Applied Soft Computing 11 (2011) 5498–5507

Contents lists available at ScienceDirect

Applied Soft Computing

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

A software system for collaborative robotics applications and its application in particle swarm optimization implementations

Cristian I. Vasile*, Cătălin Buiu

Department of Automatic Control and System Engineering, "Politehnica" University of Bucharest, Splaiul Independenței nr. 313, sector 6, 060042 Bucharest, Romania

ARTICLE INFO

Article history:
Received 20 December 2009
Received in revised form 30 March 2011
Accepted 1 May 2011
Available online 7 May 2011

Keywords:
Swarm intelligence
Particle swarm optimization
Software system
Collaborative robotics
Multiagent system

ABSTRACT

This paper presents a particle swarm optimization (PSO)-inspired multi-robot search application based on an innovative software system for collaborative robotic applications. The system has a multi-layer architecture which provides low- and high-level interfaces to the robots, resource (robots) management, security policies and concurrent robot access. The main result is the successful testing of the PSO-inspired algorithm on real-world experiments, using Khepera III and e-puck robots. Simulated results obtained in other studies are therefore validated by the real-world experiments. Differences between simulation and real-world experiments are presented and discussed critically.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

There are many real-world applications which require highly fault-tolerant robotic systems, and in which a single complex robot may not be appropriate, since a fatal error, either hardware, or software, leads to mission failure. The swarm intelligence (SI) paradigm, based on using multiple simple autonomous agents was inspired by the behaviour of bird flocks, fish schools and insect societies, like ants, termites or bees. For example, ants' collective behaviours (foraging, division of labour, etc.) have been studied extensively and are used as inspiration sources for similar robotic swarms behaviours [1]. The number of agents in the swarm gives the system its robustness, and the loss of robots may not hinder the successful completion of the task. Systems which use SI techniques have distributed control and information is used by every individual to update its state or react to it, therefore no single decision-making unit exists. Interactions between robots themselves and with the environment are exploited to develop robust, goal-oriented and sometimes emergent collective behaviours [2].

SI fundamental ideas have led to the development of many algorithms, some of which are briefly described in the following. Ant Colony Optimization (ACO) is a path optimization algorithm class based on the search behaviour of ants and uses virtual ants and pheromones to find the best path. A routing scheme using ACO has been developed by Di Caro and Dorigo [3]. Particle swarm optimization (PSO) is a global stochastic optimization algorithm for problems in which the best solution can be represented as a point or surface in a n-dimensional space. Stochastic Diffusion Search (SDS) is a probabilistic global search algorithm used for problems in which the objective function can be decomposed into multiple independent partial-functions.

It has been shown that such collective behavioural approaches can be used to solve a variety of tasks. Robots can assemble themselves to stabilize over uneven ground, to cross gaps or transport large objects [4]. Other task may include inspection of inaccessible structures like jet turbine engines with a swarm of miniature robots [5], sewage or industrial installations. One particular interesting problem is locating one or more targets in an unknown environment, because it is well suited to swarm robotics. Adapted variants of the PSO algorithm have been presented in [6,7] and were studied in a simulated environment. The PSO approach was shown to have better performance than a genetic algorithms (GA)-based one [8]. In [9] an inner PSO-inspired search strategy was tuned with an outer PSO, while a hybrid PSO with fuzzy logic controllers strategy was considered in [10]. The effects of real-world factors [11] and the group size [12] were also studied. In this paper a PSO-inspired search algorithm will be presented and the results of real-world experiments on Khepera III and e-puck robots will be discussed. There are two main contributions of this paper. Firstly, a software system for implementing multi-robot applications is proposed. Secondly, previous work on PSO-inspired multi-robot search is validated by implementing and running the approach on real robots (Khepera III and e-puck research platforms).

* Corresponding author. Tel.: +40 0722679831; fax: +40 0212500745.
E-mail addresses: cvasile@ics.pub.ro, wasserfeder@yahoo.com, ptheory@gmail.com (C.I. Vasile), cbuiu@ics.pub.ro (C. Buiu).

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.05.009

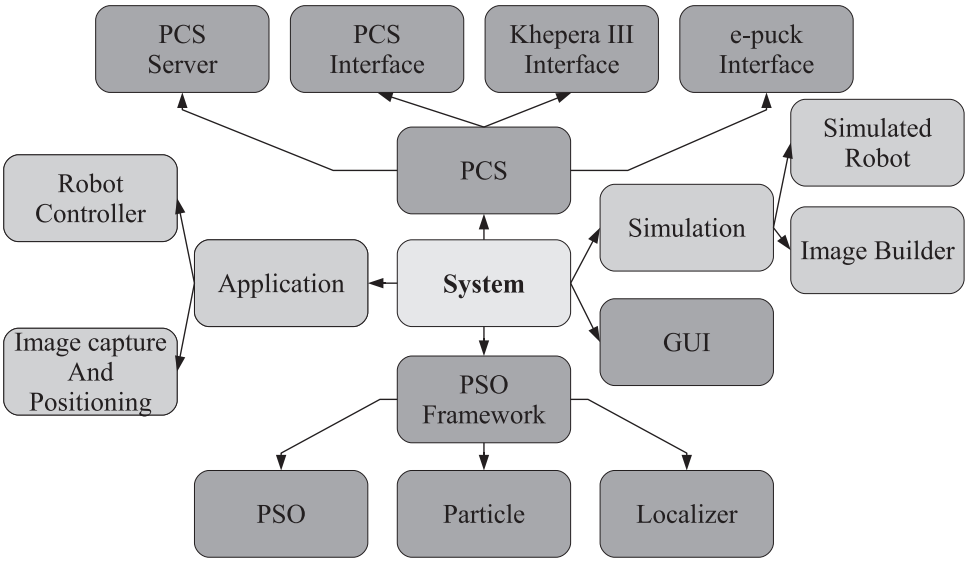


Fig. 1. Block diagram of the software system.

Section 2 presents the architecture of the proposed software system, while the next section describes the collective search application used for testing the software system. Section 4 gives some experimental results and critical comments. The paper ends with conclusions and directions for further research.

2. Software system

The software system currently supports Khepera III and e-puck robots, but support for other types of robots can be easily added. A short description of the robots is given below in order to point out the available hardware features.

The Khepera III is a research robotic platform. Features available include 11 infrared sensors, 5 ultra-sonic sensors, 2 DC motors with incremental encoders, swappable Lithium-Polymer battery pack and a built-in Bluetooth module. The robot is designed to be modular by using a new fully open extension bus compatible with the KoreBot system. The KoreBotLE extension has an Intel XScale PXA-255 400 MHz microcontroller and is flashed with a Linux operating system, enhancing the robot's computing power.

