On-line Path Planning for Swarm of Mobile Robots Based on Particle Swarm Optimization

C. Karakuzu[1*] and R. Babuška[2]

Abstract— Generally, most of the related studies in the literature report only simulation results on path planning problems. In this paper, a PSO-based path planning method is proposed and applied in real-time experiments to determine an appropriate on-line path for a swarm of mobile robots in a 2D environment. This paper presents a novel on-line path planning strategy for a swarm of mobile robots. The method is based on particle swarm optimization (PSO). Each robot is considered an agent, and an independent particle swarm is used to find an appropriate path in real time. At each sampling instant, the PSO algorithm optimizes the target position to be for the robot at the next time step. The PSO cost function accounts for the distance of the robot from the goal, the positions of the other robots, and obstacles in the environment. The proposed on-line path planning strategy has been implemented and evaluated on an experimental setup. Experimental results for three different path planning scenarios under varying conditions are included in this paper. The most significant contributions of this study are the results of the proposed method in real-time experiments.

Index Terms— path planning, mobile robot, particle swarm optimization, robot control, local search.

I INTRODUCTION

Path planning is the task of determining an optimal path from an initial position to a target position while avoiding collisions between robots and obstacles in the environment. This task may be achieved either through off-line path planning or through on-line, real-time path planning. The advantage of real-time path planning is that the path can be replanned if the situation changes, such as in an environment with dynamic obstacles. In this paper, we propose a real-time path planning method based on particle swarm optimization.

Computationally, path planning is an NP-complete problem. Therefore, the computational time required to solve a problem increases dramatically when the scale of the problem increases [1]. This is the main reason swarm intelligence techniques have recently received considerable attention in mobile robot path planning. These techniques are inspired by the social behavior of animals such as birds or bees. Typical examples of swarm intelligence methods are particle swarm optimization (PSO), inspired by flocking bird swarm behavior, and an artificial bee colony (ABC), inspired by the foraging behavior of honey bees. The use of swarm intelligence in path planning can be found in the literature such as [2], [3], and [4]. In this paper, a PSO algorithm is used as it is computationally more effective than other swarm intelligence algorithms, and it finds a good solution quickly. Below, several remarkable studies are summarized that coincide with the scope of this study.

Doctor et al.[5] studied single-target and multiple-target search using PSO in a simulation environment. They concluded that PSO is quite reliable in target-searching applications. Lei et al. [6] and Chen and Li [7] used PSO and its stochastic variant S-PSO to solve a path planning problem for mobile robots in a simulated environment with obstacles. The simulation results of Chen and Li (2006) demonstrate that the proposed S-PSO algorithm is effective thanks to its exploration ability and low computational costs with small swarm size. Vahdat et al. [8] compared differential evolution (DE) and PSO methods with a standard Monte Carlo localization (MCL) algorithm for a mobile robot localization problem. Rigatos [9] used distributed gradient and particle swarm optimization for multi-robot motion planning. The performance of both approaches has been evaluated through simulation tests in a 2D environment with polyhedral obstacles. Even though both performance methods were satisfactory, the distributed gradient had advantages over particle swarm optimization. A drawback of that method, however, is that it needs complex gradient computations. In [10], two fuzzy controllers, called distance and angle controllers, were designed to control a mobile robot in a simulated 2D environment without obstacles. The PSO was used to determine optimal parameters of the fuzzy controller's membership functions so that the robot could effectively move to the desired position.

All the above papers report only simulation results. In this

[1]*Corresponding author mail: cihan.karakuzu@bilecik.edu.tr (https://orcid.org/0000-0003-0569-098X)
Computer Engineering, Bilecik Seyh Edebali University, Bilecik-Turkey
[2] Second author mail: r.babuska@tudelft.nl (https://orcid.org/0000-0001-9578-8598)
Department of Cognitive Robotics, Delft University of Technology, Delft-Netherlands

paper, a PSO- based path planning method is proposed and applied in real-world experiments to determine an appropriate on-line path for mobile robots moving as a swarm in a 2D environment with obstacles. This method requires information only about the robot's attitude (position and orientation) and registers on-line path planning in real time. Each mobile robot uses its own independent particle swarm optimizer to search for the most appropriate position for the next sampling time in a local search area formed around the current position. In this way, a path for the robot is determined step by step. Experimental results are given for three different path planning scenarios. The main contribution of our work is the application of this PSO-based method to an experimental setup in real-time.

The rest of the paper is organized as follows: Section II introduces the algorithm for path planning with obstacle avoidance; Section III presents the experimental setup used in this study; Section IV summarizes the experimental results obtained with the proposed path planning strategy. In the last section, conclusions are drawn.

II On-line path planning based on PSO

The aim is to determine an appropriate path for each robot in the swarm from some initial position to a specified goal position without colliding with other robots and obstacles.

A PSO algorithm

The stochastic PSO algorithm called S-PSO [7] is used in this study. The algorithm is defined by the following equations:

$$v_i(n+1) = \psi(n)[v_i(n) + c_1 r_1 (p_i^{pb}(n) - p_i(n)) + c_2 r_2 (p^{gb}(n) - p_i(n)) + \lambda R(n)] \quad (1)$$

$$p_i(n+1) = \alpha p_i(n) + v_i(n+1) + \frac{1-\alpha}{c_1 r_1 + c_2 r_2}(c_1 r_1 p_i^{pb}(n) + c_2 r_2 p^{gb}(n)) \quad (2)$$

The symbols have the following meaning: p_i is the position and v_i the velocity of the ith particle; c_1, c_2 are learning constants, and α is the learning rate. Furthermore, r_1 and r_2 are uniformly distributed random numbers, λ is a small constant, and R is a vector including zero-mean, unit-variance normally distributed random numbers. The superscripts pb and gb refer to the personal best and global best particle, respectively. The personal best particle (p^{pb}) corresponds to the particle's own best-known position found so far. The global best particle (p^{gb}) corresponds to the best one found so far among all the particles. After the last iteration, denoted by Gmax, the global best particle is assigned as the final solution. The inertia term $\psi(n)$ decays to zero as the number of iterations n grows [7]:

$$\psi(n) = \frac{2.5}{n+1} \quad (3)$$

Bratton and Kennedy [11] showed that this inertia term improves the PSO algorithm's performance.

The above algorithm was implemented in Matlab with the following parameter values determined through a series of simulation experiments: c1 = c2 = 2.5, α = 0.5, $\lambda=10^{-2}$. Particle positions are restricted within the physical limits of the area where the robots move, and their velocities are restricted to $[-r/5 \quad r/5]$ with the local search area radius r explained in the following section.

B Path planning strategy

The PSO algorithm explained in the previous section was applied to determine in real time the most appropriate position of a mobile robot in a 2D environment. Each mobile robot uses its own PSO swarm with N particles associated with the candidate positions of the robot. Figure 1 illustrates the strategy used to determine the path of a robot. The target position for the next sampling instant is searched in the neighborhood with radius r of the robot's current position pa. The area enclosed by the dashed line circle is called the local search area (LSA).

