Applied Soft Computing 25 (2014) 426–434

Contents lists available at ScienceDirect

Applied Soft Computing

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

Swarm intelligent approaches to auto-localization of nodes in static UWB networks

Stefania Monica*, Gianluigi Ferrari

Wireless Ad-hoc and Sensor Networks Laboratory (WASNLab), Department of Information Engineering, University of Parma, I-43124 Parma, Italy

ARTICLE INFO

Article history:
Received 30 July 2013
Received in revised form 26 June 2014
Accepted 26 July 2014
Available online 11 September 2014

Keywords:
Auto-localization
Particle swarm optimization (PSO)
Hybrid PSO
Two-stage maximum-likelihood (TMSL) method
Plane intersection (PI) method

ABSTRACT

In this paper, we address the problem of localizing sensor nodes in a static network, given that the positions of a few of them (denoted as "beacons") are a priori known. We refer to this problem as "auto-localization." Three localization techniques are considered: the two-stage maximum-likelihood (TSML) method; the plane intersection (PI) method; and the particle swarm optimization (PSO) algorithm. While the first two techniques come from the communication-theoretic "world," the last one comes from the soft computing "world." The performance of the considered localization techniques is investigated, in a comparative way, taking into account (i) the number of beacons and (ii) the distances between beacons and nodes. Since our simulation results show that a PSO-based approach allows obtaining more accurate position estimates, in the second part of the paper we focus on this technique proposing a novel hybrid version of the PSO algorithm with improved performance. In particular, we investigate, for various population sizes, the number of iterations which are needed to achieve a given error tolerance. According to our simulation results, the hybrid PSO algorithm guarantees faster convergence at a reduced computational complexity, making it attractive for dynamic localization. In more general terms, our results show that the application of soft computing techniques to communication-theoretic problems leads to interesting research perspectives.

1. Introduction

The problem of locating sources in an indoor environment can be addressed by using wireless sensor networks, which combine low-to-medium data rate communications with positioning capabilities. As a matter of fact, radio signal exchanges between pairs of nodes allow estimating the distance between them, by extracting from these signals some physical quantities, such as the received signal strength, the angle of arrival, or the time of flight [1]. In this way, under the assumption of perfect knowledge of the exact positions of a sufficiently large number of nodes, a new one can be located by measuring and properly processing its distances from a few nodes with known positions.

Since wireless communications are affected by errors, the obtained positions will not be precise. The main sources of errors in indoor communications are phenomena such as non-line-of-sight propagation, multipath, and multiple access interference. In order to obtain more accurate position estimates by reducing the impact of these sources of errors, we consider ultra wide band (UWB) signaling between nodes, because (i) UWB signals' large bandwidth allows penetrating through obstacles and resolving multipath components and (ii) their high time resolution improves the ranging capability [2]. A time difference of arrival (TDOA) approach, based on the differences between the arrival times of signals, is used to estimate the distances between pairs of nodes.

Indoor node localization has applications in many fields, such as, for example, tracking of people in high security areas and of patients in hospitals. Another scenario of interest involves the location of "tags" in large warehouses, where it can be assumed that many static nodes are placed on the ceiling: the knowledge of their positions would allow locating and tracking moving people and/or vehicles. In order to guarantee accurate position estimates of a moving target in every accessible point inside a large warehouse, a very large number of accurately positioned static nodes would be necessary. Moreover, if the geometry of the warehouse changes (e.g., for varying quantities of stored goods), the nodes might be replaced and/or new ones might be added. Accurate placement of a huge number of nodes would thus be very demanding also from an economic point of view. For this reason, it is interesting to focus on the auto-localization of the static nodes, assuming to know the exact positions of only a few of them, which will be denoted as "beacons." Then, the other nodes would "learn"

* Corresponding author. Tel.: +39 0521 905759.
E-mail address: stefania.monica@studenti.unipr.it (S. Monica).

http://dx.doi.org/10.1016/j.asoc.2014.07.025
1568-4946/

their positions starting from the beacons: eventually, all static nodes would self-localize and then cooperate to localize (possibly moving) tags. The number of beacons should be small, in order to reduce the cost of installation, but sufficiently large to guarantee reliable position estimates of other static nodes, which is necessary to properly use them to locate moving targets. In this work, we investigate self-localization in UWB networks.

Among the wide variety of location estimation techniques which have been proposed in the literature, it is worth recalling iterative methods, such as those based on Taylor series expansion [3] or the steepest-descent algorithm [4], graph-based methods [5], or methods based on metaheuristics [6]. To overcome some limitations of these methods, closed-form algorithms have been studied, such as the plane intersection (PI) method [7] and the Two-stage maximum-likelihood (TSML) method [8]. Both these methods deal with solving linear or non-linear systems of equations, whose data contain the coordinates of beacons and the range estimates between them and the current tag. In some particular cases (for instance, if the considered beacons lay on the same line), the matrices involved in the aforementioned algorithms might become ill-conditioned, thus leading to a wrong position estimate of the current tag. To avoid this kind of problems, it is possible to re-interpret the "geometric" localization problem as a minimization one. In this paper, by re-formulating the initial system of equations of the TSML method in terms of an optimization problem, we solve it through the use of particle swarm optimization (PSO). Direct comparison of the performance of a geometric method (i.e., the PI or the TSML methods) with that of the PSO algorithm shows that the latter can significantly outperform the former.

In [9], we compare the performance of the TSML method with that of the PSO algorithm in terms of the impact of the distances between beacons and nodes. In [10], we investigate the performance of the TSML method, the PI method, and the PSO algorithm was investigated, in a comparative way, for various numbers of beacons. In this paper, we extend the preliminary investigation of [9,10]. In particular, we consider a larger number of topologies, analyzing the performance of the TSML, PI, and PSO methods for various configurations in terms of number of beacons and distances between beacons and nodes. As predicted by the preliminary analysis in [9,10], our results show that the PSO method is to be preferred regardless of the network configuration (number of beacons and topology).

We then propose a novel improved version of the PSO algorithm, which extends the hybrid approach proposed in [11], and we compare its performance with that of the standard PSO algorithm. In particular, we investigate the number of iterations which are needed to achieve a given error tolerance in the position estimate. Our simulation results show that the proposed hybrid version of the PSO algorithm guarantees, at a significantly reduced computational complexity, the fastest convergence in all considered scenarios, making it very attractive from an applicative viewpoint.

This paper is organized as follows. In Section 2, the considered scenario is described. In Section 3, the positioning algorithms are described and simulation-based performance results are presented. In Section 4, a hybrid version of the PSO algorithm, with improved performance, is proposed. Section 5 concludes the paper.