The e-puck robot is another research robotic platform which has 8 infrared sensors, 3 microphones, an accelerometer, a VGA camera, 2 DC motors with encoders, a led ring, a speaker, swappable Lithium-Polymer battery and a built-in Bluetooth module [13].

Communication with a Khepera III (without the KorebotLE extension) or an e-puck can be done through a Bluetooth connection via their BSP (Bluetooth Serial Port) service. This type of connection has many disadvantages. Because only one device can use the Bluetooth connection at a time, switching between different clients who want to access the robot is very slow. It does not allow to implement access policies, remote access from outside the robot's communication range nor traffic logging for future use. Another very important problem is the lack of protection against accidental misuse of Khepera III's 'U' command. This command is used to enter the serial boot loader mode to upgrade the firmware and can corrupt the dsPic's memory [14].

The software system proposed was created to solve the above mentioned problems. Other features were also added to help and speed-up robot and multi-robot application development. It was designed to be flexible, easy to use and extensible. In order to meet these requirements, the system was split into components, each one being developed independently. This also lowers the maintenance and testing costs. The system can be extended to add support for various other robots and features as well.

The software system defines the following components: the Poll and Command Service (PCS), a generic concurrent PSO framework, a GUI and the collective search application created using this system (Fig. 1). The implementation was done in Python 2.5 and was tested under Ubuntu 8.04.

2.1. PCS service

The service offers an abstraction layer in the design and development of robotic applications. This component is the core of the software system and is designed as a resource manager for the Bluetooth connection to the robot. It is composed of a server program, a custom communication protocol, a low-level API and a high-level interface.

The PCS server mediates the communication between the clients and the robot. Request from clients are sent though TCP/IP connections to the server, which sends appropriate commands to the robot via the Bluetooth connection. This solution also eliminates the need for Bluetooth devices to access the robot, especially for desktop PCs which do not usually come with such extensions. Clients do not need such devices, since the Bluetooth connection is maintained and managed by the server. The overall communication rate of the PCS service is limited by the Bluetooth serial communication, which has a rate of 115,200 bps. The current version of the server implements a simple CREW access policy. The read and write permissions were defined by splitting up the robot's commands into two groups: poll commands and set commands. These are used to check the permissions for a pending request. For the Khepera III robot, the 'U' is not recognized by the server in order to protect the robot from misuse. The server also supports logging and has an interactive mode for use by an administrator. Communication with the server is done by using the PCS communication protocol. The protocol defines two types of packages, request and response, and the request handling procedure. It is a protocol over TCP/IP and uses ASCII messages. The package format is composed of two fields: [Type][Message]. Type field represents the first byte of the package and can take one of the five following values:

- 'A' – acknowledge response package
- 'N' – not acknowledge response package

- 'R' – request write permission request package
- 'P' – poll command request package
- 'S' – set command request package

The message field contains additional information passed to the client in response packages or the command to be forwarded by the server to the robot via Bluetooth in poll and set request packages. The message field in write permission request packages is reserved for future use and is ignored. The total length of the message is limited to 1024 bytes. An example of a PCS package is: $||S|D, l5000, l5000\langle LF\rangle||$, which is a set command request package for setting the speed of the two motors to 5000 encoder steps per second for a Khepera III robot.

The request handling procedure implemented by the server has three steps. In the first step the type and message fields are extracted from the package. Second, it is checked if the package is of permission request type and if so, a response package, given by the active access policy, is sent back. Third, it is checked if the package is of poll or set request type. Next, it is checked if the message body is void and then the permissions are verified. If the test is passed, the message is forwarded to the robot and according to its response, a PCS response message package is sent back to the client. The acknowledge package will contain, in its message field, the response from the robot.

A low-level API was created, implementing the client-side PCS protocol, to be used in application development. A feature of this API is the message format and parameter values checking mechanism, according to the Khepera III manual [14] and the e-puck firmware. This API offers two methods for requesting write permissions and for sending poll or set commands. This component is used as a base for the high-level robot (Khepera III, e-puck) interface. The interface implements methods for all commands in the user manual (except for the 'U' command on Khepera III) and for reading and setting the robot states, i.e., LEDs, speed, position, etc. It also maintains local robot states, updated at every successful method call. This is useful in situations where a state cannot be read directly, like the LEDs state, or where it is not necessary to poll the robot for the values, for example for logging or debugging. The robot interface implements two high-level movement methods, one for moving forward a given distance and one for rotating a given angle, based on the robot's physical parameters given in the user manual. The forward movement method implements obstacle detection and can use a given obstacle avoidance procedure to handle the interruption. The default action, when none is given, is to stop the robot. Because the rotation is around the robot's axis, it is not necessary to check for collision in the rotation method. Both methods however return whether the target was detected or not.

2.2. PSO framework

The particle swarm optimization (PSO) algorithm is a global stochastic optimization technique developed by Eberhart and Kennedy [15]. It was inspired by the social behaviour of bird flocks. PSO is a population-based technique, the population (swarm) being composed of individuals, called particles. Problems solved by this method have solutions which can be represented as a point or surface in a n-dimensional space. The solution surface is used to define a fitness function, which the algorithm is trying to optimize. The system is initialized with a random population of solutions and its state is updated in every generation. Particles "float" through the search-space, following their best fitness value (local optimum), obtained by evaluating the fitness function, and the best solution in its neighbourhood or in the entire swarm (global optimum). The global optimum is the best solution from the local ones. The state of a particle, position, velocity, best fitness value and position, is updated by the following formula:

$$v_{ij}(k+1) = w * v_{ij}(k) + rand() * c_1 * (pbest_j(k) - x_{ij}(k)) + rand() * c_2 * (gbest_j(k) - x_{ij}(k)) \quad (1)$$

$$x_{ij}(k+1) = x_{ij}(k) + v_{ij}(k) \quad (2)$$

where v_{ij} and x_{ij} are the velocities, respectively the positions, of the particles in the search-space. w is a inertial parameter used to slow the velocity over time, ensuring that the swarm converges. It also prevents the swarm from exploding. c_1 and c_2 are learning factors and represent the influence the local optimum (p_{best}) and the global optimum (g_{best}) have over the velocity. Usually this factors are set to 2. $rand()$ is a random variable uniformly distributed in [0,1] [6].

The PSO technique does not use a complex model of the world. Each particle (robot) has to know only its current position, the position of its local optimum and the position of the global optimum. Because the PSO algorithm has a stochastic nature, it does not depend too much on the accuracy of the localization system. Due to these properties, the PSO is easy to implement and does not require large computational power and memory. This approach is different from others which may use maps and other knowledge about the environment in order to plan actions and execute them [16] and which, most of the time, may require more computational power and memory.

