10th IFAC Symposium on Robot Control
International Federation of Automatic Control
September 5-7, 2012. Dubrovnik, Croatia

A Navigation Strategy for Multi-Robot Systems Based on Particle Swarm Optimization Techniques

Raffaele Grandi, Riccardo Falconi, Claudio Melchiorri

e-mail: {raffaele.grandi, riccardo.falconi, claudio.melchiorri}@unibo.it

Abstract: In this paper, a novel strategy is presented aiming at controlling a group of mobile robots flocking through an unknown environment. The proposed control strategy is based on a modified version of the Particle Swarm Optimization (PSO) algorithm, and has been extensively validated by means of numerical simulations considering complex maze–like environments and groups of robots with different numbers of units.

Keywords: Particle Swarm Optimization, Multi–Robot System, Obstacle Avoidance

1. INTRODUCTION

In this paper, the design of a distributed control algorithm able to drive a group of mobile robots through an unknown environment from a starting area to a final one while avoiding obstacles is considered. As well known, such a problem has been widely faced in the literature on the basis of different classes of algorithms (Siegwart and Nourbakhsh, 2004). Among these, many popular techniques exploit the concept of virtual potential fields where, as described e.g. in (Antich and Ortiz, 2005), an *attractive* potential field is associated to the target area while *repulsive* potential fields are associated to the obstacles sensed in the environment by the robots. The result is that the environment is perceived by each robot as a *landscape* created by the combination of both static and time-varying factors. In this landscape, *valleys* and *peaks* represent the *global* attracting and repulsive zones respectively. Despite its versatility, this approach has many drawbacks. First of all, the robots have to gather a huge amount of information regarding obstacles, i.e. robots have to exchange many information to coordinate and to reach the target area. Moreover, a typical problem that may arise is the presence of local minima, i.e. areas in which robots are in deadlock situations. This problem has been solved by introducing a new class of *global* potential fields, called *social potentials* (Gayle et al., 2009), that can ensure the convergence of the swarm to its final destination.

Another relevant aspect in the coordination of multiple robots is the definition of the robot-to-robot interactions. As a matter of fact, as pointed out also in (Balch and Arkin, 1994), by changing the communication topology it is possible to improve the amount of information that each robot can exploit in order to achieve a predefined goal.

In this paper, these problems are solved by using a very limited exchange of data, a dynamic fitness function, and a free communication topology. The proposed approach for the control of the swarm of robots adopts a meta-heuristic algorithm based on the Particle Swarm Optimization (PSO). As a matter of fact, this technique has some interesting features that can be exploited for the guidance of a swarm of robots, such as its reliability, its intrinsic simplicity and the relatively small amount of information needed to create the desired emergent behaviors. The PSO approach was originally developed in 1995 (Kennedy and Eberhart, 1995) to study social interactions and it was initially inspired by birds flocking. In the literature, many modifications to the original PSO algorithm have been proposed in order to improve its efficiency, such as in (Shi and Eberhart, 1998) where a parameter called *inertia* is introduced to preserve motion direction of the particles. More recently, PSO algorithms have been applied to path planning for robots, as in (Masehian and Sedighizadeh, 2010), where a multi-objective cost function has been used. An exhaustive analysis of publications on particle swarm optimization can be found e.g. in (Poli, 2007).

This paper is organized as follows: in Sec. 2, background notions on PSO algorithms are provided while in Sec. 3 the modified version of the PSO is introduced. The results of simulations, used to validate the presented approach, are analyzed in Sec. 4, and conclusions and future work are reported in Sec. 5.

2. BACKGROUND ON PSO

In literature, PSO is usually addressed as an optimization algorithm, i.e. it is typically used to solve optimization problems defined in a $m-$dimensional space by a fitness function $f(\cdot)$ subject to predefined constraints whose value has to be minimized (or maximized). In the original PSO formulation, the optimal solution is computed by simulating a group of n particles that explore the search space of the problem in order to find the best fitness value. Each agent (or particle) moves in the solution search space with known position and velocity, and with the ability to communicate with other agents. In particular, PSO is based on a population $\mathcal{P}$ of n particles that represents

⋆ R. Grandi, R. Falconi and C. Melchiorri are with DEIS, Dip. di Elettronica, Informatica, Sistemistica, Università di Bologna. This work has been supported by CIRI-MAM within the 2007-2013 PORFERS Emilia Romagna Operational Program, and by MIUR with the PRIN 2008 research program.