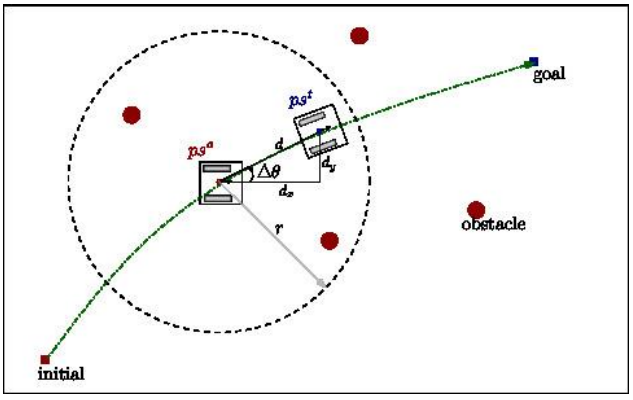

Figure 1 Local search area used for the path planning

The radius r of LSA is defined by Eq (4), where Δt is the sampling period of the system and u_{max} is the maximum velocity control signal that can be applied to the robot (0.25 m/s).

$$r = u_{max}\Delta t \quad (4)$$

In Fig. 1, p^t denotes the target position found by the PSO after a predefined maximum number of generations Gmax. This position is set as the target for the robot for the next step. The distance between the actual position p^a and the target position p^t is denoted by d, and $\Delta\theta$ is the change of the robot's orientation during Δt. At each sampling instant, the particle positions are initialized randomly with a uniform distribution within LSA, and their velocities are restricted to $[-r/5 \quad r/5]$.

To determine the local best and the global best particle, the cost value Ji of each particle i is calculated by Eq (5), where $\gamma_1, \gamma_2, \gamma_3 \in R^+$ are weight factors.

$$J_i = \gamma_1 \|p_i - p^{goal}\| + \gamma_2 F_i^{potf} + \gamma_3 F_i^{obs} \quad (5)$$

The first term is the Euclidean distance between the current position of the *i*th particle and the eventual goal position of the robot. The second term takes into account the mutual distances between the robots by using potential functions [12] given in Eqs (6) and (7). The last term is defined in Eqs (8) and (9) to avoid obstacles [7].

$$F_i^{potf} = \sum_{j=1}^{n_r} \sum_{k=1}^{n_r} f_{jk}^{pot} \tag{6}$$

$$f_{jk}^{pot} = \frac{a}{2}\|p_i^j - p_i^k\|^2 + \frac{bc}{2} e^{-\frac{\|p_i^j - p_i^k\|^2}{D_{jk}}} \tag{7}$$

$$F_i^{obs} = \sum_{m=1}^{h+n_r-1} f_m^{obs} \tag{8}$$

$$f_m^{obs} = \begin{cases} \frac{1}{\|p_i - Obs_m\|} - \frac{1}{\delta^{obst}} & \text{if } \|p_i - Obs_m\| \le \delta^{obst} \\ 0 & \text{if } \|p_i - Obs_m\| > \delta^{obst} \end{cases} \tag{9}$$

In the above equations, n_r and h respectively represent the number of mobile robots and the number of obstacles; a, b and c are constant parameters of the potential functions, and D is the safe distance defined to avoid collision between two mobile robots. *Obs* and δ^{obst} respectively represent obstacle position and the safety margin defined to ensure that the robot passes without hitting the obstacle. The last term in Eq.(5) is active if the distance between the robot and an obstacle is less than a safety margin δ^{obst}. Each robot found within the safety margin is considered an obstacle and therefore it may be a dynamic obstacle for the other robots. This term is important in that it emphasizes that the method also works for dynamic obstacles.

Now, we have the cost values for each particle in the swarm. The particle with the minimum cost is assigned as the global best particle $p^{gb,}$ and each particle is assigned as its own personal best for the first generation. Then for the next generation, the swarm is updated according to the personal best and global best particles. Finally, the cost values of the new particles are calculated again using equation (5). If a particle has a lower cost than the global best particle of the previous generation, this particle is assigned as the global best particle. The personal best particle is the particle's own best-known position found so far among the current and previous generations for each particle. The swarm is updated again according to the new personal best and global best particles. For each generation, the personal best value of each particle and the global best found so far are updated using current and prior cost values. The algorithm is conducted in the same way until the generation number n reaches the maximum generation number (*Gmax*). At the end of the *Gmax* generation, the global best particle of each PSO swarm is assigned as the new target position for the related robot. Then a motion control unit calculates the necessary control signal and sends it to the robot to reach the target position.

Fig. 2 shows the positions of particles separately according to the generation numbers for the mobile robot in a sampling time interval. As seen in the figure, initial particles are positioned randomly around the robot's actual position of the first generation. As the generation number increases, particles begin to change their positions in a particular direction. The last generation particles have their positions near the point marked as the global best.

C Robot motion control

After the target positions of each robot for the next step are determined by the relevant PSO swarm, as explained in the previous subsection, the main problem is to produce a suitable control signal to implement this strategy in real time. The control structure given in Fig. 3 has been designed for this purpose. In this structure, the classical PD control method is preferred thanks to its simplicity.

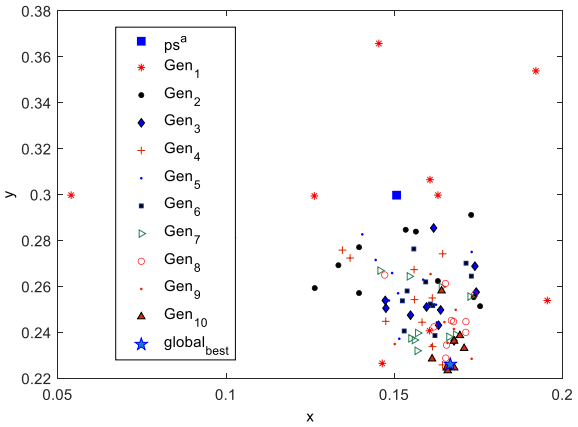

Figure 2 Positions of particles produced for a mobile robot in a sampling time interval

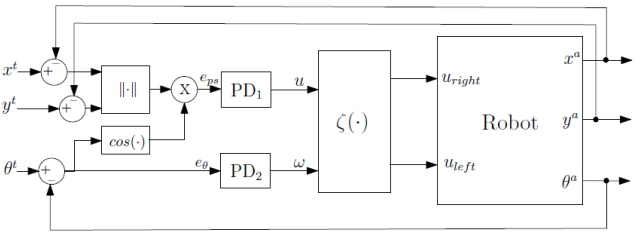

Figure 3 Motion control structure for a mobile robot.

The integral term is not used because steady-state errors can be tolerated by the nature of the method. The structure includes two PD controllers that are used separately for each robot in a robot swarm. While PD_1 produces a forward or backward movement control signal (u), at the same time, PD_2 produces an angular velocity (ω) to rotate the robot to the needed direction. Then these signals are sent to $\zeta(\cdot)$ block. This block calculates the velocities in m/s to move the robots toward the target position (p^t). The calculation realized by the motion control unit is defined as follows: the robot's target position (pt) is determined by its own PSO as:

$$p^t = [x^t\ y^t] \quad (10)$$

where x^t and y^t denote the apsis and ordinate of the target position. The target orientation θ^t of a robot is calculated by:

$$\theta^t = tan^{-1}(\frac{y^t - y^a}{x^t - x^a}) \quad (11)$$

where x^a and y^a denote the coordinates of the robot's actual position. The motion control from an actual position to a target position can be divided into two sub-tasks: position and orientation control. The first one produces the main control signal (u) to move a robot to a target position (x^t,y^t). PD_1 accomplishes this task according to the positional error. As the result of an analysis given in [13], the positional error (e_p) is formulated as in (12).

$$e_p = \sqrt{(x^t - x^a)^2 + (y^t - y^a)^2}\, cos(\theta^t - \theta^a) \quad (12)$$