The PSO framework is a generic concurrent implementation of the above described technique. Although it was designed to be used in collaborative robotics applications, it can also be used in computational optimization problems as a stand-alone module. It is composed of four components: PSO, Particle, Localizer and Barrier.

The Particle module defines an interface to be used by the PSO implementation. It maintains the state of a particle (position, velocity, best local fitness) and offers thread-safe methods to access it. This is necessary because every Particle is a thread. The behaviour of the particles is defined by implementing the fitness evaluation, movement and target detection methods. These are used in the thread, associated with the particle, inside a loop, synchronized with the PSO procedure. In every iteration the convergence condition is checked and the particle moves to the next position. The movement of the particle may correspond to a robot's movement.

The Localizer module is used to define a global localization mechanism. It offers methods to update the positions of the particles and to determine the target's position. In computational applications this module has no meaning and can be ignored.

The PSO module contains the generic and concurrent implementation of the technique. It initializes the procedure and runs the main loop of the algorithm, which is synchronized with the particles ones. In every iteration the particles' states are updated first using the given localizer, then by computing the local and global optima and finally by using the PSO formula. Particles' states are limited by given intervals, bounding the search-space and the maximum absolute value of the velocity. The stop condition is checked using the method supplied by the Particle module. When the stop condition is reached the associated threads are released and then they join with the main one.

Synchronization is made possible by the Barrier module, which contains the implementation of a custom re-entrant barrier. It is designed to unlock completely if a certain condition is met, by setting the internal flag. This modification to the standard barrier allows the main procedure to release the particles' threads when the stop condition is met.

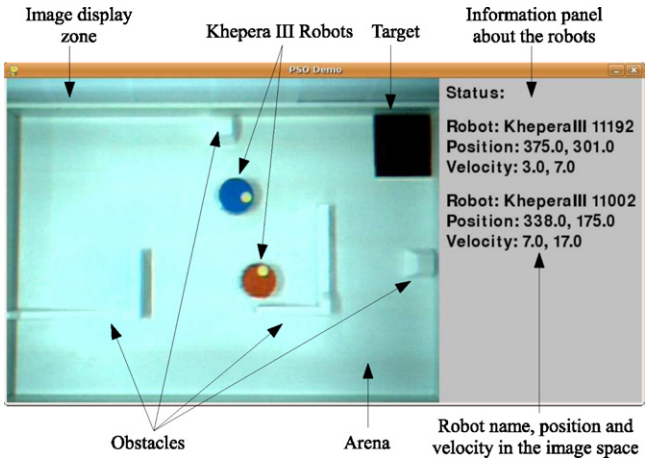


Fig. 2. GUI displaying the arena and robot-particles' states.

2.3. GUI

The GUI component is an auxiliary module and implements a lightweight interface used to display the arena and the particles' states in real-time (Fig. 2). The particles' states represent the current position and velocity vectors in the image (PSO) search space. Each particle is given a name to identify it. The image source can be either from a webcam or simulated.

3. The collective search application

The software system proposed in this paper was designed to support the development of collective robotic applications. Using this system, a collaborative search application has been developed. Its purpose is to validate the proposed architecture, software solutions, and the PSO-inspired algorithm through real-world experiments.

3.1. Search problem

Finding one or more targets in an unknown environment is an important task well suited for swarm robotics. The behaviour and architecture of the multi-robot system are designed in correspondence with the specific localization problem it addresses. It is important to specify that the search problem depends on three major factors:

- the target(s): their number, nature, dynamics, and mode of appearance;
- the search agents: their number, architecture and capabilities;
- the environment: its dimension, type, dynamics, structure, and previously known environmental information.

The movement and nature of the targets are very important factors. A movement prediction and adaptation method may be required. This may be created using a target model or adaptive learning. The nature of the target plays an important role also. Odour localization was considered in [17]. This problem depends heavily on the environment (wind direction) as well. It is important to notice that some factors have meaning in relationship to others. For example, the dimensions of the environment is taken into account relative to the dimensions of the target(s) and robots. Another important issue is the availability of robots, technology, equipment and funds.

The nature of the localization defines the solution's strategy and imposes a specific behaviour upon the system, which may vary considerably from case to case. Good identification of the problem type can generate optimization possibilities and ensures the quality of the solution, by correctly covering all the possible cases.

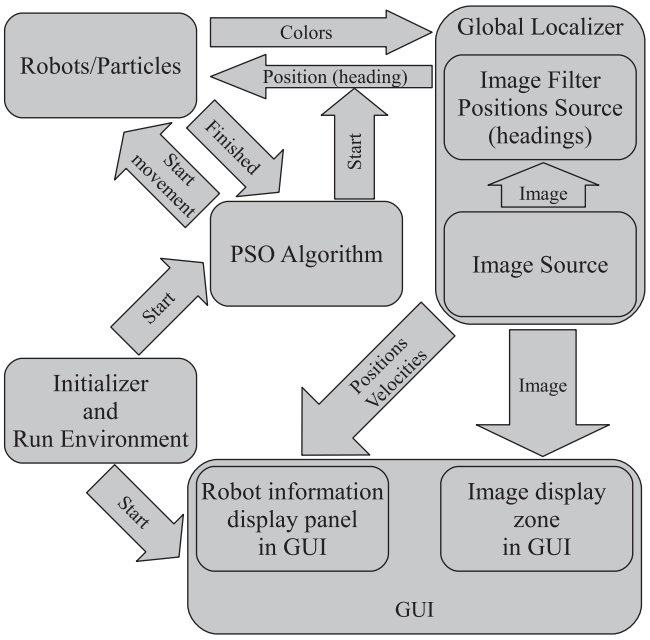


Fig. 3. Block diagram of the application.

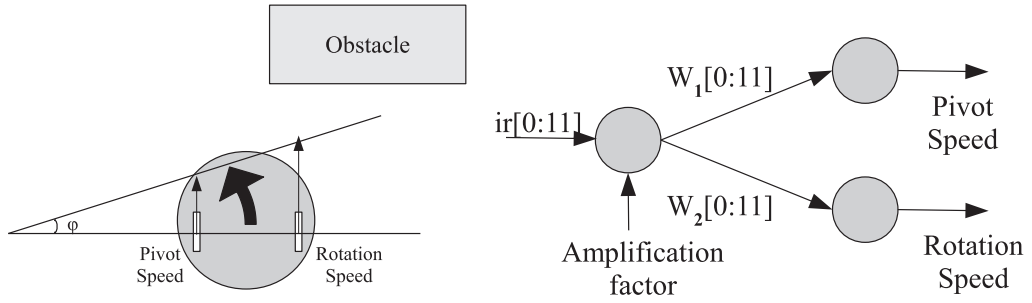
The application was designed to locate a single static target in the semi-structured, unknown, static and plain environment of the arena (see Fig. 2). PSO-inspired algorithms have been shown, in simulated experiments, to be efficient solutions to this problem. In this paper it will be shown that those results can be reproduced in real-world experiments.