2. Scenario

As anticipated in Section 1, all considered static nodes are supposed to lay, at the same height, on the ceiling of a large indoor environment. A sufficiently large (but still limited) number M of these nodes, denoted as "beacons", is used to estimate the position of each remaining node (with unknown position). The coordinates of each beacon are represented by bidimensional vectors $\underline{s}_i = [x_i, y_i]^T$, $\forall i \in \{1, \ldots, M\}$, and we denote as K_i the norm of vector $\underline{s}_i$, $\forall i \in \{1, \ldots, M\}$. All the other nodes are estimated one after another. The true position of the currently considered node is indicated as $\underline{u}_e = [x_e, y_e]^T$, while the obtained position estimate is denoted as $\hat{\underline{u}}_e = [\hat{x}_e, \hat{y}_e]^T$. Following this notation, the true and the estimated distances between the ith beacon and the considered node are, respectively:

$$r_i = \sqrt{(\underline{u}_e - \underline{s}_i)^T(\underline{u}_e - \underline{s}_i)}, \quad \hat{r}_i = \sqrt{(\hat{\underline{u}}_e - \underline{s}_i)^T(\hat{\underline{u}}_e - \underline{s}_i)}. \tag{1}$$

According to the results on UWB signaling presented in [12], we assume that the estimated distances can be expressed as follows:

$$\hat{r}_i = r_i + \nu_i$$

where $\nu_i = \varepsilon_i + b$; $\varepsilon_i \sim \mathcal{N}(0, \sigma_i^2)$; ε_i is independent from ε_j if $i \neq j$ (i, $j \in \{1, \ldots, M\}$); and b is the synchronization bias. Moreover, the standard deviation σ_i of the distance error estimate between the ith beacon and a generic node can be approximated as the following linear function of the distance between them:

$$\sigma_i = 0.01 r_i + 0.08. \tag{2}$$

The values of the coefficients in Eq. (2) are obtained in [12] by considering Channel Model 3 described in [13] and the energy

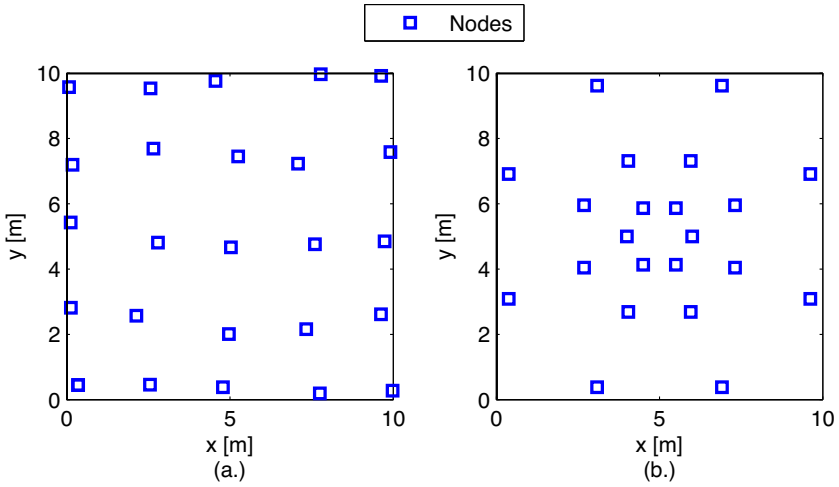

Fig. 1. Two possible configurations of nodes inside a square, with 10 m long side: (a) almost uniform norm distribution; (b) nodes laying on circumferences.

detection receiver presented in [14], which is composed by a band-pass filter followed by a square-law device and an integrator, in which the integration interval was set equal to $T_s = 1$ s. Therefore, the results presented in the following hold under these channel and receiver conditions.

As stated at the beginning of the current section, the reference scenario is a large industrial scenario, whose area may be of several hundreds of square meters. Assuming to divide the whole area into smaller squares, with 10 m-long sides, and that in each square the same number of beacons is considered, without loss of generality we only refer to a single square in the simulation-based performance analysis presented in the following. Two illustrative configurations of the distribution of the nodes inside a square are shown in Fig. 1(a) and (b).

3. Positioning techniques

In this section, we describe two classical geometrical approaches widely used to tackle localization problems, namely the TSML method and the PI method [7,8]. The initial system of equations of the TSML method is then rewritten as a minimization problem, which is solved by means of the PSO algorithm. Finally, the performance of the three approaches is compared, through simulations.

For the sake of simplicity, we preliminarily introduce the following notation:

$$\Delta_{1i} \triangleq r_i - r_1 \quad \text{and} \quad \hat{\Delta}_{1i} \triangleq \hat{r}_i - \hat{r}_1, \quad \forall i = 2, \ldots, M. \tag{3}$$

3.1. Two-stage maximum-likelihood method

Observing that $r_i^2 = (\Delta_{1i} + r_1)^2$, $\forall i \in \{2, \ldots, M\}$, the following TDOA non-linear system of equations can be derived [15]:

$$\underline{\underline{\hat{G}}}\,\underline{\hat{\phi}}_1 = \underline{\hat{h}} \tag{4}$$

where

$$\underline{\underline{\hat{G}}} = -\begin{pmatrix} x_2 - x_1 & y_2 - y_1 & \hat{\Delta}_{12} \\ x_3 - x_1 & y_3 - y_1 & \hat{\Delta}_{13} \\ \vdots & \vdots & \vdots \\ x_M - x_1 & y_M - y_1 & \hat{\Delta}_{1M} \end{pmatrix}, \quad \underline{\hat{h}} = \frac{1}{2}\begin{pmatrix} K_1^2 - K_2^2 + \hat{\Delta}_{12}^2 \\ \vdots \\ K_1^2 - K_M^2 + \hat{\Delta}_{1M}^2 \end{pmatrix} \tag{5}$$

and $\underline{\hat{\phi}}_1 = [\underline{\hat{u}}_e^T, \hat{r}_1]^T$.

The system (4) may look like a linear system in the three unknowns $\hat{x}_e$, $\hat{y}_e$ and $\hat{r}_1$, but it is not, since, by definition, $\hat{r}_1 = \sqrt{(\hat{x}_e - x_1)^2 + (\hat{y}_e - y_1)^2}$. The solution of system (4) is found in two steps. All the details of the algorithm can be found in [15] and, for the sake of simplicity, we only give a few hints. First, the system (4) is solved as if it was a linear one, by using the least square (LS) technique. Once the solution $\underline{\hat{\phi}}_1$ has been obtained, in order to take into account the relation between $\hat{x}_e$, $\hat{y}_e$, and $\hat{r}_1$ the following system is derived:

$$\underline{\underline{G}}'\underline{\phi}_2 = \underline{h}' \tag{6}$$

where

$$\underline{h}' = [([\underline{\hat{\phi}}_1]_1 - x_1)^2, ([\underline{\hat{\phi}}_1]_2 - y_1)^2, [\underline{\hat{\phi}}_1]_3^2]^T, \quad \underline{\underline{G}}' = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad \underline{\phi}_2 = [(\hat{x}_e - x_1)^2, (\hat{y}_e - y_1)^2]^T.$$

System (6) is then solved, once again, by means of the LS technique, and the solution $\underline{\hat{\phi}}_2$ is used to obtain the final position estimate of the target, which can be expressed as follows:

$$\underline{u}_e = \underline{\underline{U}}\left[\sqrt{[\underline{\hat{\phi}}_2]_1}, \sqrt{[\underline{\hat{\phi}}_2]_2}\right]^T + \underline{s}_1$$

where $\underline{\underline{U}} = \mathrm{diag}[\mathrm{sgn}(\underline{\hat{\phi}}_1 - \underline{s}_1)]$.