Based on the positional error, PD_1 produces the primary control signal (u) according to (13). u is the robot's velocity in the driving direction, and it is the average of the velocities on the left (u_{left}) and on the right (u_{right}) wheel:

$$u = K_{p_p} e_p + K_{d_p} \frac{\Delta e_p}{\Delta t} \quad (13)$$

with K_p and K_d the proportional and derivative gains for PD control. Subscript p denotes position-related variables. Orientation control is responsible for supplying the proper orientation while the robot moves to the target position. To this end, first, the orientation error is found using (14). Then based on this error, PD_2 produces an angular velocity (ω), which is the derivative of θ and is equal to the difference in the wheel velocities divided by the wheelbase (W_b), according to (15).

$$e_\theta = \theta^t - \theta^a \quad (14)$$

$$\omega = K_{p_\theta} e_\theta + K_{d_\theta} \frac{\Delta e_\theta}{\Delta t} \quad (15)$$

Here Δ denotes the difference in the relevant variables between two sampling periods, subscript θ denotes the variables pertinent to orientation. Angular velocity (ω) is used to obtain the necessary rotation of the robot while it moves to the target position. The rotation is achieved by using different velocities for the robot's two wheels. The velocities of the wheels are calculated in the $\zeta(\cdot)$ block according to (16).

$$\zeta(u, \omega) = \begin{cases} u_{right} = u + \frac{\omega W_b}{2} & \text{for the right wheel} \\ u_{left} = u - \frac{\omega W_b}{2} & \text{for the left wheel} \end{cases} \quad (16)$$

Here W_b is the distance between the two wheels called the wheel base. The control equation described above is executed for each robot at each sampling time interval.

III EXPERIMENTAL SETUP

The experimental setup [14], implemented at the Delft Center for Systems and Control (DCSC), consists of a platform where five mini robots were driven around. The robots were connected wirelessly to a central computer close to the platform and were controlled by this computer. A camera was positioned above the platform and connected to the computer via a USB cable. The camera acquired images used to calculate the positions of the robots via image recognition. Fig. 4 shows a block diagram and a photo of the setup.

The experimental platform consists of the following parts: a platform, five mini-robots, wireless communication between the robots and the central computer, a camera, image recognition software, and the central computer. The specifications for these parts can be summarized as follows:

Platform: Its size was chosen as 1.5m x 1m. It has borders with a height of 75 mm to ensure that the robots do not accidentally fall off the platform.

Robots: The setup has five Miabot Pro robots produced by Merlin Robotics in the UK. The Miabot Pro is a small cube with a size of 75 × 75 × 75 mm and a weight of 0.55kg. It has two wheels driven by two electric motors and a battery with a maximum charge cycle of 5 hours (about 1 hour when operated continuously). Its processor's specifications are Atmel ATMega64, a speed of 14.5 MIPS (RISC), the program memory of 64Kb flash, a data memory of 4 Kb SRAM, and nonvolatile storage of 2 Kb EEPROM. It has an 8 user I/O or 10-bit A/D expansion port. The robot has no built-in default sensor. The maximum speed of the robots is 3.5 m/s. The robots are wirelessly controlled by the central computer. They have a built-in Bluetooth connection with a maximum data rate of 115.2 kbps Bluetooth server used with the central computer, as shown in Fig. 4. One Bluetooth server can communicate with up to seven robots, and it is connected to the computer via a regular LAN network. When a connection is created between a robot and a PC, a virtual COM port is opened on the PC to receive and send commands. The communication is performed via a Matlab program which does not require the installation of additional software.

Camera: A camera, Lumenera Lu075c, is mounted 1.7 m above the platform. The camera has 640 × 480 pixels resolution, 60 fps frame rate, and a CCD sensor. It was chosen for its high image sharpness, color depth, and frame rate. This camera only needs a USB cable connection to enable the data transfer and power supply. The camera images are used to observe the position of the robots via image recognition software which runs on the central computer.

Computer: The central computer determines the robots' IDs and positions using image recognition, communicates with the robots, and runs customized Matlab programs to control the robots. For image recognition, codes from the robot soccer team of the University of Nottingham and the University of Twente were reused [14]. Their image recognition approach is described in [15] and [16]. This software contains functions for initializing and setting up the camera, starting or stopping the capturing of images,

calculating the robot's positions from the images, and stopping the camera. The software was compiled as a MEX file that provides the robot's ID, position (x, y) in meters, and orientation (θ) in radians.

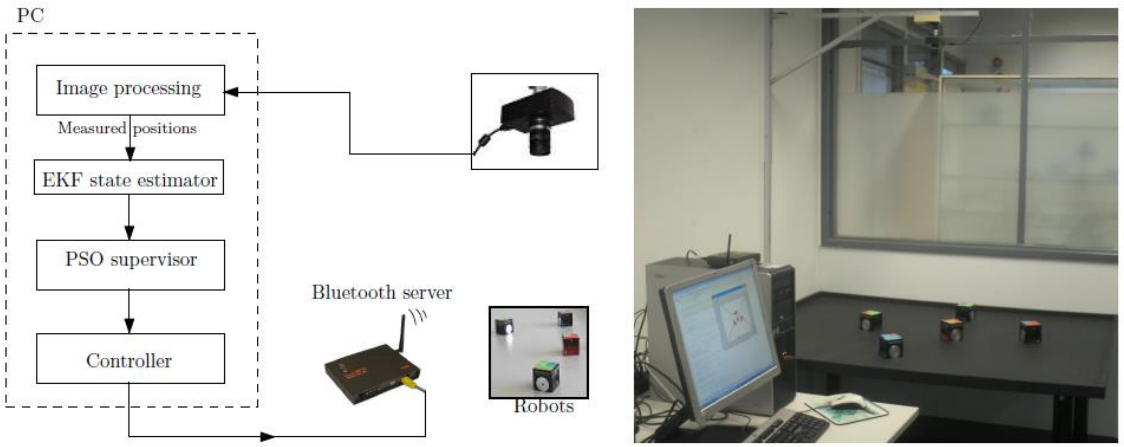

Figure 4 Block diagram and photo of the setup.

IV Experimental Results

The path planning method explained in the previous sections was implemented in a Matlab program whose flow chart is shown in Fig. 5. Three groups of experiments were performed on the setup: Aggregation formation, moving to the goal, and following the moving goal. For these experiments, the swarm size (N) and the maximum generation number ($Gmax$) were set to 10 considering fast on-line computation time. Fig. 2 proves that these numbers are sufficient for the experiments. The parameters of the potential function in (7) are $a=0.1$, $b = 20$ and $c = 0.01$. D_{jk} parameters are listed inTable 1.