3.2. PSO model versus real-world

In order to be able to apply the PSO technique for the search problem in a real environment it is necessary to overcome some problems. These are due to the difference between the PSO model and the real-world. Particles are in one-to-one correspondence with a robot, which must obey the laws of physics. Therefore their movement is restricted: a robot, having bounded velocity and acceleration determined by the technology used, cannot "teleport" as a particle does. Therefore it needs time to adjust its heading and move to the next position in each iteration. Because each robot may take a different amount of time to get to its next position, a synchronization procedure was created. Another important difference between abstract particles and real robots is that the latter have dimensions and shape as opposed to the point-particles. A collision avoidance algorithm was implemented, because robots cannot get however close to obstacles and other robots. The computation of the fitness function must also be implemented. A detailed discussion about these differences and their effect on a PSO-inspired search algorithm can be found in [6,11].

3.3. PSO-inspired solution

The solution, as mentioned above, uses the PCS service and the PSO framework. The application is composed of a robot/particle controller, a global localizer and a run environment (Fig. 3). The run environment initializes the robots/particles' states, the PSO algorithm, the localizer and launches the GUI. It also binds together all the components.



(a) The operation mode of avoiding an object, using the pivot and rotation speeds

(b) Neural network structure

Fig. 4. Obstacle avoidance behaviour. (a) The operation mode of avoiding an object, using the pivot and rotation speeds and (b) neural network structure.

3.4. Robot controller

This module is responsible for controlling a Khepera III or an e-puck robot within the PSO search algorithm. It also contains a obstacle avoidance procedure. It was built on top of the PSO particle and PCS Khepera III or PCS e-puck modules. This binds together the behaviour of a particle with its physical instance. To distinguish between the robots, each one is given a name and a color interval in the RGB space. These are used by the GUI, respectively by the localizer.

It is important to notice that unlike generic particles used in PSO, real robots have bearings, which may differ from the ones given by their velocity vector. This inconsistency may be due to the fact that the robot may enter the obstacle avoidance mode and the fact that the particle's velocity in the PSO algorithm does not correspond to the real one. The two velocities do not correspond, because of the restrictions of the physical environment (as explained above). Therefore the velocity in the PSO search-space is used just to compute the next position to be reached.

This module defines procedures, needed by the PSO framework, to compute the fitness value, to move to the next position and to check if the target was found.

The fitness value is the two-dimensional Euclidean distance between the robot's and the target's centers. The target detection method is implemented using the two infra-red ground sensors of the Khepera III robot. Because the target is a black label, it can be detected using infra-red light. On the other hand, the e-puck robots do not have ground sensors, therefore the target is considered detected if a robot is sufficiently close to the target's position (e.g. within an area of a given radius, centered in the target).

Movement to the next position is achieved in two steps. First the robot rotates towards the new location and then it moves forward to it. The rotation angle is computed as the difference between the heading, computed by using the new velocity of the particle in the PSO search space, and the robot's current heading, obtained from the global localization mechanism. The distance to the next position is obtained by multiplying the one in search space with a conversion factor. The forward movement can be interrupted by the detection of an obstacle or another robot. These cases are handled by an obstacle avoidance procedure. The behaviour of the robot changes to "coward" Braitenberg vehicle [18]. This is implemented using a feedforward neural network. It is composed of a single two-neuron layer (Fig. 4(b)). The output values of the neurons, computed using the infra-red sensors' readings, are used to set the motor speeds of the robot. However, these speeds are not set directly, because the net does not calculate the right and left wheel speeds. Instead, a pivot speed and a rotation speed are computed. The pivot wheel is then chosen based upon the difference between the sums of the right, respectively left, sensor readings. The rotation speed is always greater than the pivot speed, except when there is no obstacle, in which case they are equal. The amplification factor and the weights were determined by experiment.

The pivot speed is set to the wheel that is further away from the obstacle, moving the robot away from it (Fig. 4(a)). The movement between speeds' updates is a uniform circular one. The center of rotation is completely determined by the two speeds. In case the robot gets very close to an obstacle, the pivot speed will become zero or even negative and the center of rotation will be between the wheels. The rotation angle depends on the two speeds and their update time. This particular implementation was used because it enables the robots to avoid very quickly other robots in close proximity.

3.5. Global positioning

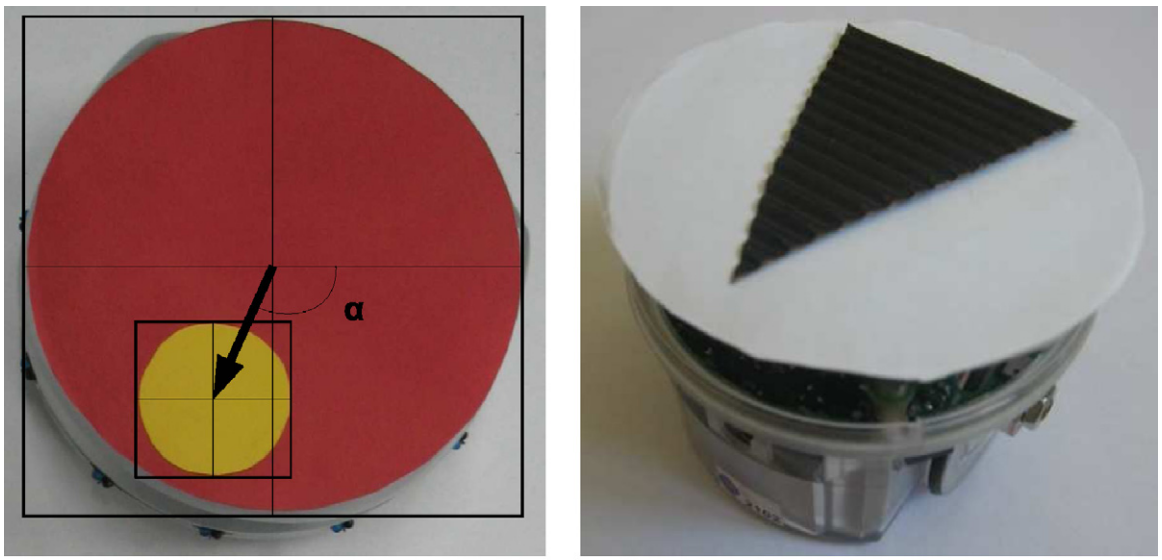
Global localization is implemented in two ways by using a custom color-based tracking procedure (used in the Khepera III experiments) or a dedicated tracking software, Swistrack [19] (used in the e-puck experiments).