3.2. Plane intersection method

According to the PI method, introduced in [7], any triple of beacons (which leads to a pair of TDOA measurements) identifies the major axes of a conic, a focus of which is the position of the source. Given at least two triples of **beacons**, the position estimate can then be determined by solving the system given by the equations of the corresponding axes. By considering the axes identified by $\{\underline{s}_1, \underline{s}_2, \underline{s}_k\}$, $k \in \{3, \ldots, M\}$, the system can be written as

$$\underline{\underline{\hat{A}}}\,\underline{\hat{u}}_e = \underline{\hat{b}} \tag{7}$$

where

$$\underline{\underline{\hat{A}}} = \begin{pmatrix} (x_1 - x_2)\hat{\Delta}_{13} - (x_1 - x_3)\hat{\Delta}_{12} & (y_1 - y_2)\hat{\Delta}_{13} - (y_1 - y_3)\hat{\Delta}_{12} \\ (x_1 - x_2)\hat{\Delta}_{14} - (x_1 - x_4)\hat{\Delta}_{12} & (y_1 - y_2)\hat{\Delta}_{14} - (y_1 - y_4)\hat{\Delta}_{12} \\ \vdots & \vdots \\ (x_1 - x_2)\hat{\Delta}_{1M} - (x_1 - x_M)\hat{\Delta}_{12} & (y_1 - y_2)\hat{\Delta}_{1M} - (y_1 - y_M)\hat{\Delta}_{12} \end{pmatrix} \tag{8}$$

and the ith element of $\underline{\hat{b}}$, $\forall i\{1, \ldots, M-2\}$, is

$$\hat{b}_i = -\hat{\Delta}_{12}\hat{\Delta}_{1,i+2}(\hat{\Delta}_{1,i+2} - \hat{\Delta}_{12}) + (K_1^2 - K_2^2)\hat{\Delta}_{1,i+2} - (K_1^2 - K_{i+2}^2)\hat{\Delta}_{12}.$$

The LS solution of (7) is then given by

$$\underline{\hat{u}}_e = (\underline{\underline{\hat{A}}}^T\underline{\underline{\hat{A}}})^{-1}\underline{\underline{\hat{A}}}^T\underline{\hat{b}}. \tag{9}$$

3.3. A swarm intelligent approach

The starting point for the TSML method is system (4). Through simple algebraic manipulations, this system can be rewritten as

$$\underline{\underline{B}}\,\underline{\hat{u}}_e = \underline{\hat{t}} \tag{10}$$

where

$$\underline{\underline{B}} = -2\begin{pmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \\ \vdots & \vdots \\ x_M - x_1 & y_M - y_1 \end{pmatrix}, \quad \underline{\hat{t}} = \begin{pmatrix} \hat{r}_2^2 - \hat{r}_1^2 + K_1^2 - K_2^2 \\ \hat{r}_3^2 - \hat{r}_1^2 + K_1^2 - K_3^2 \\ \vdots \\ \hat{r}_M^2 - \hat{r}_1^2 + K_1^2 - K_M^2 \end{pmatrix}. \tag{11}$$

Note that with the new formulation (10), the measurements affected by noise only appear in vector $\underline{\hat{t}}$, while matrix $\underline{\underline{B}}$ only contains known parameters. Instead of solving directly the system (10), we reinterpret it as an optimization problem, with solution $\underline{\hat{u}}_e$ expressed as:

$$\underline{\hat{u}}_e = \mathrm{argmin}_{\underline{u}} F(\underline{u}_e) \tag{12}$$

where the fitness function $F(\cdot)$ is the following:

$$F(\underline{u}_e) \triangleq ||\hat{t} - \underline{B}\,\underline{u}||. \tag{13}$$

Among all possible optimization algorithms which could be used to solve (10), we now consider the PSO algorithm, originally introduced in [16]. We remark that the use of soft computing techniques for the localization of nodes in WSNs has been already proposed in the literature [17,18].

According to the PSO algorithm, the set of potential solutions of an optimization problem can be modeled as a swarm of particles, whose positions are randomly initialized in the region of interest, and which are guided towards the optimal solution by exploiting "social interactions" among themselves. In the following, the size of the swarm is denoted as S. It is assumed that every particle i in the swarm, at each iteration step t ($t \in \{0, 1, 2, \dots\}$), is associated with a position $\underline{x}^i(t)$ (within the region of interest) and with a velocity $\underline{v}^i(t)$, which are both randomly initialized, at the beginning, with values $\underline{x}^i(0)$ and $\underline{v}^i(0)$ and which are updated at each iteration [19]. Moreover, it is supposed that the entire system has a global memory, which allows each particle to know, at every instant, not only its own best position (according to the lowest value of the fitness function (13)) reached so far, but also the best position among the ones reached by any other particle in the swarm. Each particle also keeps track of the values of the fitness function in correspondence to both its best position and the global best position. All these values are used to iteratively update the velocity and the position of every particle. The updating rule for the velocity of particle i is [20]

$$\begin{aligned}\underline{v}^i(t+1) &= \omega(t)\underline{v}^i(t) + c_1R_1(t)(\underline{y}^i(t) - \underline{x}^i(t)) + c_2R_2(t)(\underline{y}(t) - \underline{x}^i(t)),\\ &\quad i \in \{1, \dots, S\}\end{aligned} \tag{14}$$

where $\omega(t)$ is a weight called *inertial factor*; c_1 and c_2 are positive real parameters called *cognition parameter* and *social parameter*, respectively; $R_1(t)$ and $R_2(t)$ are independent random variables uniformly distributed in (0, 1), whose realizations are drawn at each iteration step. Finally, $\underline{y}^i(t)$ and $\underline{y}(t)$ are the position of the ith particle with the best objective function and the position of the particle with the best (among all particles) objective function reached until instant t, respectively. For the considered minimization problem (12), these positions can be expressed as follows:

$$\begin{aligned}\underline{y}^{(i)}(t) &= \operatorname{argmin}_{\underline{z} \in \{\underline{x}^{(i)}(0), \dots \underline{x}^{(i)}(t)\}} F(\underline{z})\\ \underline{y}(t) &= \operatorname{argmin}_{\underline{z} \in \{\underline{y}^{(1)}(t), \dots \underline{y}^{(S)}(t)\}} F(\underline{z}).\end{aligned} \tag{15}$$

The rationale behind the updating rule (14) is to add to the velocity at the previous epoch (which is weighed by a multiplicative factor) a stochastic combination of the direction to the best position of the ith particle and to the best global position. The definition of the velocity given in (14) is then added to the current position, in order to update the position of the corresponding particle, according to the rule

$$\underline{x}^i(t+1) = \underline{x}^i(t) + \underline{v}^i(t+1), \quad i \in \{1, \dots, S\}. \tag{16}$$

This process is iterated until a stopping criterion (such as achieving a satisfactory value of the fitness function or reaching a maximum number of iterations) is met. The solution of (12) will then correspond to the position of the particle which best suits the optimization requirements in the last iteration.