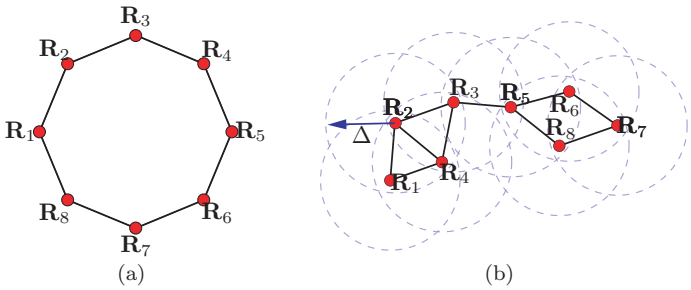


Fig. 1. Examples of a 2D communication graph: a *ring* graph, 1(a), and a Δ-disc graph, 1(b), in case of $\|\mathcal{P}\| = 8$ with $\Delta_i = \Delta$, $i = 1 \dots 8$.

a set of possible solutions of the given m-dimensional problem, i.e. $\|\mathcal{P}\| = n$ where the operator $\|\cdot\|$ computes the cardinality of a given set. The position and the velocity of the i-th particle at the k-th iteration of the algorithm are identified by the m-dimensional vectors

$$\begin{aligned} \boldsymbol{p}_i(k) &= [p_{i,1}(k) \dots p_{i,m}(k)]^T \\ \boldsymbol{v}_i(k) &= [v_{i,1}(k) \dots v_{i,m}(k)]^T \end{aligned} \qquad i = 1 \dots n$$

When the PSO algorithm is initialized, a random position $\boldsymbol{p}_i(0)$ and a starting random velocity $\boldsymbol{v}_i(0)$ are assigned to each particle. At each iteration of the PSO algorithm, a set of candidate solutions is optimized by the particles, moving through the search space toward better values of the fitness function $f(\cdot)$. Each particle of the initial swarm knows the value of the fitness function that corresponds to its current position in the m-dimensional search space and is able to *remember* data from previous iterations. In particular, each particle is able to remember the position where it has achieved the best value of the fitness function, namely $\boldsymbol{p}_i^*$ (called *local best*). This value can be exploited by neighbors to change their behavior. In fact, by assuming the possibility of a global communication between particles, each of them can gather the positions $\boldsymbol{p}_j^*$ of the other teammates where they have detected their best fitness value, and therefore the best value $\boldsymbol{p}_i^+$ among all can be defined (the *global best*). The propagation of $\boldsymbol{p}_i^+$ within the swarm depends on the communication topology, thus it may happen that non-communicating particles have temporarily different values. Moreover, each agent can use the best current fitness value among its neighbors $\boldsymbol{p}_i^\times$ (called *neighborhood best*), as another term to define its behavior. Fig. 2 clarifies the concept.

In conclusion, at each iteration of the algorithm, the behavior of each particle is defined by a proper function of three factors: $\boldsymbol{p}_i^*$ the local best, $\boldsymbol{p}_i^+$ the global best and $\boldsymbol{p}_i^\times$ the neighborhood best.

From these considerations, it follows that the communication topology, chosen to route information among particles, is an important feature able to drastically change the behavior of the whole swarm and, therefore, it has to be carefully chosen. As an example, in case of static communication topology (Jadbabaie et al., 2002), the contribution of $\boldsymbol{p}_i^+$ generates a quite *static* behavior, and therefore it is not guaranteed that the search space is fully and properly explored. As a matter of fact, in this case the movements of each particle must be somehow limited in order to remain connected to the neighbors, limiting therefore the search

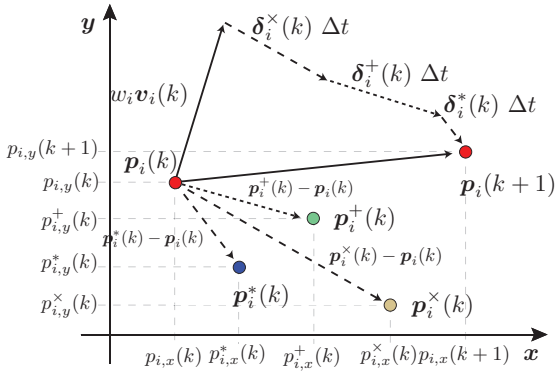


Fig. 2. Example of the original PSO algorithm applied to particles moving in a 2D environment.

possibilities. It follows that this approach could drive the system e.g. to a local minimum and thus to a local optimal solution. Furthermore, if the chosen static communication topology has not enough connections, each agent could even have a different value for $\boldsymbol{p}_i^+$ since the data could not be updated properly. In this case, the swarm could eventually reach dispersed configurations. The ring graph depicted in Fig. 1(a) is an example of a static communication topology.

In order to avoid these problems, we assume a dynamic communication topology that depends on the inter-particles distances. Each particle can move without constraints in the search space and communicate with any other neighbor within its communication range. In this way we obtain a better exploration of the search space and a faster update of the data related to the environment. If a particle has better data on the environment with respect to its neighbors, these automatically tends to aggregate to reach the best position available.