3.5.1. Custom tracking procedure for the Khepera III experiments

The custom tracking module was built using the open source artificial vision OpenCV (Open Source Computer Vision) and the PIL (Python Image Library) libraries. It is responsible for updating the positions and heading of particles between the PSO algorithm's iterations. It also provides the GUI with images from a webcam. It is built on top of the Localizer module and implements a global positioning system. A relative positioning system used in collaborative robot applications can be found in [20]. A detailed presentation about positioning systems can be found in [21].

The localization system uses a Microsoft LifeCam VX-3000, positioned over the arena. Robot identification is implemented by attaching colored disks on top of the robots. The disks are blue, red and green colored, and each contains a smaller yellow interior disk. This interior disk indicated the front of the robot. The robots' positions and headings are computed based on the disks' colors and the interior disks. The target is a black square on the arena surface (Fig. 2). The system uses three-dimensional interval vectors, each component corresponding to one RGB channel, to identify the colors: red, blue, green, yellow and black.

The positions are computed by extracting color blob boundaries from the image for each robot's color. Then for each robot the boundary of its inner yellow blob is extracted. The center of the outer blob is the robot's position. The heading is determined by the position-vector defined by the centers of the two disks, inner and outer (Fig. 5(a)). Positions are absolute and headings are taken clockwise from the x-axis. The point of reference (the origin of the axes) is the upper left corner of the image.



(a) Color based tracking (b) Black and white marker tracking

Fig. 5. Robot localization. (a) Color based tracking and (b) black and white marker tracking.

The locations and headings are used by the PSO algorithm to update the robot-particles states before computing the fitness values and the local and global optima. It is important to mention that a delay is inserted before each start of update procedure. This ensures that the camera has been stabilized. In addition, the access to the internal image buffer is synchronized, because images from the webcam are used in the localization procedure and in the GUI as well. This module also collects state information (position and velocity in the image-search-space) from the robots and delivers them to the GUI.

3.5.2. Interfacing with the Swistrack tracking software for the e-puck experiments

In the experiments using the e-puck robots, a webcam (Microsoft LifeCam VX-3000) placed above the arena is used as well for global positioning. In this case a dedicated multi-object tracking software, Swistrack, is used to compute absolute positions and headings of the robots. Markers are also placed on the top of the e-puck robots. In this case, the marker is a black triangle on a white disk (Fig. 5(b)). Swistrack uses blob detection and tracking in order to compute positions and headings. Interfacing with the Swistrack software is done through a TCP/IP connection.

This positioning system uses identical markers, therefore a method is implemented in order to establish the correspondence between robots and tracked markers. This correspondence has to be done before running the experiments. In order to establish the correspondence each robot is rotated a fixed angle (30°) while the other robots do not move. Rotation is used because this movement does not involve obstacle avoidance. After each robot "knows" its corresponding tracked marker the experiment may start.

3.6. Simulation module

This module is a lightweight implementation of the PSO search algorithm in a simulated environment. It was created to offer a graphical, robot-independent, method for testing and monitoring of the PSO algorithm. The program does not take into account environmental factors, nor the structure of the robots, but it implements a simple collision detection and avoidance mechanism. It uses the same PSO framework as the real-world application. Although it is a very simple implementation, the experience gained developing and using it has helped in debugging the PSO framework.

4. Results and interpretation

Experiments were conducted in order to show that the PSO-inspired algorithm can successfully locate a target in an unknown environment. These were done for different arena configurations, start locations and number of robots.

4.1. Experimental setup

The experiments were made on a flat, white, 1.5 m × 1 m surface. It is bounded by walls and contains a number of white polystyrene objects, used as obstacles, and the target, which is a black or red label stuck on the table. A webcam has been put above the arena and is used in the positioning system. Markers (colored or black and white disks) have been attached to the robots, to allow the localization system to compute their position and heading. The software run environment was implemented as a script which contains all parameters needed to run the collective search application. This includes the PSO parameters (inertial coefficient, learning factors, search-space boundaries), the camera parameters (index, brightness, contrast and hue) and the robot's parameters (name and color intervals). They are presented in Tables 1 and 2.

Table 1
Run environment parameters.

PSO parameters	
Parameter	Value
Number of particles	2, 3 (Khepera III) and 4, 6 (e-puck)
Dimensions of the search-space	2
Boundaries of the search-space	([0, 640], [0, 480])
Maximum velocity of a particle	141
Inertia coefficient (w)	1.2
Local learning factor (c_1)	2
Global learning factor (c_2)	2
Stop condition	Target found or maximum number of iterations exceeded
Maximum number of iterations	40
Conversion constant	2.34375 mm/pixel

Table 2
Camera parameters and colors.

Parameter	Value
Camera parameters	
Index	1
Brightness	50
Hue	50
Contrast	50
Delay	1 s
Robots and target color intervals for the Khepera III experiments	
Red (outer disk)	([160, 255], [0, 120], [0, 90])
Blue (outer disk)	([0, 30], [90, 140], [160, 255])
Green (outer disk)	([90, 125], [160, 225], [90, 130])
Yellow (inner disk)	([130, 255], [150, 220], [0, 120])
Black (target)	([0, 30], [0, 30], [0, 30])

Table 3
Experimental results with 3 Khepera III robots.

Experiment no.	No. of iterations	Duration
1	9	3 min 44 s
2	7	2 min 45 s
3	6	2 min 25 s
4	9	3 min 42 s
5	12	5 min 02 s

A number of hypotheses were assumed in the design of the experiments. These are as follows:

- The environment is static and unknown to the robots.
- Illumination of the environment is considered to be uniform in order to avoid dark shadows.
- There is only one target in the arena and it is static.
- The distance between a robot and a target can be "sensed" by the robot from any location in the arena, but the location (the coordinates) of the target is unknown to the robots.
- Each robot "knows" its own position in the arena. The initial position is random.
- Robots can communicate between each other at any range. They exchange data about local optima in order to establish the global optimum and also for synchronization between loops.
- There is no central coordination node, and the robots will try to follow their local (own) and global optima. The system is completely autonomous, it runs without human intervention to find the target.

In order to measure the performance of the multi-robot system, two metrics are considered, the total execution time of an experiment and the number of iterations. The total execution time is important because it is desired to find the target as fast as possible. The number of iterations, on the other hand, is used in order to measure the performance of the PSO control algorithm without taking into account the execution time of each iteration. The execution time of an iteration is determined by the localization system and the movement of the robots. The robots' movement depends on the robots' types and the movement procedure used. The number of iterations and the total execution time give a good estimation of the performance of the system.

