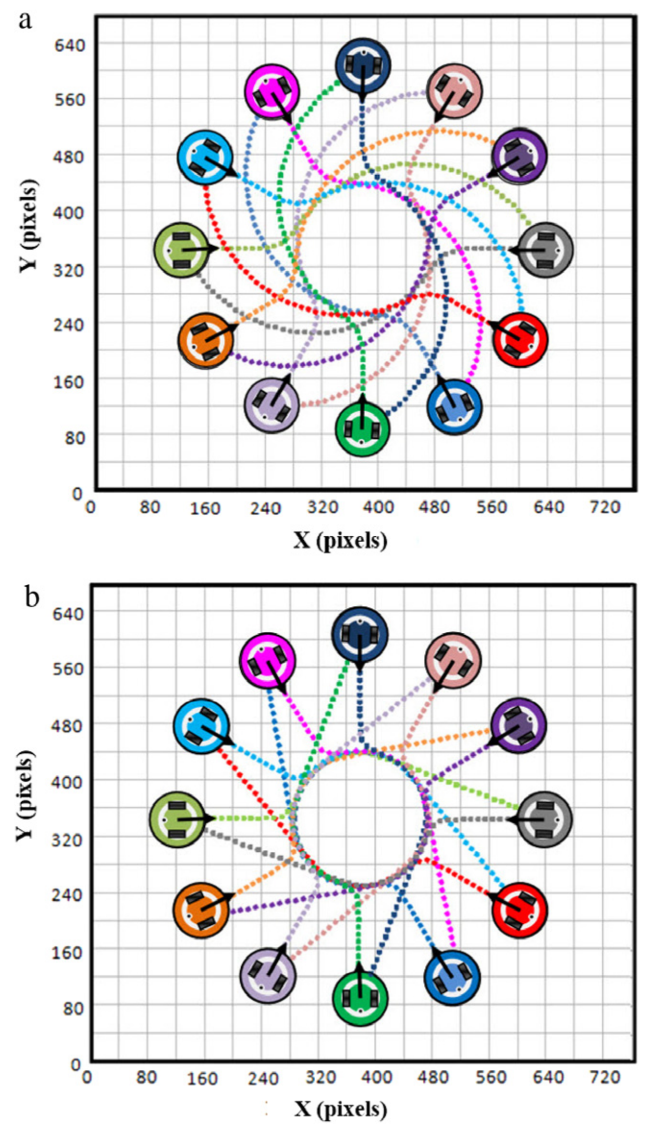
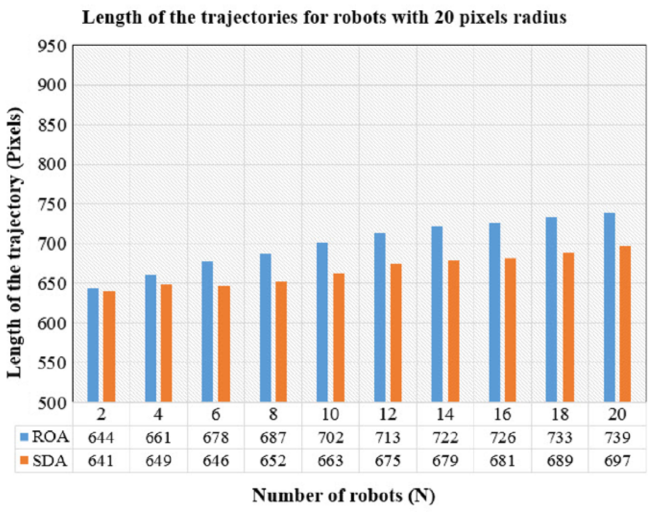
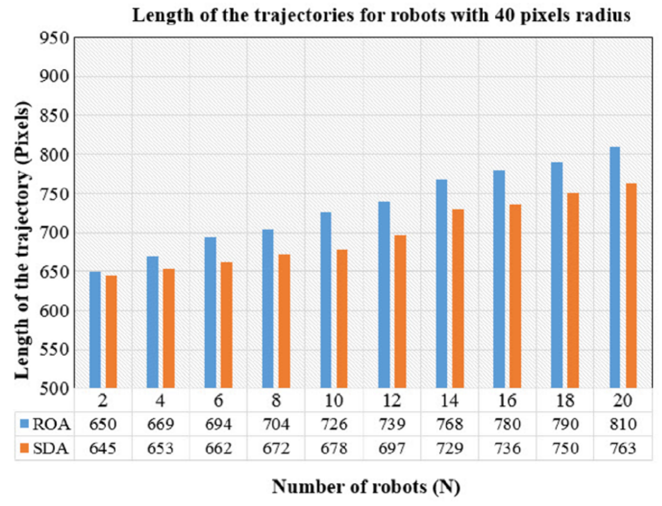


Fig. 20. The multi-robots formation with the circle scenario (speed $= 80$ pixel/s): (a) ROA; (b) SDA.