Communication graphs whose topology depends on the inter-agents distance are usually addressed as Δ-disc graphs, a topology typically applied to 2-D networks (Bullo et al., 2009). With abuse of notation we address distance-based communication topology as Δ-disc graphs also in case of agents defined in $m > 2$ dimensions. An example of a 2D Δ-disc graph dynamic topology is shown in Fig. 1(b). To this purpose, we define the neighbor set of the i-th particle as the set of all the particles whose distance from $\boldsymbol{p}_i$ is smaller than a predefined threshold Δ_i. The *neighbors subset* of the i-th particle is defined as

$$\mathcal{N}_i = \left\{ j \in \mathcal{P} : |\boldsymbol{p}_i - \boldsymbol{p}_j| \leq \Delta_i \right\} \tag{1}$$

where $|\cdot|$ computes the length of a vector. At each iteration of the PSO algorithm, the state of each particles is updated as follows

$$\boldsymbol{v}_i(k+1) = \xi_0(w_i\boldsymbol{v}_i(k) + \boldsymbol{\delta}_i^*(k) + \boldsymbol{\delta}_i^\times(k) + \boldsymbol{\delta}_i^+(k)) \tag{2}$$

$$\boldsymbol{p}_i(k+1) = \boldsymbol{p}_i(k) + \boldsymbol{v}_i(k+1)\ \Delta t \tag{3}$$

where w_i is a scalar constant that represents the *inertia* of the particle, ξ_0 is a variable parameter called *constriction factor* (Eberhart and Shi, 2000) introduced to avoid dispersion of the particles. The term $w_i\boldsymbol{v}_i(k)$ is addressed in the literature as *persistence*, and represents the tendency of a particle to preserve its motion direction, while Δt is the simulation time step. The terms $\boldsymbol{\delta}_i^*(k)$, $\boldsymbol{\delta}_i^\times(k)$ and $\boldsymbol{\delta}_i^+(k)$ represent *historical* and *social* contributions and are defined as

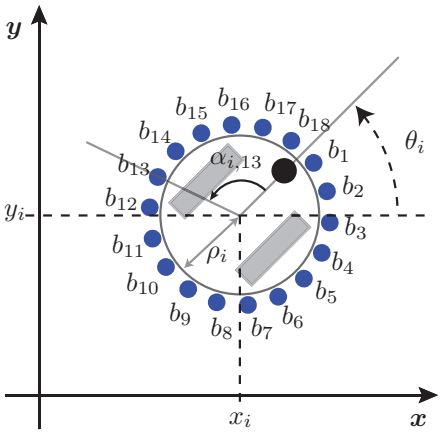


Fig. 3. Kinematic model of the robots: the black dot identifies the robot's front, $[b_1 - b_{18}]$ are the bumpers, and $\alpha_{i,j}$ is the angle of the j-th bumper.

$$\boldsymbol{\delta}_i^*(k) = \phi \cdot r_i^*(k)\ \frac{\boldsymbol{p}_i^*(k) - \boldsymbol{p}_i(k)}{\Delta t}$$
$$\boldsymbol{\delta}_i^\times(k) = \phi \cdot r_i^\times(k)\ \frac{\boldsymbol{p}_i^\times(k) - \boldsymbol{p}_i(k)}{\Delta t}$$
$$\boldsymbol{\delta}_i^+(k) = \phi \cdot r_i^+(k)\ \frac{\boldsymbol{p}_i^+(k) - \boldsymbol{p}_i(k)}{\Delta t}$$

The first term represents the speed contribution given by the *best individual* value of the particle, while the second and the third ones represent the *neighborhood* and the *swarm* contribution respectively. Each term is a vector that attracts each particle towards the corresponding point into the search space. The parameters $r_i^*(k)$, $r_i^\times(k)$ and $r_i^+(k)$ are usually selected as uniform random numbers in $[0, 1]$, computed at each iteration of the algorithm to give a certain degree of variation to the particle's behavior. The parameter ϕ is used to modulate the maximum influence of the randomic behaviour of the particles. Because of its importance in tuning PSO parameters, the ϕ value has been defined in the literature by using many methods, see e.g. (Clerc and Kennedy, 2002; Trelea, 2003), often with empirical approach as in our case: after many simulations and considering in particular the capability of the swarm to overcome obstacles without 'loosing' any particle, we have set $\phi = 2$, see Sect. 4.

3. IMPROVING THE PSO ALGORITHM

The algorithm presented in this paper has been developed in order to drive a swarm of robots $R_i, i = 1, \ldots, n$ moving in unknown planar environments. To this purpose, we consider a group of differential-wheeled robots (see Fig. 3) whose kinematics is given by

$$\begin{aligned} x_i(k+1) &= x_i(k) + u_i(k)\cos(\theta_i(k)) \\ y_i(k+1) &= y_i(k) + u_i(k)\sin(\theta_i(k)) \\ \theta_i(k+1) &= \theta_i(k) + \omega_i(k)\Delta t \end{aligned} \tag{4}$$

where $u_i(k), \omega_i(k)$ are respectively the forward and the steering control velocities of the robot during the time step $[k,\ k+1]$. Two problems to be solved in order to apply the PSO algorithm to the navigation of a robotic swarm are how to match each particle to a single robot, and the search space to the environment surrounding the swarm.