TABLE 1
Desired distance parameter values for two robots

Desired distance between two robots	D_{jk}
10 cm	5.10^{-4}
15 cm	3.10^{-3}
20 cm	5.10^{-3}
25 cm	1.10^{-2}
30 cm	2.10^{-2}
35 cm	3.10^{-2}
40 cm	4.10^{-2}
45 cm	5.10^{-2}

The a, b, c parameter values and D_{jk} in Table 1 were determined by a trial-and-error curve analysis of the function in (7) in a simulation environment. The function in (7) with parameters of $a=0.1$, $b = 20$, and $c = 0.01$ has a minimal value at the relevant distance between two robots for the respectiveD_{jk} value given in Table 1. Fig. 6 shows the curves for some D_{jk} values against the distance between two robots to provide a general idea about their function in this work. D_{jk} is the value that must be chosen before the experiment according to the desired distance between jth and kth robots. These values are assigned as a square matrix determined by the number of robots (n_r) in the program. For instance, if we use two robots on the platform and the desired distance between them is 15 cm, then this matrix would be

$$D = \begin{bmatrix} 0 & 0.003 \\ 0.003 & 0 \end{bmatrix}$$

Three group experiments were performed on the setup according to the procedure outlined below:

- Calibrate the image processing software via an interface program
- Select the experiment type: *Aggregation formation*, *Moving to goal*, *Following the moving goal*
- Select the weight factors of the main cost function in (5)
  - *if fixed or moving goals are used for the robots, then $\gamma_1 \geq 1$; otherwise, $\gamma_1 = 0$.*
  - *if the desired distance between the robots is used, then $\gamma_2 = 1$; otherwise, $\gamma_2 = 0$.*
  - *if obstacles on the platform are defined or if more than one robot is used, then $\gamma_3 = 1$; otherwise, $\gamma_3 = 0$.*
- Assign the constants or parameters for the program: *Swarm size (N), maximum generation number (Gmax), wheelbase (Wb), robot IDs to be used, physical size of the platform, server IP, obstacles (Obs), desired distance between the robots (D = [Djk]), potential field function parameters (a, b, c),*

fixed or moving goal(s), etc.

- Run the program to perform the experiment
- Obtain the results and record data

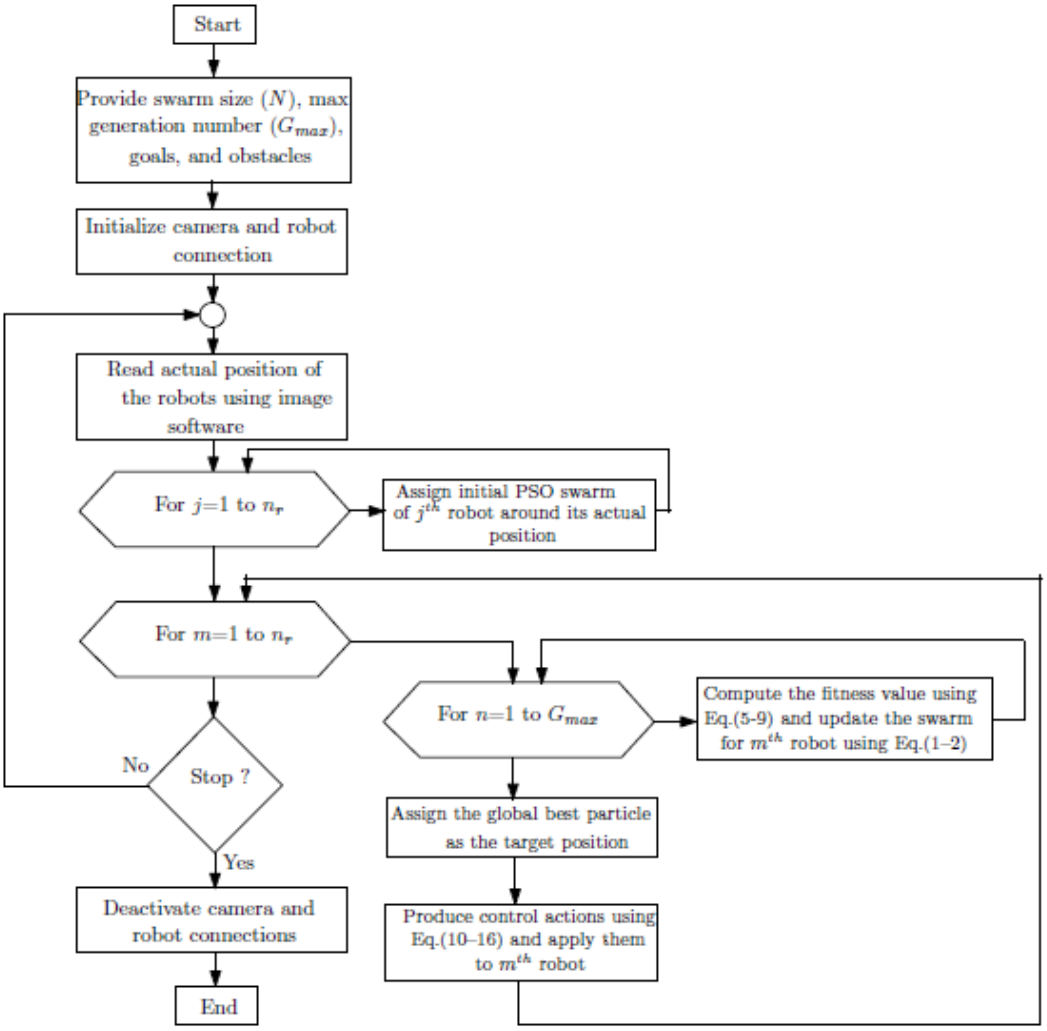

Figure 5 Flow chart for the real-time program.

A Aggregation formation experiment

The purpose of an aggregation formation experiment is to gather robots on the setup from their initial positions to the desired final form, which is predetermined by setting the D = $[D_{jk}]$ matrix to the relevant values by the user. In the experiments, robots are initially placed manually on any of the corners on the platform at any orientation. For this task, γ_1 in (5) is set to zero since there is no fixed goal position for the robots, γ_2 is set to 1 to obtain the distance between any two robots (j^{th} and k^{th}) defined by D_{jk}. Although there is no fixed obstacle on the platform, γ_3 is set to 1 since each robot is a dynamic obstacle that the others must avoid to prevent possible collisions with the parameter of δ^{obs}=0.15 in (9).

The aggregation experiment was done by placing robots two-by-two onto two cross corners of the platform, facing the center of the platform. The distance between the robots in their final positions is defined by taking D = $[D_{jk}]$ matrix (17).

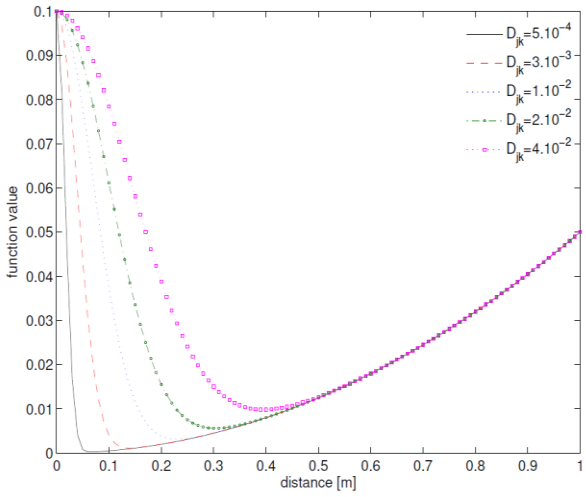

Figure 6 Curves of the function in (7) for some Djk values versus distance between two robots (a=0.1, b = 20 and c = 0.01)

$$D = \begin{bmatrix} 0 & 0.01 & 0.003 & 5.10^{-4} \\ 0.01 & 0 & 5.10^{-4} & 5.10^{-4} \\ 5.10^{-3} & 5.10^{-4} & 0 & 0.01 \\ 5.10^{-4} & 5.10^{-4} & 0.01 & 0 \end{bmatrix} \quad (17)$$

The program whose flow chart is given in Fig. 5 was designed to record the related data for each robot throughout the experiment conducted. Fig. 7 shows the observed results: Fig. 7a shows the positions of the robots on the platform during the experiment. The change of distance between two robots is given in Fig. 7b-d. In these figures, the desired distances between two robots are drawn with dotted lines. The aggregation behavior was generated by the algorithm depicted in Fig. 5. At the end of this experiment, the robots were positioned close to each other as defined and kept the desired distance between each other. Then the manual position disturbances were applied to each robot at different times. Fig. 8 shows the regulation of these disturbances throughout the experiment. As seen from the figure, manual disturbances were applied to R1 at t=5 s, R2 at t=12 s, R3 at t=30 s, and R4 at t=14 s time instances. These disturbances were eliminated by applying control signals given in Fig. 9. After applying the appropriate control actions, each robot returned to where it should be close to its previous position in an acceptable tolerance interval. Fig. 10 shows the result of another experiment with five robots.