As stated in Section 3.4, the target is considered to be found if a robot detects the black label on the arena with its ground infrared sensors in the Khepera III experiments, or if a robot is within an area of 10 cm radius centered in the target's position in the e-puck experiments.

The distance of a robot to the target is computed based on the information from the positioning system. Therefore the distance "sensed" by the robots is influenced by the noise and errors of the positioning system, in particular by the webcam. More details are provided in Section 4.2.

4.2. Results

The following comments were made based on the results:

- In the Khepera III experiment, which use the color based tracking system, the localization time is significant, about 50% of the total

Table 4
e-Puck experimental results.

Experiment no.	No. robots	No. of iterations	Duration	Arena configuration	Target position
1	4	7	54 s	Without obstacles	Center of the arena
2	4	6	47 s	Without obstacles	Center of the arena
3	4	6	48 s	Without obstacles	Center of the arena
4	4	5	42 s	Without obstacles	Center of the arena
5	4	6	48 s	Without obstacles	Center of the arena
6	6	5	37 s	Without obstacles	Center of the arena
7	6	7	1 min 03 s	Without obstacles	Center of the arena
8	6	7	1 min 06 s	Without obstacles	Center of the arena
9	6	5	35 s	Without obstacles	Center of the arena
10	6	5	38 s	Without obstacles	Center of the arena
11	4	10	1 min 45 s	Few obstacles	Center of the arena
12	4	9	1 min 25 s	Few obstacles	Center of the arena
13	4	9	1 min 26 s	Few obstacles	Center of the arena
14	4	12	1 min 47 s	Few obstacles	Down left corner of the arena
15	4	16	2 min 38 s	Few obstacles	Up right corner of the arena
16	6	8	1 min 20 s	Few obstacles	Center of the arena
17	6	9	1 min 36 s	Few obstacles	Center of the arena
18	6	10	1 min 44 s	Few obstacles	Center of the arena
19	6	12	2 min	Few obstacles	Center of the arena
20	6	7	1 min 05 s	Few obstacles	Center of the arena
21	6	13	2 min 10 s	Few obstacles	Down left corner of the arena
22	6	14	2 min 28 s	Few obstacles	Up right corner of the arena
23	4	8	1 min 17 s	Many obstacles	Center of the arena
24	4	8	1 min 11 s	Many obstacles	Center of the arena
25	4	12	2 min 10 s	Many obstacles	Down left corner of the arena
26	6	6	1 min 02 s	Many obstacles	Center of the arena
27	6	5	46 s	Many obstacles	Center of the arena
28	6	7	1 min 02 s	Many obstacles	Center of the arena
29	6	23	4 min 13 s	Many obstacles	Down left corner of the arena
30	6	10	1 min 34 s	Many obstacles	Up right corner of the arena

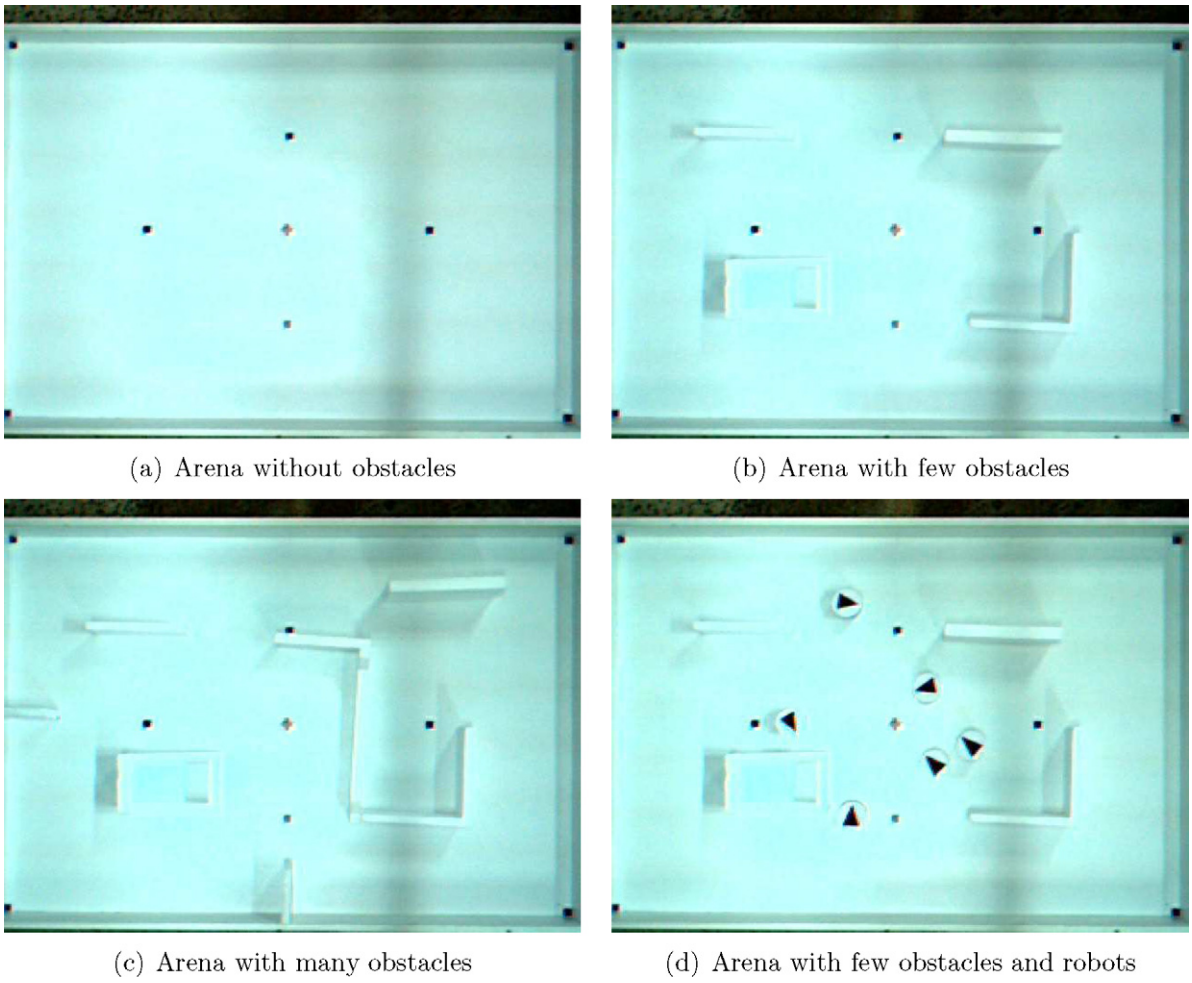


Fig. 6. Arena configuration for the e-puck experiments. (a) Arena without obstacles, (b) arena with few obstacles, (c) arena with many obstacles and (d) arena with few obstacles and robots.

execution time. In the e-puck experiments, which use the Swistrack software, the localization time is less expensive and can be neglected.