3.1 Matching PSO agents with physical robots

As the PSO algorithm considers each agent as a single integrator in x, y coordinates (i.e. it calculates a velocity vector $\boldsymbol{v}_i(k) \in \mathbb{R}^2$ without taking into account the robot kinematics), the robot exploits the data $\boldsymbol{p}_i(k+1) = [p_{i,x}(k+1),\ p_{i,y}(k+1)]^T$ given by the PSO as a target for its movements. Since we are dealing with robots moving on a plane and it is $\boldsymbol{p}_i(k) = [p_{i,x}(k),\ p_{i,y}(k)]^T$ for R_i, the control input $u_i(k+1)$, $\omega_i(k+1)$ are computed as

$$u_i(k+1) = K_u |\boldsymbol{v}_i(k+1)\ \Delta t| \tag{5}$$
$$\omega_i(k+1) = K_\omega\ \mathrm{atan2}({}^y\delta_{p_i}, {}^x\delta_{p_i})\ \Delta t \tag{6}$$

where ${}^y\delta_{p_i} = p_{i,y}(k+1) - p_{i,y}(k)$, ${}^x\delta_{p_i} = p_{i,x}(k+1) - p_{i,x}(k))$ and K_u, K_ω are two proper constants. In particular, we assume that the velocities of the left and right motor (ν_L and ν_R respectively) are saturated at ν_{max}, thus $u_{max} = r\nu_{max}$ and $\omega_{max} = 2u_{max}/d$ where r is the radius of the wheels and d is the distance between them. We set $K_\omega \gg K_u$ in order to favor the *turning–on–the–spot* behavior of each robot. In practice, when the control actions are calculated, the robot starts turning on the spot and it moves slowly forward, then it accelerates only when it is almost aligned with the desired direction and it stops when it reaches the target, or detects a collision. Moreover, we have supposed that each robot knows its own position with respect to a global inertial frame.

3.2 Matching the search space to the environment

This problem has been solved by defining a proper fitness function embedding not only the optimization problem but also the obstacle avoidance. The fitness function can be described as the distance between two points $\boldsymbol{a}_g$ and $\boldsymbol{a}_i$ belonging to a 3D space called *fitness map*. This map is defined by adding a third component z to the standard x, y coordinates of the robot in the arena. In this manner, the position $\boldsymbol{p}_i = (x_i, y_i)$ of the i-th robot in the arena has a corresponding point $\boldsymbol{a}_i = (x_i, y_i, z_i)$ in the map. The coordinate z_i represents, as described in the following, the 'cost' of point $\boldsymbol{p}_i$ in terms of robot perception of the environment. In fact, the coordinate z_i is computed by a function $\mathcal{Z}(\boldsymbol{p}_i, k, \mathcal{C}_i^k) \geq 0$ (see eq. (10)) that, roughly speaking, is a combination of Gaussian functions related to the distance of the robot from known obstacles.

In the presented PSO–based coordination algorithm, each robot of the team, flocking toward a common predefined global target (or *goal*), positioned in $\boldsymbol{p}_g = (x_g, y_g)$ and corresponding to $\boldsymbol{a}_g = (x_g, y_g, 0)$ in the fitness map, evaluates the following fitness function

$$f(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \gamma_i\ |\boldsymbol{a}_g - \boldsymbol{a}_i(k))| \tag{7}$$

where the parameter γ_i can be tuned to modify the relevance given by each robot to the target point. The z component of $\boldsymbol{a}_g$ is always zero, so that the target point $\boldsymbol{p}_g$ defines the minimum value of the fitness function.

Another important aspect has to be considered in order to apply the PSO approach to a robot swarm: the standard PSO algorithm does not consider agents with physical properties or constraints. Therefore, with the purpose of defining an algorithm that allows agents (i.e. robots) to

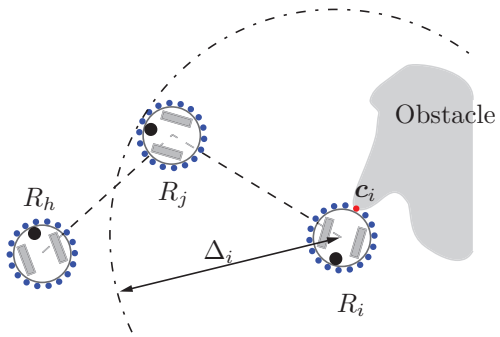


Fig. 4. Example of broadcasted data for a collision. The dash-dotted circumference shows the communication area of robot R_i while dashed lines represent the inter-robot communication network.

navigate in an unknown environment while avoiding obstacles, passing through narrow passages and preserving connectivity, we propose two modifications to the PSO algorithm. These modifications concern the *obstacle avoidance* capability and a modified social interaction between neighbor particles, named *neighbors' aggregation vector*.