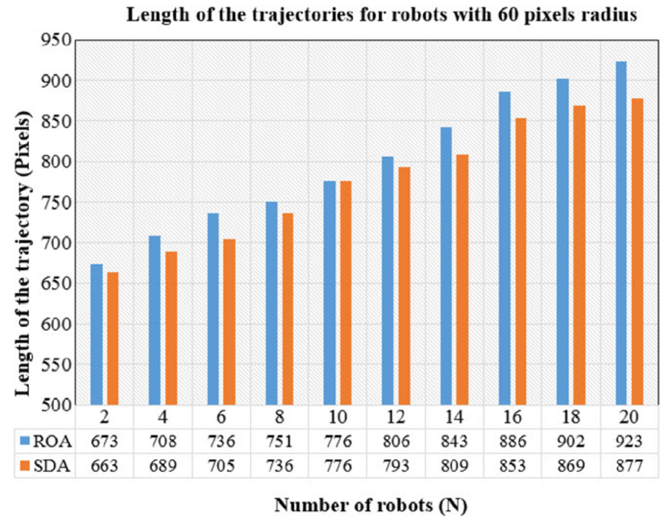
	2	4	6	8	10	12	14	16	18	20
■ ROA	644	661	678	687	702	713	722	726	733	739
■ SDA	641	649	646	652	663	675	679	681	689	697

Fig. 21. Comparison of ROA and SDA: length of the trajectory between source and target for robots with a radius of 20 pixels.



	2	4	6	8	10	12	14	16	18	20
■ ROA	650	669	694	704	726	739	768	780	790	810
■ SDA	645	653	662	672	678	697	729	736	750	763

Fig. 22. Comparison of ROA and SDA: length of the trajectory between source and target for robots with a radius of 40 pixels.



	2	4	6	8	10	12	14	16	18	20
■ ROA	673	708	736	751	776	806	843	886	902	923
■ SDA	663	689	705	736	776	793	809	853	869	877

Fig. 23. Comparison of ROA and SDA: length of the trajectory between source and target for robots with a radius of 60 pixels.

improves the angle of orientation and makes the robot not far from the direction of his goal, but the new algorithm provides a shorter path for all the scenarios investigated in this paper. These refer to three benchmark cases and simulations with a different number of robots and several values of the radius. Our algorithm makes use of the Bresenham straight line and circular arc algorithms that reduce the amount of computation of the robot trajectories.

The current implementation of the SDA is mainly for robots moving in a two-dimensional plane, but can be easily extended to autonomous vehicles moving in a three-dimensional space.

References

[1] J. Canny, J. Reif, New lower bound techniques for robot motion planning problems, in: IEEE Symposium on Foundation of Computer Science, 1987, pp. 49–60.

[2] J. Barraquand, B. Langlois, J.-C. Latombe, Numerical potential field techniques for robot path planning, IEEE Trans. Syst. Man Cybern. 22 (2) (1992) 224–241.

[3] I. Krjanc, G. Klancar, Optimal cooperative collision avoidance between multiple robots based on Bernstein–Bézier curves, Robot. Auton. Syst. (2010) 1–9. (Elsevier).

[4] S. Leroy, J.P. Laumond, T. Simeon, Multiple path coordination for mobile robots: A geometric algorithm, in: International Joint Conference on Artificial Intelligence, IJCAI, 1999.