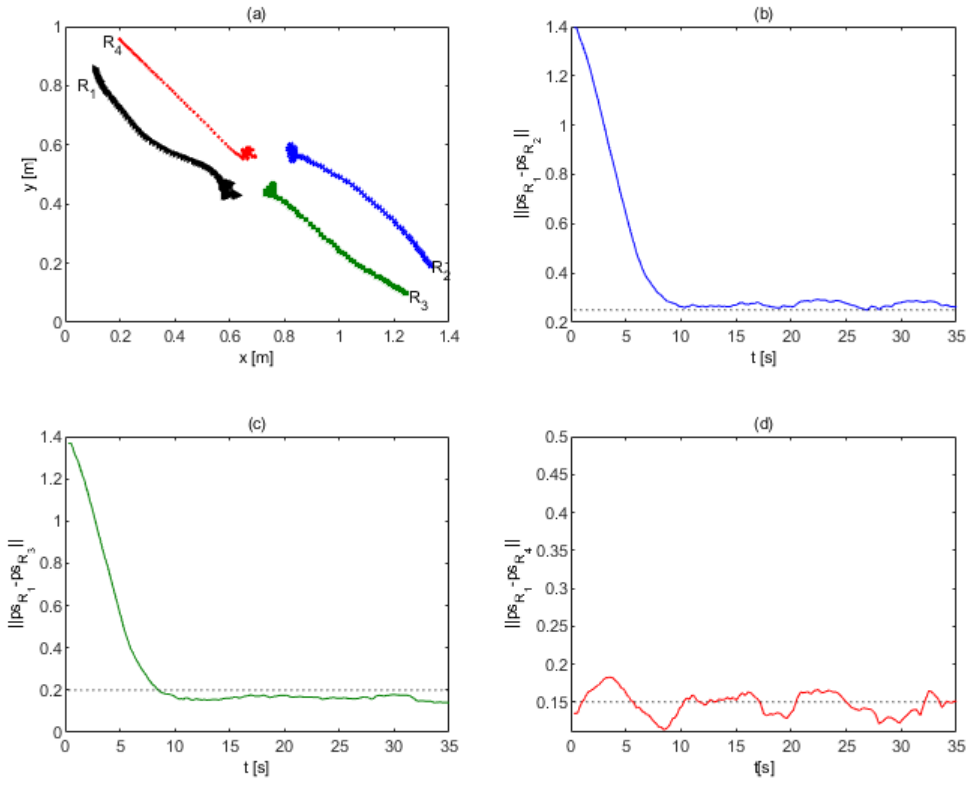

Figure 7 Experimental result for the aggregation task with four mobile robots: positions (a), distance between R1 and R2 (b), distance between R1 and R3 (c), distance between R1 and R4 (d).

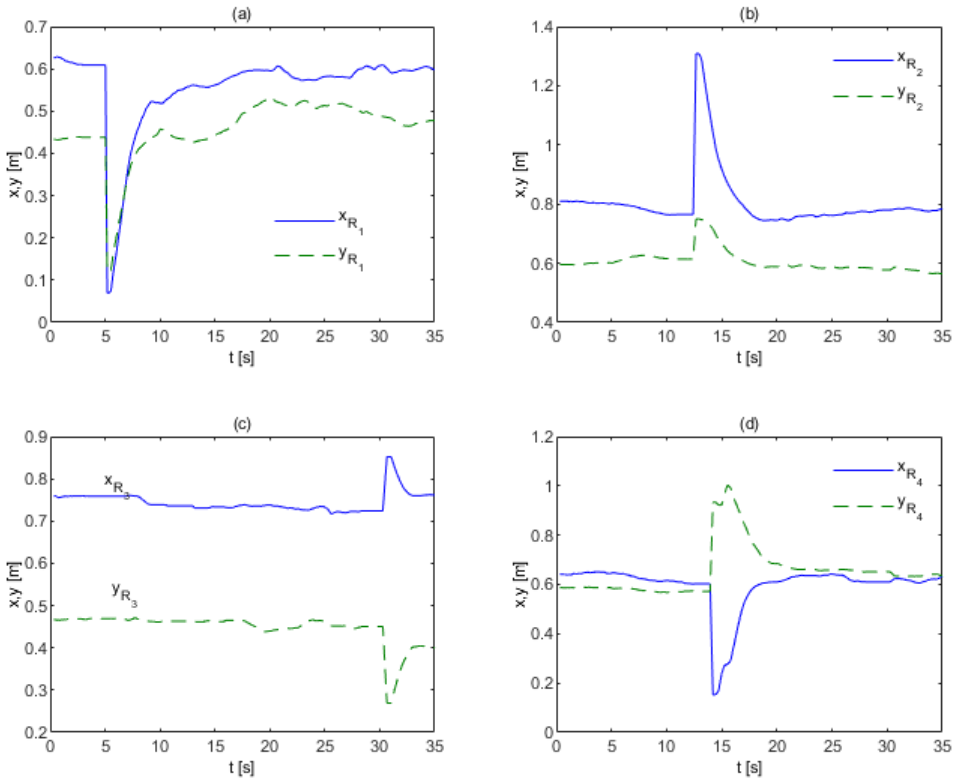

Figure 8 Changing positions under manual disturbances and regulations of them: Changing position for R1 (a), changing position for R2 (b), changing position for R3 (c), and changing position for R4 (d).

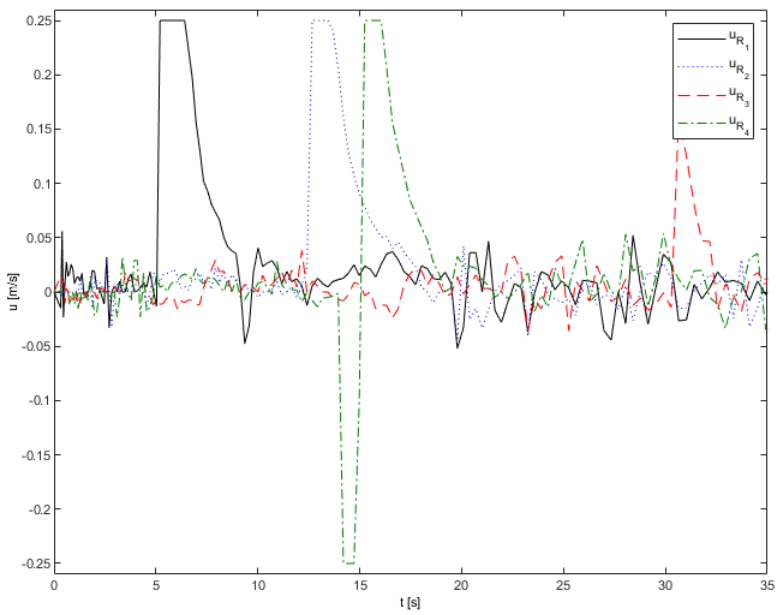

Figure 9 Control signals applied to the robots to regulate the disturbances.