3.3 Obstacles and Local Minima Avoidance

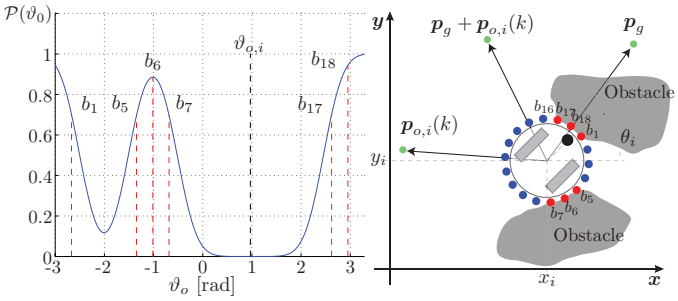
Since PSO was originally defined as an algorithm to drive particles in a *virtual* m-dimensional environment, the case of agents with physical properties was not considered. In our case, as we are considering real robots, we must take into account that they have a not-null radius (see Fig. 3) and that they can possibly collide with obstacles and teammates. In order to deal with these constraints, we consider each robot equipped with eighteen bumpers evenly spaced on its perimeter. Moreover, in case a collision is detected, its position is broadcasted to teammates. If we suppose that the robot R_i detects a collision, as depicted in Fig. 4, the position of the contact point $\boldsymbol{c}_{i,h} = [x_{c,i},\ y_{c,i}]^T$ is computed as

$$\begin{aligned} x_{c,i} &= x_i + \rho_i \cos(\alpha_{i,m} + \theta_i) \\ y_{c,i} &= y_i + \rho_i \sin(\alpha_{i,m} + \theta_i) \end{aligned} \tag{8}$$

where ρ_i is the radius of the i-th robot, $\alpha_{i,m}$ is the angle of the m-th bumper (see also Fig. 3), and $h \geq 0$ is a label that uniquely identifies the collision point. Moreover, the time step k_h^0 at which a collision h is detected is saved. Let us remark that, as long as robots do not have any knowledge of the environment, different $\boldsymbol{c}_{i,h}$ could correspond to the same obstacle. More formally, considering for simplicity the 2D case, let us define $\mathcal{O} \in \mathbb{R}^2$ and $\mathcal{R}_i \in \mathbb{R}^2$ as the set of points of the plane that are part of an obstacle and of a robot, respectively. We can then define the set of collisions detected by R_i up to the k-th step as

$$\mathcal{C}_i^k := \left\{ \boldsymbol{c}_{i,h} \in \mathbb{R}^2 : \boldsymbol{c}_{i,h} = (\mathcal{O} \cap \mathcal{R}_i) \wedge (k - k_h^0) \leq \tilde{k} \right\} \quad h \in [1..n_i(k)] \tag{9}$$

where $\tilde{k}$ represents the *lifetime* of the detected collisions and $n_i(k)$ is the total number of collisions detected by the i-robot from the beginning of the simulation until the time step k. Namely, (9) states that a point of the plane can be considered as a collision point for the i-th robot at time k if there is at least an intersection between the set of points of the robot and the set of points of the obstacles. In order



(a) Probability curve related to (13) (b) Escape windows technique

Fig. 5. Example of escape window computed by considering obstacles detected by bumpers $\{b_{17},\ b_{18},\ b_1\}$ and $\{b_5,\ b_6,\ b_7\}$.

to include obstacle avoidance in the fitness function, the function $\mathcal{Z}$ used to compute the fitness map is defined as

$$\mathcal{Z}(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \beta \sum_{\boldsymbol{c}_{i,h} \in \mathcal{C}_i^k} \Theta(k, k_h^0)\ g(|\boldsymbol{p}_i - \boldsymbol{c}_{i,h}|, \sigma_q) \tag{10}$$

where β is a proper parameter, $\Theta(k, k_0^h)$ is a *forgetting factor* for detected collisions, $g(\cdot)$ is a Gaussian function. The forgetting factor $\Theta(\cdot)$ is introduced in order to decrease in time the effects of an obstacle since, very likely, it is also physically distant being the robot moving in the environment. For $k \geq k_h^0$, it is defined as