In order to obtain the simulation results presented in Section 3.4, the PSO algorithm has been implemented in Matlab with a population of $S = 40$ individuals and with a stopping condition corresponding to 50 PSO iterations. We consider $c_1 = c_2 = 2$, in order to make both the averages of the weights for social and cognition contributions equal to 1. The initial positions of all the particles are randomly initialized over the entire region of interest, namely a square with 10 m-long sides. The choice of the inertial factor $\omega(t)$ deserves some more comments. From (14), it can be observed that this parameter quantifies the ability of the swarm to explore new areas. More precisely, a relatively large value of $\omega(t)$ improves the exploitation ability of the swarm, which is a good feature to avoid local optima in optimization problem. Since, however, we are considering a convex problem (i.e., a problem without local minima), we can ignore this parameter and, unlike [9,10], we set $\omega(t) = 0$, as proposed in [21]. This choice, besides accelerating the speed of convergence, also reduces the computational complexity of the algorithm. As a matter of fact, since the first addend at the right-hand side of (14) is not considered, there is no need to initialize and define velocities, and the updating rule for the positions of the particles becomes:

$$\begin{aligned}\underline{x}^i(t+1) &= \underline{x}^i(t) + c_1R_1(t)(\underline{y}^i(t) - \underline{x}^i(t)) + c_2R_2(t)(\underline{y}(t) - \underline{x}^i(t)),\\ &\quad i = 1, \dots, S.\end{aligned}$$

3.4. Results

We compare, through simulations, the performance of the aforementioned algorithms, considering the impact of (i) the number of beacons and (ii) the distance between beacons and nodes. All the simulation results presented throughout the paper have been obtained by using a Matlab simulator, in which we have implemented the two geometrical localization methods and the PSO algorithm. The Matlab implementation follows strictly the theoretical operational principles of the considered algorithms (no ad-hoc simulation tools have been considered). The performance is evaluated in terms of the following root mean square error (RMSE) between true and estimated positions:

$$\text{RMSE} \triangleq \sqrt{\mathbb{E}[(\hat{x}_e - x_e)^2 + (\hat{y}_e - y_e)^2]}. \tag{17}$$

In the following simulations, this average inside the square root at the right-hand side of (17) is obtained considering 100 independent simulation runs per scenario.

3.4.1. Impact of the number of beacons

We investigate how the obtained localization accuracy changes as the number of beacons reduces. In order to do this, we first consider the scenario shown in Fig. 1(a), where the number M of beacons is set to either 8 or 4. These two cases are shown in Fig. 2(a) and (c) respectively, where circles correspond to beacons while squares represent the nodes whose positions need to be estimated – we recall that the position estimates of the latter group of nodes are obtained sequentially (one at a time).

In Fig. 2(b) and (d) the RMSEs corresponding to the scenarios in Fig. 2(a) and (c) are shown as functions of the node index. In all cases, the RMSE is computed using the three considered algorithms (namely the TSML method, the PI method, and the PSO algorithm). It can be observed that when using 8 beacons, the performance of the three algorithms is comparable. When, instead, the number of beacons is reduced to 4, only the PSO algorithm guarantees reliable position estimates of all the nodes. These results suggest that the PSO algorithm is an appealing choice for auto-localization of static nodes, especially in large environments such as the ones we are interested in. As a matter of fact, the possibility of halving the number of beacons in a huge warehouse (keeping, at the same time, the same accuracy in the position estimates), allows significant cost savings.

3.4.2. Impact of physical distances

We now investigate the impact of the distance between beacons and nodes on the resulting position estimates. For this purpose, we

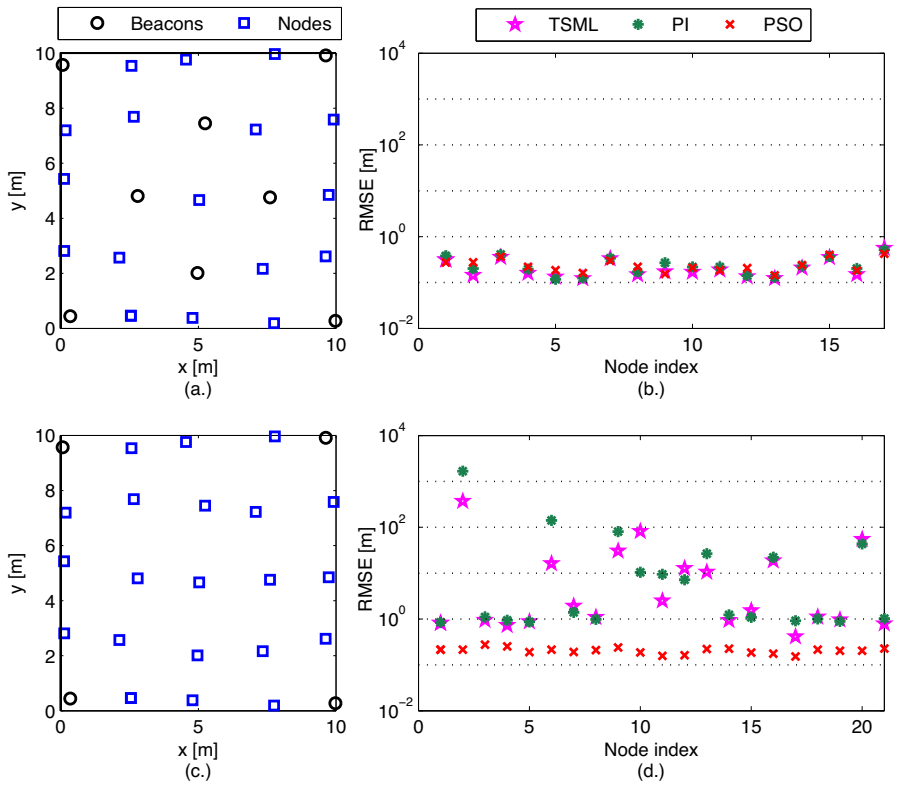

Fig. 2. Performance results in the scenario of Fig. 1(a). In (a) and (c) two possible configurations of beacons (*circles*) and nodes (*squares*) are shown. In (b) and (d), the corresponding performance, in terms of RMSE, is shown, considering the TSML method (*stars*), the PI method (*dots*) and the PSO algorithm (*crosses*).