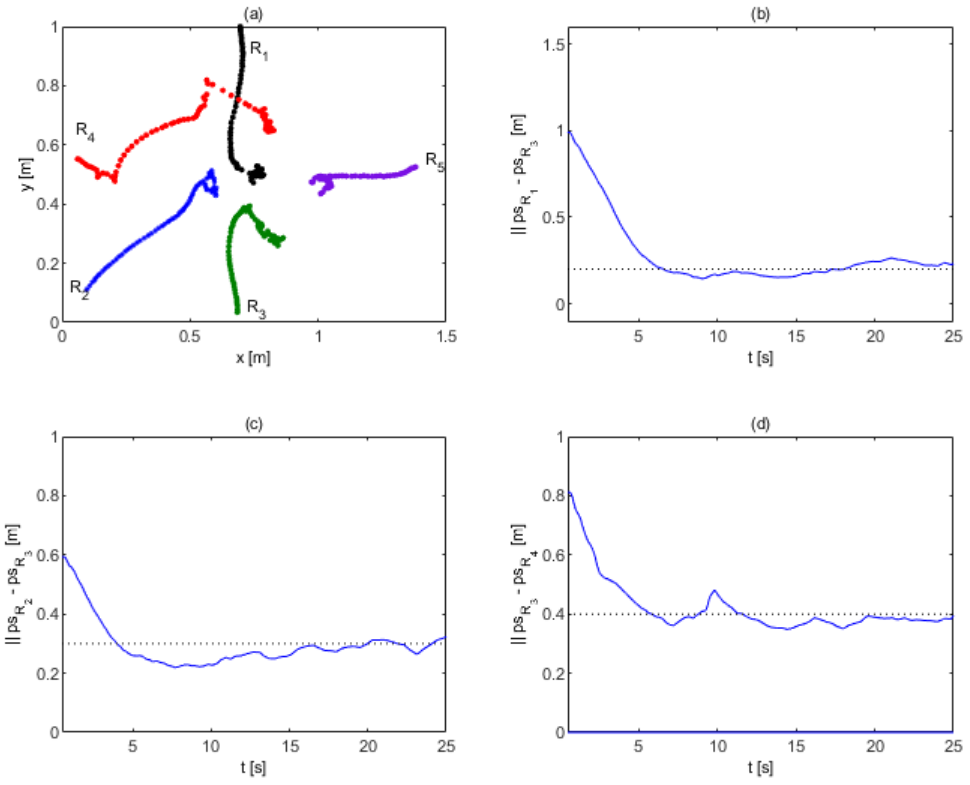

Figure 10 Aggregation task with five robots: positions (a), distance between R1 and R3 (b), distance between R2 and R3 (c), distance between R3 and R4 (d).

B Following a moving goal experiments

In this group of experiments, only one robot was used on the platform without obstacles to follow a moving goal. For this task, since there was only one robot on the platform, the parameters in (5) are set to γ_1=1, γ_2=0, and γ_3=0. The search radius was recorded as three times that of equation (4) to enable quicker robot movement since the moving goal position changes quickly.

The experiment with a robot to follow a moving goal shaped like a figure eight was executed. The results of this experiment are shown in Fig. 11. As can be seen from Fig. 11a, the proposed method determined positions precisely on the goal trajectory. The robot has followed this trajectory with a delay of approximately 0.7s which occurred because there was a large distance between the robot's initial position and the initial trajectory position on the platform, as seen in Fig. 11b.

The setpoint of the PD controller is the point determined by the PSO at the current time interval. For this reason, unfortunately, the PD control cannot handle the robot's movement in one sample time. As seen in Fig. 11a, the robot's motion is on the exact trajectory except for the bottom left and top right areas. In these areas, the robot takes turning commands from the controller according to its current and goal positions (see Fig. 11b and d). These sharp turning commands cause the robot to spin; hence the robot's position departs slightly from its trajectory. Since our primary aim is to obtain the PSO algorithm's path planning performance, we were not concerned about the details of this problem.

C Moving to fixed target points experiments

The third task chosen was to move the robots from an initial position to fixed target positions in the environment with obstacles. For the task, the parameters in (5) were set to γ_1=1, γ_2=0, and γ_3=1. δ^{obs} in (9) was set to 0.10 and 0.20 for static obstacles and dynamic obstacles (i.e., the other robots), respectively. The LSA radius is the value defined by (4).

This task was experimentally analyzed by putting two robots on each of the opposite diagonal corners of the platform. Two robots were initially directed towards the center, and the other two robots were directed outward on the platform. Fig. 12 illustrates the obtained experimental results. In the figure, the red points and the green lines show the positions and orientations of the robots. The black points are the target positions indicated by the PSO of the

current time interval. As seen from the figure, each robot has arrived at its goal position by following the positions (p^{gb}) determined by the PSO. The paths of R2 and R4 are especially significant in terms of their avoidance of collisions and obstacles. The path of R4 clearly shows the avoidance of obstacles: as the robots approach each other, the PSO dictates that the robot R4 wait at its actual position for the time interval 7–10 s. Then the PSO generates a suitable path by keeping a safe distance between the robots and obstacles. The control signals produced in the experiment are given in Fig. 13.

In Fig. 14, experimental results for another experiment with different initial positions are shown. The third experimental result with a denser obstacle environment is also given in Fig. 15. As seen in the diagram, robots were initially positioned near the center of the platform, and their goal positions were defined near the corners. Robots reached their goals by avoiding obstacles and collisions.

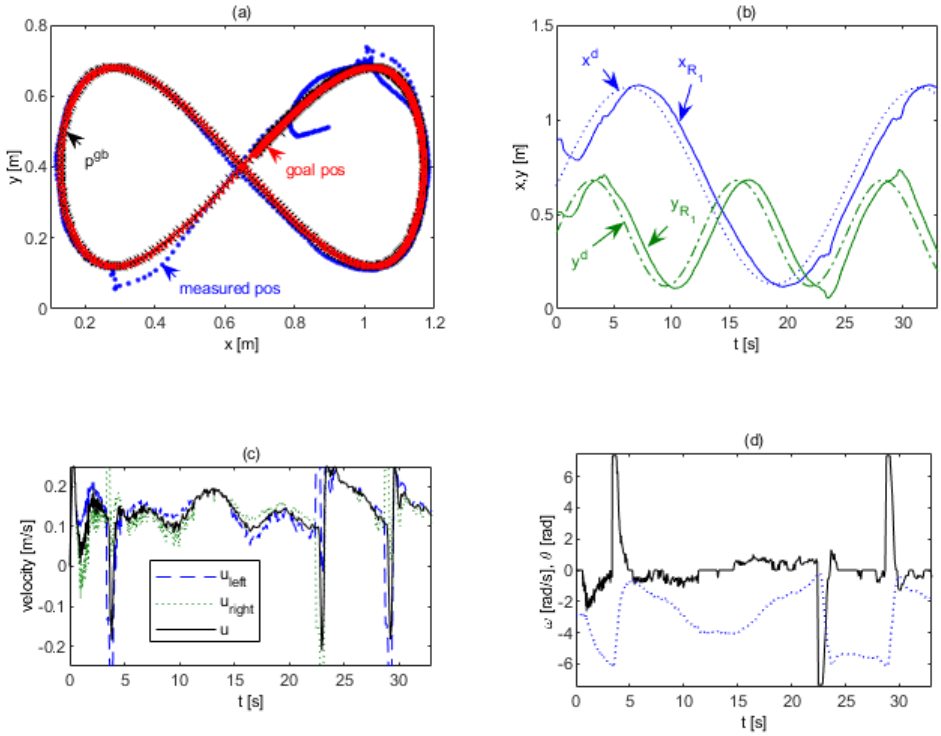

Figure 11 Tracking a figure-8 shaped trajectory with one mobile robot: positions in 2D (a), positions in time (b), applied control signals (c), and angular velocity ω and orientation θ (d).