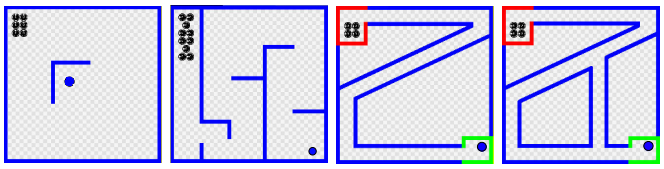
$$\Theta(k, k_h^0) = \begin{cases} \dfrac{\lambda}{k - k_h^0 + 1} & \text{if } (k - k_h^0) \leq \tilde{k} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $\tilde{k}$ is defined in eq. (9). In our case we set $\lambda = 1$ and $\tilde{k} = 500$. The term $g(|\boldsymbol{p}_i - \boldsymbol{q}_j|, \sigma_q)$ is a normal Gaussian distribution defined as

$$g(|\boldsymbol{p}_i - \boldsymbol{q}_j|, \sigma_q) = \frac{exp\{-|\boldsymbol{p}_i - \boldsymbol{q}_j|^2/(2\sigma_q^2)\}}{\sigma_q\sqrt{2\pi}} \tag{12}$$

where the variance σ_q is used to shape the influence of a collision on its surrounding area. In this way, the function (10) defines a map shaped with 'peaks' and 'valleys' generated by collisions. When two robots are within the communication range, the relative collision sets $\mathcal{C}_i^k$ are exchanged, and each robot merges its information with those of the other one. In this manner, by sharing information about detected collisions, a *collective memory* is introduced able to drive agents faster towards the target while avoiding obstacles.

On the other hand, the obstacle avoidance technique defined by (10) is not sufficiently fast for avoiding local minima in real time. Indeed, the fitness function takes into account obstacles, but the detected collisions do not affect directly the robots motion and therefore a considerable amount of time could be necessary to move a robot away from critical zones. In order to solve this problem, a technique inspired by the dynamic window approach (Fox et al., 1997) has been applied. In particular, as depicted in Fig. 5(a) and similarly to what described in (Falconi and Melchiorri, 2008), we have assumed that each robot can project around itself a probability curve that depends on the obstacles detected by on board sensors. Thus, each robot computes the relative angle $\vartheta_{o,i}$ where the probability of colliding with an obstacle is minimal



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3 (d) Scenario 4

Fig. 6. Four scenarios used to test the PSO-NAV algorithm. The target area is depicted as a blue point.

$$\vartheta_{o,i} = \min(\varphi_{o,i}), \qquad \varphi_{o,i} = \sum_{h=1}^{N_b} b_h \; g(\beta, \sigma_{h,i}, \alpha_{h,i}) \tag{13}$$

where $\beta \in [-\pi \dots \pi]$, $b_h = 1$ if the h-th bumper detects a collision and $b_h = 0$ otherwise, and $g(\cdot)$ is the normal Gaussian distribution centered on $\mu = \alpha_{h,i}$ defined by

$$g(\beta, \sigma_{h,i}, \alpha_{h,i}) = \frac{exp\{-(\beta - \alpha_{h,i})^2/(2\sigma_{h,i}^2)\}}{\sigma_{h,i}\sqrt{2\pi}} \tag{14}$$

where the variance $\sigma_{h,i}$ can be used to define how each bumper affects the robot's perceptions. Once the value of $\vartheta_{o,i}$ in (13) is determined, a *virtual target* $\boldsymbol{a}_{o,i}(k) = [x_{o,i}(k),\ y_{o,i}(k),\ 0]^T$ is temporarily defined, with

$$\begin{aligned} x_{o,i}(k) &= x_i(k) + \varrho_i \cos(\vartheta_{o,i}(k)) \\ y_{o,i}(k) &= y_i(k) + \varrho_i \sin(\vartheta_{o,i}(k)) \end{aligned} \tag{15}$$

where $\varrho_i = |\boldsymbol{p}_g - \boldsymbol{p}_i(k)|$ is the distance between actual position of the robot and the target area's center. The fitness function (7) is then modified as (see Fig. 5(b))

$$f(\boldsymbol{p}_i, k, \mathcal{C}_i^k) = \gamma_i |(\boldsymbol{a}_g + \boldsymbol{a}_{o,i}(k)) - \boldsymbol{a}_i(k)|$$

3.4 Neighbors Aggregation Vector

The last main modification introduced respect to the standard PSO algorithm is the *Neighbors Aggregation Vector* (NAV), indicated as $\boldsymbol{\delta}_i^{\otimes}(k)$. This term replaces $\boldsymbol{\delta}_i^{\times}(k)$ in (2) and is inspired by the idea that *if the teammates move in a certain direction, maybe that direction is the right one* (Reynolds, 1987). The term $\boldsymbol{\delta}_i^{\otimes}(k)$ modifies the robots behavior by taking into account the best solution found by their neighbors, and is defined as

$$\boldsymbol{\delta}_i^{\otimes}(k) = \phi \cdot r_i^{\otimes}(k) \sum_{\boldsymbol{p}_j \in \mathcal{N}_i} \frac{\boldsymbol{p}_j^{\times}(k) - \boldsymbol{p}_i(k)}{\Delta t} \tag{16}$$

where $r_i^{\otimes}(k) \in [0,\ 1]$ is a random number updated at each iteration and ϕ a proper tuning value. In a nutshell, $\boldsymbol{\delta}_i^{\otimes}(k)$ represents a randomly weighted barycenter of the position where the neighbors of R_i have detected their *neighbor best* at the k-th time step.