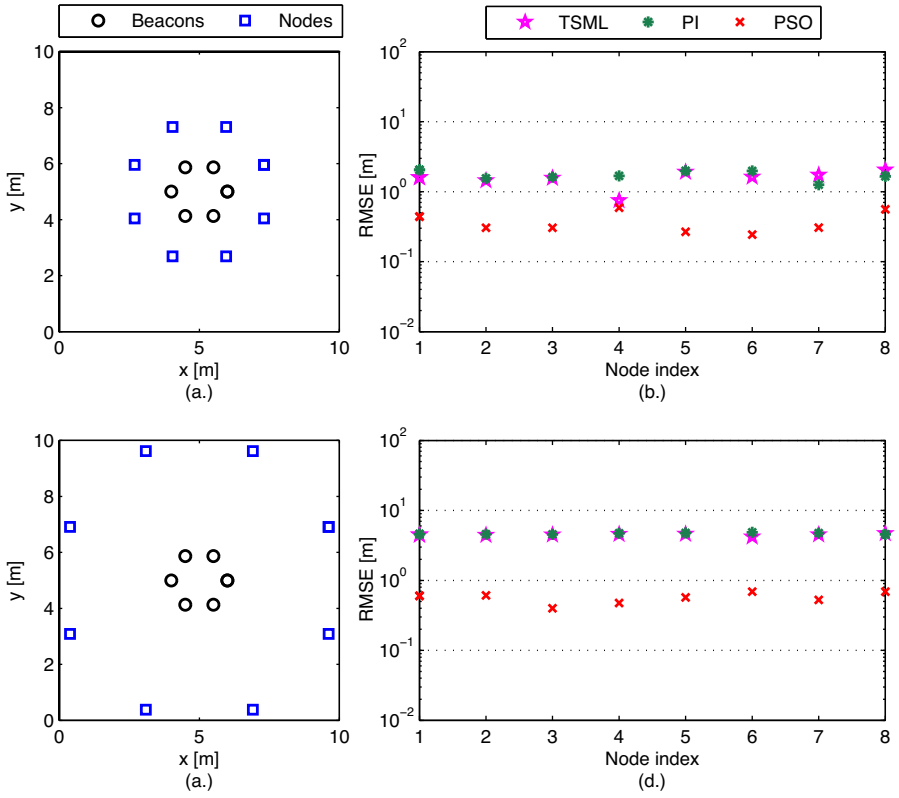

Fig. 3. Performance results in the scenario of Fig. 1(b). In (a) and (c) two possible configurations of beacons (*circles*) and nodes (*squares*) are shown. In (b) and (d), the corresponding performance, in terms of RMSE, is shown, considering the TSML method (*stars*), the PI method (*dots*) and the PSO algorithm (*crosses*).

consider the scenario described in Fig. 1(b), where the beacons are the 6 nodes closer to the center of the square. First, we estimate the positions of the 8 nodes which lay on the circumference with the smaller radius, namely the ones whose distance from the baricenter of the beacons is 2 m. This scenario is shown in Fig. 3(a), where circles correspond to beacons while squares correspond to the nodes whose positions need to be estimated. Then, we estimate the positions of the 8 farthest nodes, whose distance from the baricenter of the beacons is 5 m, as shown in Fig. 3(c). Simulation results show that, in both cases, the PSO algorithm outperforms the TSML and PI methods. Moreover, it can be observed that, as the distance between beacons and nodes increase, the performance with the TSML and PI methods significantly degrades (the RMSE increases by almost an order of magnitude), whereas the precision accuracy of the PSO algorithm does not change.

4. A hybrid PSO

In this section, a hybrid version of the PSO algorithm, inspired by evolution strategy, is proposed. A preliminary version was proposed in [11].

4.1. The idea

We assume that the main assumptions behind the standard PSO algorithm still hold, but we introduce a new processing step, at each iteration, after the computation of the new estimated positions of the particles and the corresponding values of the fitness function. To be more precise, among all the S particles, we consider the $T \triangleq \lfloor S/2 \rfloor$ ones which have the worst (namely, the higher) values of the fitness function, and we move them in a neighbourhood of the particle which has the best (namely, the lowest) value of the fitness function at that step. This procedure can be described as follows: at each epoch t of the recursive PSO algorithm, given the current fitness function of each particle, we consider a permutation $\mathcal{P}(t)$ in the set of indices $\{1, \ldots, S\}$, such that

$$F(\underline{x}^{\mathcal{P}[i]}(t)) \leq F(\underline{x}^{\mathcal{P}[j]}(t)), \quad \text{if} \quad i \leq j.$$

In other words, we order the indices of all the particles in ascending order, starting form the one with the best fitness value (whose position is $\underline{x}^{\mathcal{P}[1]}(t)$) to the one with the worst value of the fitness function (whose position is $\underline{x}^{\mathcal{P}[S]}(t)$).

Then, for every index $k \in \{1, \ldots, T\}$, we replace the positions of the particles with indices $\{\mathcal{P}[T+k]\}$, with positions close to $\underline{x}^{\mathcal{P}[1]}(t)$, which is the position of the best particle. The rule for this replacement is the following:

$$\underline{x}^{\mathcal{P}(T+k)}(t) \leftarrow \underline{x}^{\mathcal{P}(1)}(t) + r^{(k)}(t), \quad k = 1, \ldots, T \tag{18}$$

where $r^{(k)}(t) \sim U(-0.5, 0.5)$ and $\{r^{(k)}(t)\}$ are independent. In this way, half of the particles (the "worst" ones, i.e., the ones with higher values of fitness function) are moved towards the position of the "best" particle. After this position replacement, the fitness function of the T just moved particles is evaluated, and, from then, the proposed algorithm continues following the steps of the standard PSO algorithm and the updating rules for velocities and positions are the same.

In [11], a similar approach is considered, the only difference being that the replacement rule (18) is substituted by the following:

$$\underline{x}^{\mathcal{P}(T+k)}(t) \leftarrow \underline{x}^{\mathcal{P}(k)}(t) + r^{(k)}(t), \quad k = 1, \ldots, T, \tag{19}$$

where $r^{(k)}(t) \sim U(0, 1)$ and $\{r^{(k)}(t)\}$ are independent.

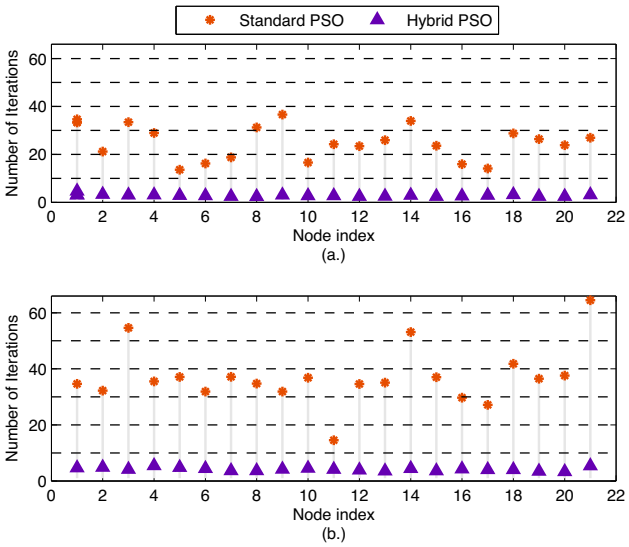

Fig. 4. (a) Averages of the minimum number of iterations needed to achieve an error tolerance of 10^{-1} m, for each of the 21 nodes which need to be estimated, when using the standard PSO algorithm (*dots*), and when using the hybrid PSO algorithm (*triangles*). (b) Averages of the minimum number of iterations needed to achieve an error tolerance of 5×10^{-2} m, for each node, when using the standard PSO algorithm (*dots*), and when using the hybrid PSO algorithm (*triangles*).

4.2. Comparison between standard PSO and hybrid PSO

The changes introduced in the hybrid version of the PSO algorithm are meant to fasten convergence to the optimal solution. As a matter of fact, the effect of the additional step is to rapidly push the swarm towards the (currently) best solution. It is worth observing that the introduction of the additional step described in (18) implies that the computational cost of each iteration of the hybrid PSO algorithm is approximately 1.5 times the cost of each iteration of the standard PSO algorithm. As a matter of fact, half of the particles need to be replaced and the corresponding values of the fitness function must be re-evaluated. However, it will be shown that, from a global computational viewpoint, the hybrid PSO algorithm turns out to be significantly more efficient than the standard PSO algorithm.