V CONCLUSION

This paper introduces an on-line path planning strategy based on the PSO for a swarm of mobile robots in real time. The proposed method uses an on-line independent PSO for each mobile robot in a swarm to determine the most appropriate position around the actual position of the robot (called local search area, or LSA) for the next step. This method searches for possible candidate positions using PSO search with a hybrid structural cost function consisting of three parts. While it searches for the target position of each agent (robot) for the next sampling period, it also considers avoiding collisions with obstacles and other agents as well as achieving the desired formation such as aggregation, moving to the goal, and following a trajectory on the platform. This strategy has been tested in an experimental setup. The performance obtained from various experiments results shows the availability and applicability in real-time of the proposed method.

ACKNOWLEDGMENT

This research was funded by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 2219/2009.

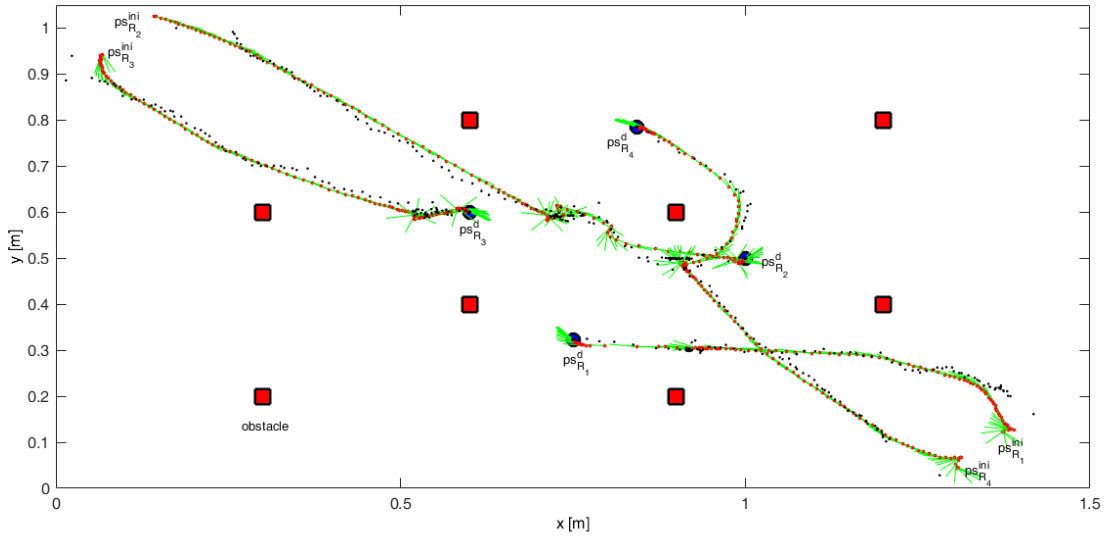

Figure 12 Four mobile robots in an environment with obstacles.

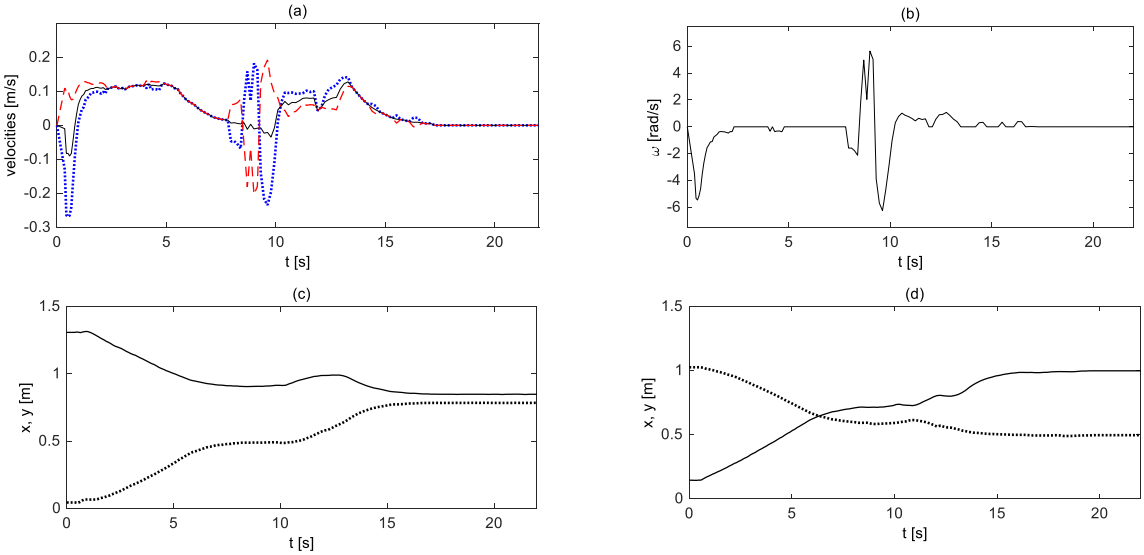

Figure 13 Control signals applied (a) and angular velocity (b) determined by the controllers for R4; measured positions changing with time for R4 (c) and R2 (d).

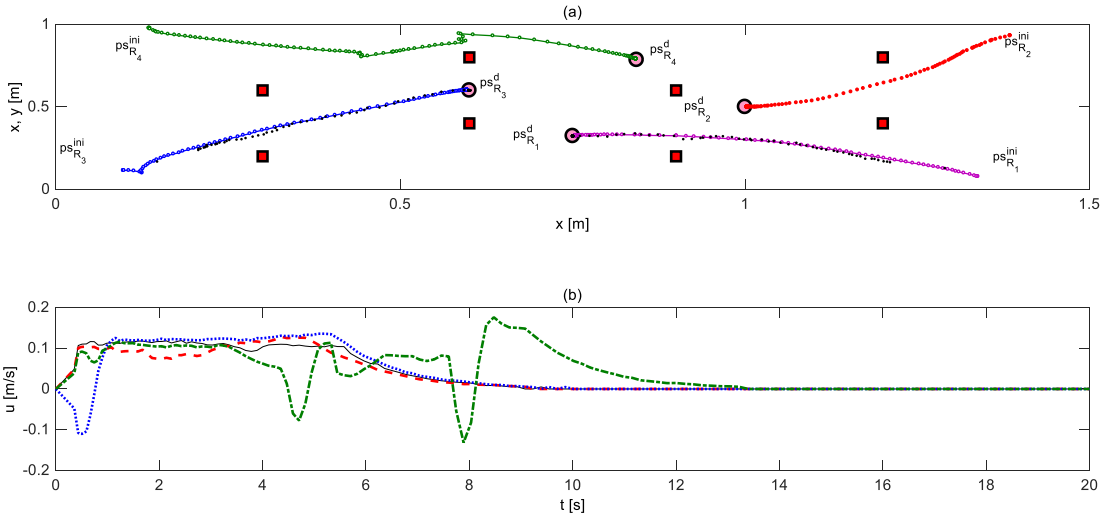

Figure 14 Path planning performance of going to fixed goals defined around the center of the platform with four robots located in the corners (a) and control signals applied to the robots (b).