3.5 Dynamic constriction factor

Moreover, the parameter ξ_0 in (2) is replaced by a dynamic *constriction factor* that depends, for each robot, on the mean value of the number of collisions memorized by the robot itself in the last 3 time steps. If we assume $\mathcal{D}_i^j \subset \mathcal{C}_i^k$ where $j \in [k-2, .., k]$ the constriction factor is defined as

$$\xi_i = \xi_0 (1 + \frac{1}{3} \sum_{j=k-2}^{k} \|\mathcal{D}_i^j\|) \tag{17}$$

In practice, the term ξ_i in (17) is a sort of collision trading that increases if a high number of collisions is detected, thus changing the robot behavior and pushing it away from critical zones. It follows that in environments with few obstacles robots flock clustered, while in case of many constraints robots are forced to move sparsely.

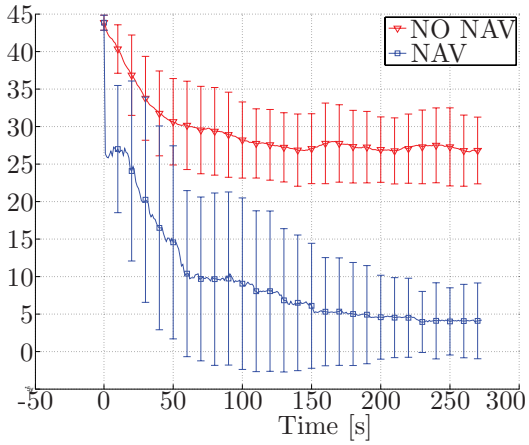


Fig. 7. MSE and STDEV of distance error between each robot's position and the goal both with and without NAV. Data gathered over 20 simulations on Scen.3.

To conclude, with the proposed modifications, the standard PSO algorithm defined in (2)–(3) is rewritten as

$$\boldsymbol{v}_i(k+1) = \xi_i \left(w_i \boldsymbol{v}_i(k) + \boldsymbol{\delta}_i^{*}(k) + \boldsymbol{\delta}_i^{\otimes}(k) + \boldsymbol{\delta}_i^{+}(k) \right) \tag{18}$$

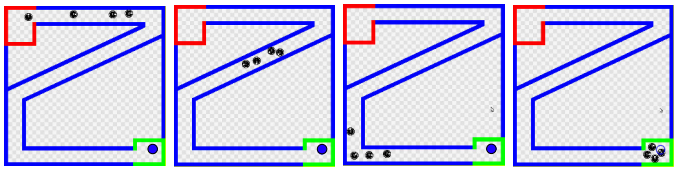
$$\boldsymbol{p}_i(k+1) = \boldsymbol{p}_i(k) + \boldsymbol{v}_i(k+1)\ \Delta t \tag{19}$$

4. SIMULATIONS AND COMMENTS

The presented PSO algorithm has been widely tested in our simulated environment JUSE (Grandi et al., 2011). In particular, we have considered a 2×2 $[m]$ virtual arena where the starting and goal areas are placed in non-trivial positions, i.e. with at least an obstacle between them. Robots have been modeled as real differential-wheeled robots with radius 0.05 $[m]$, maximum speed $u_{max} = 0.35$ $[m/s]$, $w_{max} = 7$ $[rad/s]$ and communication range $\Delta = 0.4$ [m]. Obstacles have been modeled as rectangles of arbitrary length and width of 0.05 $[m]$, and a sample time of $\Delta t = 0.01[s]$ has been selected for the controller.

To enlighten the algorithm's performances, we have considered different scenarios, see Fig. 6. The starting area has been placed in the upper-left corner while the goal area is at the opposite corner, except in the case of Fig. 6(a), where the target is in the center of the arena.