In order to evaluate the convergence speed, instead of considering, as a stopping criterion, a fixed number of iterations (as indicated in the last paragraph of Section 3.3), we now consider the achievement of a fixed maximum RMSE. In particular, in Section 4.2.1 we investigate the impact of the number of iterations, whereas in Section 4.2.2 the impact of the swarm size is investigated. Finally, in Section 4.2.3, a comparison of the computational complexities of PSO and hybrid PSO algorithms is presented.

4.2.1. Accuracy versus number of iterations

We now present some simulation results concerning the minimum number of iterations required by the PSO algorithm to achieve a given error tolerance. The population size and all the other parameters are kept equal to the ones already set in Section 3.3, except, as indicated above, for the stopping criterion. We compare the minimum number of iterations needed (to obtain the same performance) when using the standard PSO algorithm and when using the proposed hybrid version. All the results are relative to the scenario described in Fig. 1(a), and the beacons are assumed to be the 4 nodes at the vertices of the square, as in Fig. 2(c).

First, we set the maximum acceptable RMSE to 10^{-1} m. The simulation results, obtained by averaging over 100 independent runs, are shown in Fig. 4(a). It can be observed that the number of iterations needed to achieve the given error tolerance when using the

hybrid PSO algorithm is far smaller than the one needed when using the standard PSO algorithm. To be precise, the average (over all nodes) minimum number of iterations needed to achieve an RMSE equal to 10^{-1} m is 25, while this value decreases to 3 when using the proposed hybrid PSO algorithm.

Then, we reduce the error tolerance to 5×10^{-2} m: the corresponding results, are shown in Fig. 4(b). Once again, the number of iterations needed to achieve the given tolerance decreases when using the hybrid PSO algorithm instead of the standard PSO algorithm. In this case, the average (over all nodes) minimum number of iterations needed to achieve the given tolerance is: 37 when using the standard PSO algorithm; and 4 when using the proposed hybrid PSO algorithm.

Comparing the results of Fig. 4(a) with the ones in Fig. 4(b), it can be observed that, as expected, the number of iterations needed when the tolerance is smaller (i.e., 0.5×10^{-2} m) is larger than when considering a higher value of the maximum tolerable RMSE (namely, 10^{-1} m).

4.2.2. Impact of the population size

In the following, considering the same node configuration of Section 4.2.1 and setting the error tolerance to 5×10^{-2} m, we evaluate, as a function of the size S of the swarm, the number of iterations needed, on average, to achieve this tolerance. Besides the already considered value $S = 40$, we consider also its half ($S = 20$) and its double ($S = 80$). The simulation results relative to these values are shown in Fig. 5. More precisely, Fig. 5(a) refers to the standard PSO algorithm, while Fig. 5(b) refers to the hybrid PSO algorithm. In both cases, as expected, the number of iterations decreases as the population size increases. Comparing the results in Fig. 5(a) with those in Fig. 5(b) it can be noticed that, regardless the swarm size, the hybrid PSO algorithm allows obtaining the same performance in a far smaller number of iterations. More precisely, when $S = 20$, the average number of iterations required by the standard PSO algorithm is 95, but it reduces to 7 when using the hybrid PSO algorithm. Similarly, when $S = 40$, the average number of iterations with the PSO algorithm is 37, and it reduces to 4 when using the hybrid PSO algorithm. The number of iterations with the standard and the hybrid PSO algorithms reduces to 14 and 3, respectively, when the population size S is 80.

The impact of the population size on the convergence speed is clearly highlighted in Fig. 6, where the average number of iterations (averaged over 100 independent runs and over all the nodes whose positions need to be estimated) needed to achieve a maximum tolerable position estimation RMSE of (a) 10^{-1} m and (b) 5×10^{-2} m is shown as a function of the population size S (varying from 5 to 80 individuals). For both tolerance values, Fig. 6 shows that the proposed hybrid version of the PSO algorithm allows obtaining the required accuracy with a far smaller number of iterations, especially if the population size is small. Moreover, the convergence of the hybrid PSO is very fast even when the population size is small (e.g., $S = 20$ or $S = 30$), making it a very attractive choice also for real-time dynamic localization.

4.2.3. Computational complexity

We now show that, in spite of the additional step in (18), the hybrid PSO algorithm is computationally less expensive than the standard PSO algorithm. This complexity gain comes mainly from the fact that the number of iterations needed to achieve a given accuracy is, as shown in the previous subsections, far smaller with the hybrid PSO algorithm.

Let us denote as N the number of nodes which have to be estimated and as $N_{\text{iter}}^{(\text{PSO})}$ and $N_{\text{iter}}^{(hPSO)}$ the numbers of iterations required by the PSO and the hybrid PSO algorithms to converge, respectively. From the results in Fig. 6, it can be concluded that $N_{\text{iter}}^{(\text{PSO})}$ can be roughly approximated as $1000/S$, i.e., the swarm size S needs to be very large to guarantee fast convergence. At the opposite, $N_{\text{iter}}^{(\text{hPSO})}$ is an approximately constant function of S, typically assuming values between 5 and 10 (the only exception being the case with a tolerance of 5×10^{-2} m and $S = 5$, i.e., with a very small population size, where $N_{\text{iter}}^{(\text{hPSO})} = 17$). At each iteration, the computational complexity is associated with two operations carried out for each particle of the swarm at each node: (i) the evaluation of the fitness function and (ii) the position update.

- Considering the *PSO algorithm*, the evaluation of the fitness function defined in (13) for a single particle in the swarm involves $M - 2$ additions and $M - 1$ multiplications. Moreover, from (14) and (16) it follows that the position update of a single particle in the swarm involves 5 additions and 5 multiplications. Hence, for each particle in the swarm, at each iteration one needs to compute $M + 3$ additions and $M + 4$ multiplications. The computational complexity over the entire swarm for the position estimate of a single node is then obtained by multiplying the above numbers by S, obtaining: $S(M + 3)$ additions and $S(M + 4)$ multiplications.
- When considering the *hybrid PSO algorithm*, the number of operations involved in the evaluation of the fitness function for the entire swarm is twice that of the PSO algorithm: as a matter of fact, at each iteration, the fitness function of each particle is evaluated twice. This leads to $2S(M - 2)$ additions and $2S(M - 1)$ multiplications for the evaluation of the fitness function for all particles of the swarm. The position update of each particle involves the same numbers of operations as in the PSO algorithm, with $\lfloor S/2 \rfloor$ supplementary additions, due to the replacement of half of the particles given in (18), leading to $5S + \lfloor S/2 \rfloor$ additions and $5S$ multiplications for the entire swarm at a node. It can then be concluded that, at each iteration, the hybrid PSO algorithm requires $S(2M + 1) + \lfloor S/2 \rfloor$ additions and $S(2M + 3)$ multiplications for the position estimate of each node.