- The algorithm can overcome situations of reciprocal blockage. This is done in several iterations though.
- The algorithm's behaviour is especially highlighted by two PSO specific behaviours in particular. The first observed behaviour is that of the robots moving towards each other. This is specific to the PSO technique, because the particles velocities, used to compute their positions, are calculated based on the global optimum. Therefore the swarm has the tendency to converge. The second one can be observed when the robot is near the target, but does not go over it, i.e., does not detect it, and moves away from it. In the next iteration the robot goes back towards the target. This behaviour indicates that the robot is going towards the best fitness value. Since it has a worse value, because it moved away, the robot will rotate and go back towards the local and global optimum, towards the target.
- In all experiments, the robots managed to find the target before the maximum number of iterations was exceeded. In a number of instances, all robots were in the target's vicinity. The number of iterations and execution time are presented in Table 3 for Khepera III experiments and Table 4 for e-puck experiments. It can be deduced that the multi-robot system converges to the target stably and quickly.
- The obstacle avoidance mechanism is working well, no wall-robot or robot-robot collisions were observed, but its performance may increase if more adaptive control techniques would be used.
- Most of the movements to the next positions are interrupted by the detections of an obstacle. The robots thus enter the obstacle avoidance procedure frequently.

Three arena configurations were considered in the e-puck experiments (see Fig. 6), without obstacles (Fig. 6(a)), with few obstacles (Fig. 6(b)) and with many obstacles (Fig. 6(c)). The deployment of the swarm is a very important problem and it may have a great impact on the performance of the search algorithm. In all experiments the robots were distributed randomly in the arena. In some experiments, the target's position was changed from the center of the arena to one of its corners. The convergence of the robots to the target (center of the arena) can be seen in Fig. 6(d). The dimensions of the robots in relation to the arena can also be noticed.

In order to prove the relative effectiveness of the proposed approach, random search experiments were conducted. It is shown in Fig. 7, the minimum, maximum and mean execution time of the random search experiments in the three arena configurations and two target positions, center and upper right corner of the arena. The execution time in random searches varies very much between experiments and depends heavily on the initial positions of the robots. Moreover, in the experiments with the target in the upper right corner of the arena, the performance of the random search decreases significantly. The decrease in performance can be

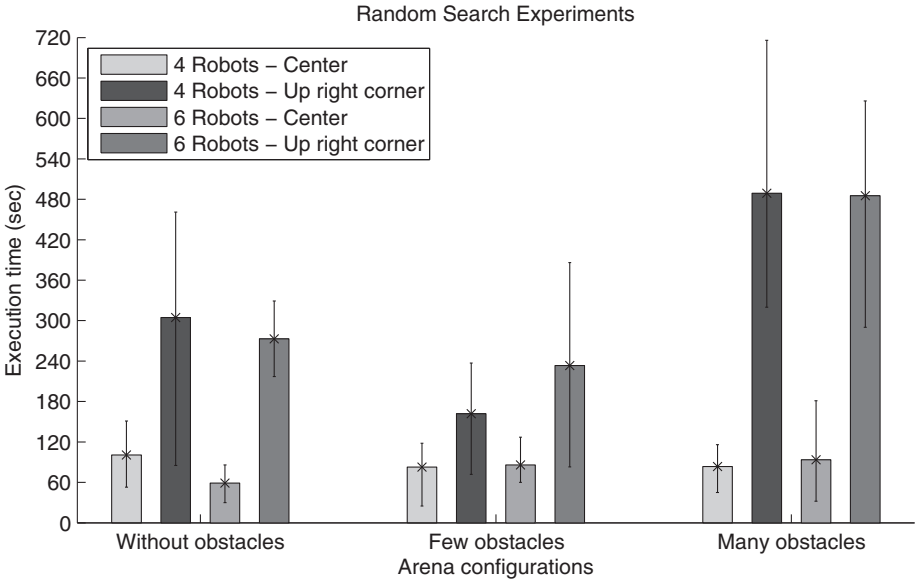


Fig. 7. Random search experiments.

explained by the fact that it is less likely for the robots to randomly reach the corner. Also, the detection area is a quarter of a disk, because of the arena boundaries. On the other hand, with the proposed PSO-based search, the execution time is much more stable and there is only a slight decrease of performance for the case with the target in the upper right corner.

It is very useful in many applications to identify and analyse the error sources in order to minimize their influence and effects on the application and results. The most important error sources in the proposed collaborative search application are:

- *Webcam* – It is the most important source of error in the application, because it is used in the computation of the positions and headings of the robots. Errors may be large due to the poor image quality obtained with the available Linux driver, gpscav1-20071224. They are important in the computation of the bearings. If the markers are not properly detected, then the robots will move on random headings. Another factor is the limited resolution of the camera with a conversion coefficient of 2.34375 mm/pixel. By experiment it has been observed that the localization errors are less than: 10 cm for position and 20° for heading, for the color based tracking algorithm, and 2 cm for position and 5°, for heading for the Swistrack based procedure.

  These errors are, however, not fatal to the PSO procedure which, due to its nature, is very fault-tolerant against positioning errors. If a robots moves in a random way, in the next iteration it will go towards the new optima. This can be explained by the fact that the PSO algorithm itself contains random components, which are used for optimization.

  The effect of the errors on the algorithm is of delaying the convergence, that is the total execution time increases.
- *Illumination* – Illumination is the second most important source of errors, because it affects both the webcam and the infra-red sensors. By creating shadows and reflections, it causes color shifts, which make the marker detection less precise. In the color based localization, larger RGB intervals must be taken in order to detect all the pixels of a given color. This, however, increases the risk of detecting pixels which do not belong to the robot. This phenomena complicate the calibration process of the localization system and contribute indirectly to its errors' magnitude.
- *Infra-red sensors* – The infra-red sensors are the main source of errors in the obstacle avoidance algorithm. They are affected by the noise in environment and by sources of infra-red light, like the sun, incandescent lightbulbs and infra-red sensors of other robots. Errors are significant, especially in the case of robot–robot interaction. The effects of these error have been minimized by implementing a special infra-red error correction module.
- *Wheel slippage* – Because the surface of the arena used is smooth, the wheels may slip on it. The errors derived from this phenomenon are insignificant in this application, but can become significant if odometric positioning is used instead of a global one.

5. Conclusions

The software system proposed in this paper has been successfully used to develop a collaborative robotics application. It was shown that it facilitates the application development, by offering standard easy-to-use robot control interfaces, a framework for the swarm intelligence PSO technique, a number of utility programs for simulation and visualization and sample and test code.