In order to gather statistical results about the performance of the proposed PSO algorithm in driving robot swarms, we have performed 20 simulations, executed in the arena depicted in Fig. 6(c) using both standard PSO and PSO–NAV. Results are shown in Fig. 7 where the mean square error (MSE) and standard deviation (STDEV) for the data set provided by a team of 5 robots are reported. Note that the distance error between each robot and final goal area does not converge to zero if the standard PSO is used. Vice versa, by using the NAV, the system converges to a value close to zero. Due to robots' physical dimensions, the value of the MSE cannot be null because robots aggregate around the final point. STDEV demonstrates that despite the random parameters introduced in (2)–(3) and (16), the algorithm is able to adapt and to drive the robots to the



(a) t = 5 s (b) t = 50 s (c) t = 170 s (d) t = 270 s

Fig. 8. Scenes of a simulation with 4 robots in Scenario 3.

goal area. Moreover, let us remark that the Z–shaped maze in Fig. 8 would not allow to either classic potential field algorithms or original PSO algorithm to reach the target area since, before moving toward the final location, robots have to move temporarily away from it. For this reason, PSO-NAV increases the swarm dispersion to overcome local minima as shown in fig. 7 at about T=100 [s]. This it is related to the choice $\phi = 2$: lower values make the swarm too compact and unable to overcome obstacles (and local minima), while higher values makes the swarm to free and thus not able to converge after their exceeded. Moreover, notice that the considered scenarios do not have the same complexity. Scenario 3 and 2 are the most difficult. With 2, the obstacle located in the center of the arena decreases the NAV contribution and the algorithm's performances are close to classical PSO.Fig. 8 shows a simulation involving 4 robots in Scenario 3. We have considered swarms with a number of robots ranging from 3 to 12. From the case studies, it results that swarms with less than 3 robots are usually not able to perform the given tasks in complex scenarios, while more than 10-12 robots create too many collisions and the algorithm fails to maintain the group compact. In this case, the swarm breaks up into small groups that reach the target autonomously. Probably, with larger arenas, the number of robots necessary to have a fully functional swarm would be greater but in our environment, the simulations suggest that 4-6 robots are the best trade-off. Finally, we want to remark that only a very limited set of data is exchanged by robots during rendez vous: the three vectors $\boldsymbol{p}_i^+$, $\boldsymbol{p}_i^\times$, $\boldsymbol{p}_i^*$, and the set $\mathcal{C}_i^k$. More complex simulations are available at `http://www.youtube.com/watch?v=hEiN_rNB7pI`.

5. CONCLUSIONS AND FUTURE WORKS

In this paper, a novel version of the classic Particle Swarm Optimization algorithm has been proposed in order to drive a group of mobile robots in unknown environments from a starting point to a final one. In particular, the PSO algorithm has been modified in order to consider robots' physical constraints, i.e. their dimensions and their interaction with the environment. Mobile robots have been modeled as differential-wheeled robots, able to access their own position, to broadcast data to neighbors and to sense the environment by using bumper sensors. The performances of this modified version of the PSO algorithm have been validated using data collected in several simulations, where different groups of robots have been simulated in many maze-like environments. Future works will include experiments on real robots such as those described in (Grandi et al., 2011). Moreover, the application of this algorithm to more complex systems as groups of heterogeneous robots (e.g. group of robots including both ground and aerial vehicles) and in more complex environments (e.g underwater) will be considered.

REFERENCES

Antich, J. and Ortiz, A. (2005). Extending the potential fields approach to avoid trapping situations. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, 1386 – 1391.

Balch, T. and Arkin, R.C. (1994). Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1, 27–52. URL `http://dx.doi.org/10.1007/BF00735341`. 10.1007/BF00735341.

Bullo, F., Cortes, J., and Martinez, S. (2009). *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms (Princeton Series in Applied Mathematics)*. Princeton University Press.

Clerc, M. and Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE*, 58–73.

Eberhart, R. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proc. 2000 Congress on*, 84 –88 vol.1.

Falconi, R. and Melchiorri, C. (2008). A decentralized control algorithm for swarm behavior and obstacle avoidance in unknown environments. *Proc. 2nd IFAC Work. Navigation, Guidance Control of Underwater Vehicles*.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 23–33.

Gayle, R., Moss, W., Lin, M., and Manocha, D. (2009). Multi-robot coordination using generalized social potential fields. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, 106 –113.

Grandi, R., Falconi, R., and Melchiorri, C. (2011). Unibot remote laboratory: A scalable web-based set-up for education and experimental activities in robotics. *Proc. 18th IFAC World Congress*, 8521–8526.

Jadbabaie, A., Lin, J., and Morse, A. (2002). Coordination of groups of mobile autonomous agents using nearest neighbor rules. In *Decision and Control, 2002, Proc. 41st IEEE Conference on*, 2953 – 2958 vol.3.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Neural Networks, 1995. Proc.*

Masehian, E. and Sedighizadeh, D. (2010). A multi-objective pso-based algorithm for robot path planning. *2010 IEEE International Conference on Industrial Technology*, 465–470.

Poli, R. (2007). An analysis of publications on particle swarm optimization applications. *Journal of Artificial Evolution and Applications*, 1–57.

Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, 25–34.

Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *In Proc. IEEE Int. Conf. on Evolutionary Computation*.

Siegwart, R. and Nourbakhsh, I.R. (2004). *Introduction to Autonomous Mobile Robots*. The MIT Press.

Trelea, I. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85, 317–325.