Therefore, one obtains the computational complexities (in terms of numbers of multiplications and additions) summarized in Table 1: the first two rows refer to the computational complexities per iteration (obtained by multiplying the previously explained results by the number of nodes N), whereas the last two rows refer to the total complexity (taking into account proper numbers of iterations for the considered algorithms).

We now refer to the node configuration in Fig. 1(a) and assume that the beacons are the $M = 4$ nodes nearest to the corners. Hence, the remaining $N = 21$ nodes' positions need to be estimated. We first consider a swarm composed of $S = 30$ particles at each of the $N = 21$ nodes with positions to be estimated. According to the results in Fig. 6, in this case the numbers of iterations needed to achieve an accuracy of 5×10^{-2} m are $N_{\text{iter}}^{(\text{PSO})} = 53$ and $N_{\text{iter}}^{(\text{hPSO})} = 4$, respectively. Then, we consider a swarm composed of $S = 60$ particles at each of the $N = 21$ nodes. According to the results in Fig. 6, in this case the numbers of iterations needed to achieve an accuracy of 5×10^{-2} m are $N_{\text{iter}}^{(\text{PSO})} = 36$ and $N_{\text{iter}}^{(\text{hPSO})} = 3$, respectively. In Table 2, the corresponding values of the total numbers of multiplications and additions are shown. It can be observed that, in both cases, the computational complexity of the hybrid PSO algorithm is around 10% (in terms of number of multiplications and number of additions) of that of the PSO algorithm.

In the last row of Table 2, the simulation run times (on Mac OS 10.5.8, with a 2 GHz Intel Core 2 Duo processor) required by the considered algorithms to estimate the positions of the 21 nodes in the aforementioned scenario are also shown. The obtained simulation run times confirm the analytical prediction of Table 1.

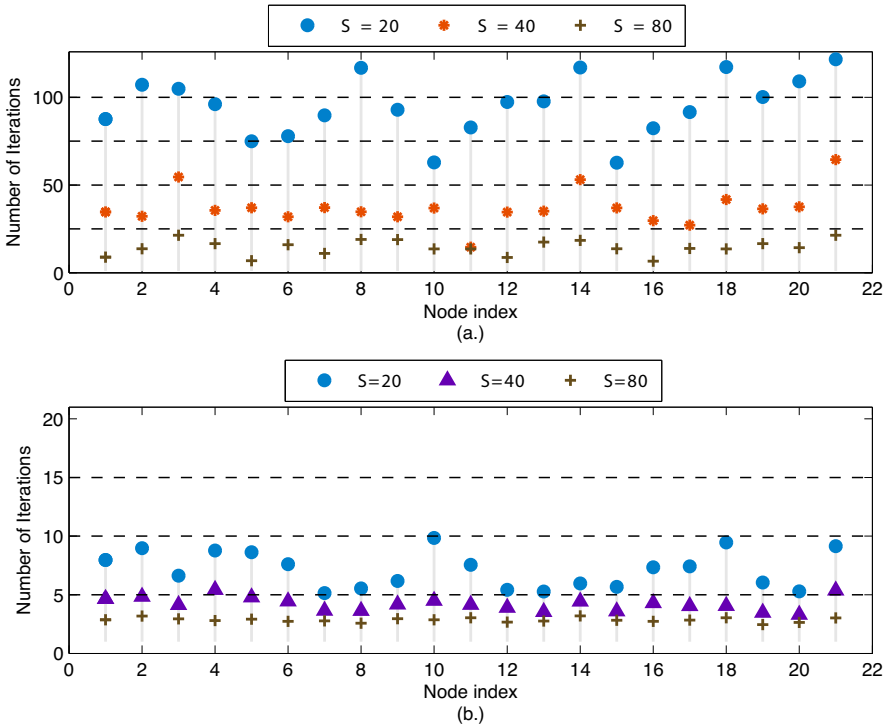

Fig. 5. (a) The averages of the minimum number of iterations needed to achieve an error tolerance of 5×10^{-2} m are represented, for each of the 21 nodes, when using the standard PSO algorithm and when the population size is $S = 20$ (*circles*), $S = 40$ (*dots*), and $S = 80$ (*plus*). (b) The averages of the minimum number of iterations needed to achieve an error tolerance of 5×10^{-2} m are represented, for each node, when using the hybrid PSO algorithm and when the population size is $S = 20$ (*circles*), $S = 40$ (*triangles*), and $S = 80$ (*plus*).

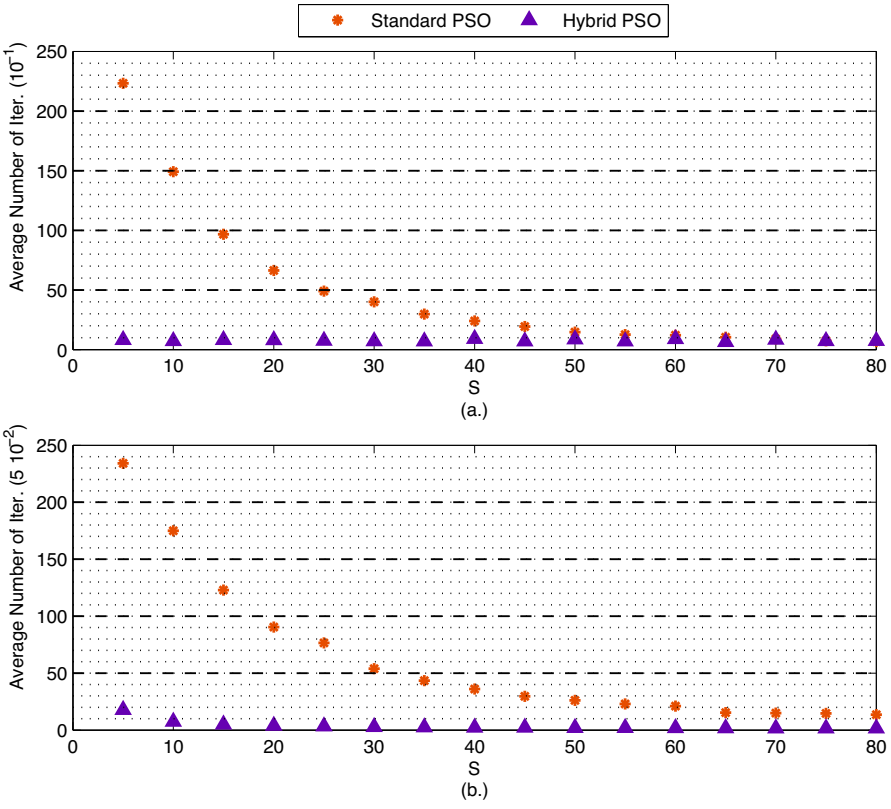

Fig. 6. (a) Average number of iterations, averaged over all the nodes, needed to achieve an error tolerance of 10^{-1} m when using the standard PSO algorithm (*dots*), and when using the hybrid PSO algorithm (*triangles*). (b) Average number of iterations, averaged over all the nodes, needed to achieve an error tolerance of 5×10^{-2} m when using the standard PSO algorithm (*dots*), and when using the hybrid PSO algorithm (*triangles*).