The proposed application and its implementation demonstrate the importance and the power of the PSO algorithm, even in the presence of multiple error sources. The robots have managed to find the target in a low number of iterations and in all the tests. Thus, it is shown that the swarm intelligence methods can be used successfully for solving complex tasks, like finding a target in an unknown environment. The experiments have shown that the execution times of the PSO algorithm are similar, unlike in the random-walk searches. Therefore it has been demonstrated that the collaborative solution is reliable, stable and efficient. These results, which were already established in simulated experiments [6,7], were obtained in real-world experiments, thus validating the simulated results. The authors are not aware of any previous publications on real-world experiments on multi-robot search using PSO-inspired algorithms. The main result of the study is the successful testing of a PSO-inspired algorithm in real environments.

Future work will include improvements to the PCS server and protocol. These will include adding support for Koala and other robots, support for multi-robot control from a single server, extra access policies, error messages in the Not Acknowledge response packages, support for more detailed logging and more commands to the server. Another important change will be to implement a

concurrent version of the request handling mechanism. The robot interfaces will also be extended with more high-level movement and behaviour methods. The GUI will be extended to allow more interaction with the algorithms and the robots.

However the most important change will be adding a completely distributed variant of the PSO algorithm, with a local positioning system. Other swarm intelligence techniques will also be explored.

Acknowledgement

This work was supported by CNCSIS UEFISCSU, project number PNII IDEI 1692/2008.

References

[1] T.H. Labella, M. Dorigo, J.L. Deneubourg, Division of labour in a group of robots inspired by ants' foraging behaviours, ACM Transactions on Autonomous and Adaptive Systems 1 (1) (2001) 4–25.
[2] A. Martinoli, Swarm Intelligence in Autonomous Collective Robotics: From Tools to the Analysis and Synthesis of Distributed Control Strategies. Ph.D. (Theses number 2069). École Polythechnique Fédérale de Lausanne, Department of Informatics, 1999.
[3] G. Di Caro, M. Dorigo, AntNet: distributed stigmergetic control for communications networks, Journal of Artificial Intelligence Research 9 (1998) 317–365.
[4] A.L. Christensen, R. O'Grady, M. Dorigo, Morphology control in a self-assembling multi-robot system, IEEE Robotics and Automation Magazine 14 (4) (2007) 18–25.
[5] N. Correll, A. Martinoli, Collective inspection of regular structures using a swarm of miniature robots, in: Springer Tracts in Advanced Robotics, The 9th Int. Symp. on Experimental Robotics, Singapore, 2004, pp. 375–385.
[6] J. Pugh, A. Martinoli, Inspiring and modeling multi-robot search with particle swarm optimization, in: IEEE Swarm Intelligence Symposium, Honolulu, Hawaii, 2007, pp. 332–339.
[7] J. Pugh, A. Martinoli, Distributed adaptation in multi-robot search using particle swarm optimization, in: Lecture Notes in Computer Science, 10th International Conference on the Simulation of Adaptive Behavior, Osaka, Japan, 2008, pp. 393–402.
[8] J. Pugh, A. Martinoli, Multi-robot learning with particle swarm optimization, in: International Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 2006, pp. 441–448.
[9] S. Doctor, G.K. Venayagamoorthy, V.G. Gudise, Optimal PSO for collective robotic search applications, in: IEEE Congress on Evolutionary Computation, Portland, Oregon, 2004, pp. 1390–1395.
[10] G. Venayagamoorthy, L. Grant, S. Doctor, Collective robotic search using hybrid techniques: fuzzy logic and swarm intelligence inspired by nature, Engineering Applications of Artificial Intelligence 22 (3) (2008) 431–441.
[11] J. Pugh, A. Martinoli, The cost of reality: effects of real-world factors on multi-robot search, in: IEEE International Conference on Robotics and Automation, Roma, Italy, 2007, pp. 397–404.
[12] A. Hayes, How many robots? Group size and effciency in collective search tasks, in: Proc. of the 6th Int. Symp. on Distributed Autonomous Robotic Systems, Fukuoka, Japan, 2002, pp. 289–298.
[13] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in: Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, no. 1, 2009, pp. 59–65.
[14] F. Lambercy, G. Caprari, Khepera III Manual ver 2.2. Available at: http://ftp.k-team.com/KheperaIII/Kh3.Robot.UserManual.2.2.pdf, on: 21.08.2009.
[15] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the 7th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.
[16] W. Burgard, M. Moors, C. Stachniss, F. Schneider, Coordinated multi-robot exploration, IEEE Transactions on Robotics 21 (2005) 376–386.
[17] A. Hayes, A. Martinoli, R. Goodman, Swarm robotic odor localization, in: Proc. of the IEEE Conf. on Intelligent Robots and Systems, Maui, USA, 2001, pp. 1073–1078.
[18] V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press, Cambridge, MA, 1984.
[19] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, A. Martinoli, SwisTrack – a flexible open source tracking software for multi-agent systems, in: Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS 2008), 2008, pp. 4004–4010.
[20] J. Pugh, X. Reamy, C. Favre, R. Falconi, A. Martinoli, A Fast on-board relative positioning module for multi-robot systems, in: IEEE/ASME Transactions on Mechatronics, Focused Section on Mechatronics in Multi Robot Systems, 2009, in press, http://infoscience.epfl.ch/record/131138.
[21] C. Buiu (Ed.), Roboţi cognitivi – Concepte, Arhitecturi, Aplicaţii (Cognitive robots – Concepts, Architectures, Applications), Universitară, Bucureşti, 2008.

Cristian I. Vasile was born in December 1986. He graduated the Faculty of Automatic Control and Computers of the POLITEHNICA University of Bucharest, Romania in 2009. He is a master student in the program Intelligent Control Systems at the Department of Automatic Control and Systems Engineering of the same University. He is also a teaching assistant at the Laboratory of Natural Computing and Robotics of the same department. His primary research interests are collaborative robotics, swarm intelligence, human–swarm interfaces and distributed algorithms.

Cătălin Buiu was born in June 1968. He graduated the Faculty of Automatic Control and Computers of the POLITEHNICA University of Bucharest, Romania in 1992 and obtained the PhD in control engineering at the same Faculty in 1997. He is now a Professor of Cognitive Robotics and Modelling of Biological Processes and Head of the Laboratory of Natural Computing and Robotics at the Department of Automatic Control and Systems Engineering of the same University. His active research areas include intelligent control, cognitive robotics, modelling of biological processes, and educational technologies. He has authored or co-authored 9 books and more than 60 journal and conference papers. He is an IEEE member, member of the IEEE Technical Committee on Safety, Security, and Rescue Robotics, Member of the IFAC Technical Committee on Intelligent Autonomous Vehicles.