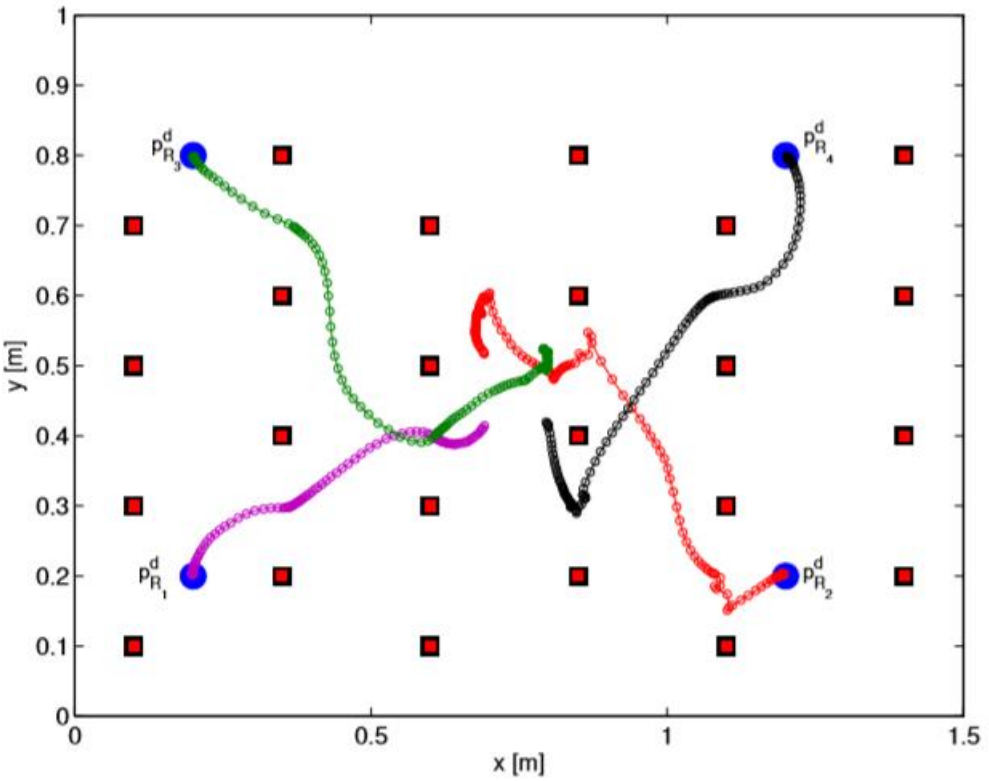

Figure 15 Path planning performance of going to fixed goals defined nearby the corners with four robots initially located in the center of the environment with denser obstacles.

REFERENCES

[1] Pal, N. S., Sharma, S., "Robot path planning using swarm intelligence: A survey". *International Journal of Computer Applications*, 83 (12), pp.5-12, 2013.

[2] Alomari, A., Phillips, W., Aslam, N., Comeau, F., "Swarm intelligence optimization techniques for obstacle-avoidance mobility-assisted localization in wireless sensor networks". *IEEE Access*, vol. 6, pp. 22368-22385, 2018.

[3] Duan, H., Qiao, P., "Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning". *International Journal of Intelligent Computing and Cybernetics*, vol. 7, no. 1, pp. 24-37, 2014.

[4] Contreras-Cruz, M. A., Ayala-Ramirez, V., Hernandez-Belmonte, U. H., "Mobile robot path planning using artificial bee colony and evolutionary programming". *Applied Soft Computing*, vol. 30, pp. 319-328, 2015.

[5] Doctor, S., Venayagamoorthy, G. K., Gudise, V. G., "Optimal pso for collective robotic search applications". In: Proceedings of the 2004 Congress on Evolutionary Computation (CEC2004). Vol. 2, pp. 1390-1395, Portland, Oregon, 19-23 June 2004.

[6] Lei, K., Qiu, Y., He, Y., "A novel path planning for mobile robots using modified particle swarm optimizer". In: Proceedings of 1st International Symposium on Systems and Control in Aerospace and Astronautics (ISSCAA 2006). Vol. 1-2, pp. 981-984, Harbin, China, 19-21 January 2006.

[7] Chen, X., Li, Y., "Smooth path planning of a mobile robot using stochastic particle swarm optimization". In: Proceeding of the 2006 IEEE International Conference on Mechatronics and Automation (IEEE ICMA 2006). Vol. 1-3, pp. 1722-1727, Luoyang, China, 25-28 June 2006.

[8] Vahdat, A. R., NourAshrafoddin, N., Ghidary, S. S., "Mobile robot global localization using diferential evolution and particle swarm optimization". In: Proceeding of IEEE Congress on Evolutionary Computation (CEC 2007), pp. 1527-1534, Singapore, Singapure, 25-28 September 2007.

[9] Rigatos, G. G., "Distributed gradient and particle swarm optimization for multi-robot motion planning". *Robotica*, vol. 26, pp. 357-370, 2008.

[10] Wong, C.-C., Wang, H.-Y., Li, S.-A., "Pso-based motion fuzzy controller design for mobile robots". *International Journal of Fuzzy Systems*, vol. 10, no. 1, pp. 24-32, March 2008.

[11] Bratton, D., Kennedy, J., "Defining a standard for particle swarm optimization". In: Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007). Vol. 2, pp.120-127, Hilton Hawaiian Village, Honolulu, Hawaii, 1-5 April 2007.

[12] Gazi, V., "Swarm aggregations using artificial potential and sliding-mode control". *IEEE Transaction on Robotics*, vol. 21, no. 6, pp. 1208-1214, 2005.

[13] Vieira, F. C., Medeiros, A. A. D., Alsina, P. J., "Dynamic stabilization of a two-wheeled differentially driven nonholonomic mobile robot". In: Proceedings of the Simpósio Brasileiro de Automação Inteligente (SBAI 2003), pp. 620-624, Bauru, SP, Brazil, August 2003.

[14] de Jong, W., "Development of an experimental research platform with mobile robots including an application of formation control", Master's thesis, Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherland, 27 August 2008.

[15] Kooij, N., "The development of a vision system for robot soccer", Master's thesis, Department of Computer Science, Distributed and Embedded Systems Group, University of Twente, Twente, The Netherland, 2003.

[16] Schepers, E., "Improving the vision of a robot soccer team", Master's thesis, Department of Computer Science, Distributed and Embedded Systems Group, University of Twente, Twente, The Netherland, 2004.

Cihan Karakuzu is full professor at the Department of Computer Engineering, Faculty of Engineering, Bilecik Şeyh Edebali University, Bilecik, Turkey. His research interests include intelligent and artificial learning systems, fuzzy and neuro-fuzzy systems, data-driven models, system identification and modeling and meta-heuristic algorithms.

Robert Babuška is full professor at the Department of Cognitive Robotics, Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, Delft, The Netherland. His research interests include reinforcement learning, nonlinear control, data-driven model construction, deep learning, system identification, state-estimation, model-based adaptive control and dynamic multi-agent systems. He has been involved in the applications of these techniques in robotics, mechatronics, and aerospace.