[5] M. Bennewitz, W. Burgard, Coordinating the motions of multiple mobile robots using a probabilistic model, in: 8th International Symposium on Intelligent Robotic Systems, SIRS, 2000.
[6] L.E. Dubins, On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, Amer. J. Math. 79 (1957) 497–516.
[7] J.A. Reeds, L.A. Shepp, Optimal paths for a car that goes both forwards and backwards, Pacific J. Math. 145 (2) (1990) 367–393.
[8] S.M. LaValle, Planning Algorithms, Cambridge Univ. Pr., 2006.
[9] J.-C. Latombe, A fast path planner for a car-like indoor mobile robot, in: Proc. AAAI Nat. Conf. Artif. Intell., 1991, pp. 659–665.
[10] N. Michael, J. Fink, V. Kumar, Experimental testbed for large multi-robot teams, IEEE Robot. Autom. Mag. 15 (1) (2008) 53–61.
[11] J. van den Berg, J. Snape, J. Guy, S.J. Guy, D. Manocha, Reciprocal collision avoidance with acceleration-velocity obstacles, in: Proc. IEEE Int. Conf. Robot. Autom., 2011, pp. 3475–3482.
[12] J. van den Berg, M. Lin, D. Manocha, Reciprocal velocity obstacles for real-time multi-agent navigation, in: Proc. IEEE Int. Conf. Robot. Autom., 2008, pp. 1928–1935.
[13] J. Snape, J. van den Berg, J. Guy, Stephen J. Guy, D. Manocha, Smooth coordination and navigation for multiple differential-drive robots, in: Proc. IEEE RSJ Int. Conf. Intell. Robot. Syst., 2010, pp. 1–13.
[14] P. Fiorini, Z. Shiller, Motion planning in dynamic environments using velocity obstacles, Int. J. Robot. Res. (1998) 760–772.
[15] J. van den Berg, M. Lin, D. Manocha, The hybrid reciprocal velocity obstacle, IEEE Trans. Robot. (2011) 696–706.
[16] S.J. Guy, J. Chhugani, C. Kim, N. Satish, P. Dubey, M. Lin, D. Manocha, Highly parallel collision avoidance for multi-agent simulation, in: Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2009, pp. 177–187.
[17] C. Fulgenzi, A. Spalanzani, C. Laugier, Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid, in: Proc. IEEE Int. Conf. Robot. Autom., 2007, pp. 1610–1616.
[18] J.H. Liang, C.H. Lee, Efficient collision-free path-planning of multiple mobile robots system using efficient artificial bee colony algorithm, Adv. Eng. Softw. 79 (2015) 47–56.
[19] A.T. Rashid, A.A. Ali, M. Frasca, L. Fortuna, Multi-robot collision-free navigation based on reciprocal orientation, Robot. Auton. Syst. 60 (10) (2012) 1221–1230.
[20] J.L. Jones, N.E. Mack, D.M. Nugent, P.E. Sandin, Autonomous floor-cleaning robot, US Patent 6,883,201, 2005.
[21] E. Prassler, J. Scholz, P. Fiorini, A robotic wheelchair for crowded public environments, IEEE Robot. Autom. Mag. 8 (1) (2001) 38–45.
[22] J. Borenstein, H.R. Everett, L. Feng, Where am I, systems and methods for mobile robot positioning, University of Michigan, 1996.
[23] Jack E. Bresenham, Ambiguities in incremental line rastering, IEEE Comput. Graph. Appl. 7 (5) (1987) 31–43.
[24] P.A. Egerton, W.S. Hall, Computer Graphics, Prentice Hall, Europe, 1998.
[25] J. Snape, J. van den Berg, S.J. Guy, D. Manocha, Independent avigation of multiple mobile robots with hybrid reciprocal velocity obstacles, in: The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2009, pp. 5917–5922.

Abduladhem A. Ali Received M.Sc. and Ph.D. Degrees from Department of Electrical Engineering University of Basrah, Iraq, in 1983 and 1996. Worked as Assistant Lecturer, Lecturer, and Assistant professor at the same Department at 1984, 1987 and 1981 respectively. Then 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 100 published papers, one patent, supervised many M.Sc. and Ph.D. Dissertations. The Editor chair for the Iraqi Journal for Electrical and Electronic Engineering and 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 Senior member of IEEE

Abdulmuttalib T. 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. and Ph.D. Degree from the same University at 1992 and 2012 respectively. Worked as Assistant Lecturer, Lecturer at the Department of Electrical Engineering University of Omer Al Mukhtar, Libya, in 1997–2007. Then at the Department of Electrical Engineering, University of Basrah, Iraq, in 2007 up to now. His field of Interest is Robotics and Industrial control. The research interests were in motion planning and control of multi mobile robots.

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, at 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 co-author of two international patents. He is IEEE Senior.

Luigi Fortuna (M'90–SM'99–F'00) was born in Siracusa, 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, micronanosensor 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.