Table 1
Numbers of multiplications and additions (per iteration and total) of the PSO and hybrid PSO algorithms as functions of the number of nodes N, the number of particles in the swarm S, the number of beacons M, and the number of iterations ($N_{\text{iter}}^{(\text{PSO})}$ and $N_{\text{iter}}^{(\text{hPSO})}$, respectively).

	PSO	Hybrid PSO
Number of multiplications/iter.	$NS(M+4)$	$NS(2M+3)$
Number of additions/iter.	$NS(M+3)$	$N\left[S(2M+1)+\lfloor S/2\rfloor\right]$
Total number of multiplications	$NS(M+4)N_{\text{iter}}^{(\text{PSO})}$	$NS(2M+3)N_{\text{iter}}^{(\text{hPSO})}$
Total number of additions	$NS(M+3)N_{\text{iter}}^{(\text{PSO})}$	$N\left[S(2M+1)+\lfloor S/2\rfloor\right]N_{\text{iter}}^{(\text{hPSO})}$

Table 2
Computational complexity comparison between PSO and hybrid PSO algorithm, in a scenario with $N=21$, $M=4$. Two possible values for the swarm size S are considered: 30 (which corresponds, from Fig. 6, to $N_{\text{iter}}^{(\text{PSO})}=53$ and $N_{\text{iter}}^{(\text{hPSO})}=4$) and 60 (which corresponds, from Fig. 6, to $N_{\text{iter}}^{(\text{PSO})}=36$ and $N_{\text{iter}}^{(\text{hPSO})}=3$). In the second and third rows, the numerical values predicted by the last two rows of Table 1 are shown. In the fourth row, the simulation run times obtained with a Mac OS 10.5.8 with a 2 GHz Intel Core 2 Duo processor are also shown.

	PSO		Hybrid PSO	
S	30	60	30	60
Total number of multiplications	267120	362, 880	27, 720	41580
Total number of additions	233730	317, 520	23, 940	35, 910
Simulation run time	2.08 s	2.68 s	0.20 s	0.31 s

5. Conclusions

Three approaches to UWB signaling-based auto-localization of nodes in a static wireless network have been considered: two stem from the geometry-oriented localization literature (the TSML method and the PI method) and one originates from the soft computing literature (a PSO-based approach). The obtained results show that the PSO-based approach guarantees, with respect to the other algorithms, a better accuracy in the position estimate, both in terms of required number of beacons and of tolerable distance between beacons and nodes. Furthermore, an improved version of the PSO algorithm has been proposed, showing that its convergence speed is significantly higher than that of classical PSO approach. We have also investigated the convergence speed as a function of the population size, showing that the novel hybrid PSO algorithm allows fast convergence even with a small swarm size. As an attractive side benefit, the improvement of the convergence speed of the hybrid PSO algorithm is associated with a relevant reduction of the total computational complexity.

References

[1] S. Gezici, H.V. Poor, Position estimation via ultra-wide-band signals, Proc. IEEE 97 (2) (Feb 2009) 386–403.
[2] J. Zhang, P.V. Orlik, Z. Sahinoglu, A.F. Molisch, P. Kinney, UWB systems for wireless sensor networks, Proc. IEEE 97 (February (2)) (2009) 313–331.
[3] Wade H. Foy, Position-location solutions by Taylor-series estimation, IEEE Trans. Aerosp. Electron. Syst. AES-12 (March (2)) (1976) 187–194.
[4] C. Mensing, P. Simon, Positioning algorithms for cellular networks using TDOA, in: 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006), Toulouse, May 2006, pp. 513–516.
[5] Y. Zheng, W. Chenshu, C. Tao, Z. Yiyang, G. Wei, L. Yunhao, Detecting outlier measurements based on graph rigidity for wireless sensor network localization, IEEE Trans. Veh. Technol. 62 (1) (2013) 374–383.
[6] G. Molina, E. Alba, Location discovery in wireless sensor networks using metaheuristics, Appl. Soft Comput. 11 (January (1)) (2011) 1223–1240.
[7] Ralph O. Schmidt, A new approach to geometry of range difference location, IEEE Trans. Aerosp. Electron. Syst. 8 (November (6)) (1972) 821–835.
[8] Y. Chan, K.C. Ho, A simple and efficient estimator for hyperbolic location, IEEE Trans. Signal Process. 42 (August (8)) (1994) 1905–1915.
[9] S. Monica, G. Ferrari, Particle swarm optimization for auto-localization of nodes in wireless sensor networks, in: Proceedings of International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), 2013.
[10] S. Monica, G. Ferrari, Impact of the number of beacons in PSO-based auto-localization in UWB networks, in: Proceedings of the International Conference on the Applications of Evolutionary Computation (EvoApplications13), Track on Nature-Inspired Techniques for Communication Networks and Other Parallel and Distributed Systems (EvoCOMNET 2013), Vienna, 2013, pp. 42–51.
[11] S. Monica, G. Ferrari, Hybrid swarm intelligence node localization in wireless sensor networks, in: XVI Portuguese Conference on Artificial Intelligence (EPIA 2013, Track On) Artificial Life and Evolutionary Algorithms (ALEA), Azores, Portugal, 2013, September.
[12] S. Busanelli, G. Ferrari, Improved ultra wideband-based tracking of twin-receiver automated guided vehicles, Integr. Comput. Aided Eng. 19 (1) (2012) 3–22.
[13] A.F. Molisch, D. Cassioli, Chia-Chin Chong, S. Emami, A. Fort, B. Kannan, J. Karedal, J. Kunisch, H.G. Schantz, K. Siwiak, M.Z. Win, A comprehensive standardized model for ultrawideband propagation channels, IEEE Trans. Antennas Propag. 54 (11) (Nov 2006) 3151–3166.
[14] D. Dardari, C.C. Chong, M.Z. Win, Threshold-based time-of-arrival estimators in UWB dense multipath channels, IEEE Trans. Commun. 56 (August (8)) (2008) 1366–1378.
[15] K.C. Ho, W. Xu, An accurate algebraic solution for moving source location using TDOA and FDOA measurements, IEEE Trans. Signal Process. 52 (September (9)) (2004) 2453–2463.
[16] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks (ICNN), Perth, 1995, November, pp. 1942–1948.
[17] J.M. Alonso, M. Ocana, N. Hernandez, F. Herranz, A. Llamazares, M.A. Sotelo, L.M. Bergasa, L. Magdalena, Enhanced WiFi localization system based on soft computing techniques to deal with small-scale variations in wireless sensors, Appl. Soft Comput. 11 (8) (2008) 4677–4691.
[18] M. Vecchio, R. Lopez-Valcarce, F. Marcelloni, A two-objective evolutionary approach based on topological constraints for node localization in wireless sensor networks, Appl. Soft Comput. 12 (7) (2012) 1891–1901.
[19] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization, Swarm Intell. J. 1 (1) (2007) 33–57.
[20] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC), Washington, 1999, July, pp. 69–73.
[21] X.S. Yang, S. Deb, S. Fong, Accelerated particle swarm optimization and support vector machine for business optimization and applications, Commun. Comput. Inf. Sci. 136 (March) (2011) 53–66.