Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)

Using the Particle Swarm Optimization Algorithm for Robotic Search Applications

James M. Hereford, *Member, IEEE,* Michael Siebold, Shannon Nichols
Murray State University
Department of Engineering and Physics
Murray, KY 42071

Abstract— **This paper describes the experimental results of using the Particle Swarm Optimization (PSO) algorithm to control a suite of robots. In our approach, each bot is one particle in the PSO; each particle/bot makes measurements, updates its own position and velocity, updates its own personal best measurement (pbest) and personal best location (if necessary), and broadcasts to the other bots if it has found a global best measurement/position. We built three bots and tested the algorithm by letting the bots find the brightest spot of light in the room. The tests show that using the PSO to control a swarm can successfully find the target, even in the presence of obstacles.**

I. Introduction

Our goal is as follows: build a suite/swarm of (very) small robots that can search a room for a "target". We envision that the robots will be about the size of a quarter dollar, or smaller, and have a sensor or sensors that "sniff" out the desired target. For example, the target could be a bomb and the robot sensors would be a chemical detector that can distinguish the bomb from its surroundings. Or the target could be a radiation leak and the sensors would be radiation detectors. This type of scenario has many possible applications. In each scenario, there is a target point that is either (a) hard to find and/or (b) dangerous for humans to search out.

In place of humans, robots can be used to do the search. We envision using several small robots instead of one large robot. Several robots provide for a natural "fault tolerance". If one or more of the robots is damaged, there are still other functional robots that can continue to search. In addition, small robots will potentially be easier to transport (lighter), quicker to search (more robots can cover more area) and cheaper to make. The suite of robots will work together to solve the problem. However, there will not be a "master" robot or global processor that will direct the movements of each robot. Each robot determines its own movement and is able to communicate with other robots wirelessly.

It is not very efficient to have the suite of robots looking randomly around the room hoping to "get lucky" and find the target. There needs to be some way to coordinate the movements of the many robots. There needs to be an algorithm that can guide the robots toward promising regions to search while not getting distracted by local variations. The search algorithm must have the following constraints:

- The search algorithm should be distributed among the many robots. If the algorithm is located in one robot, then the system will fail if that robot fails. Also, there would be a lot of communications among the robots if each robot had to wait for movement commands from a central source. Thus, the search algorithm needs to be de-centralized.
- The search algorithm should be computationally simple. The processor on each bot is small, has limited memory, and there is a limited power source (a battery) so the processor needs to be power efficient. Therefore, the processor will be a simple processor. The search algorithm needs to be tailored to such a processor.
- The search algorithm should have a minimum amount of communications among the robots. The algorithm needs to be scalable from one robot up to 10's, 100's, even 1000's of robots. The upper limit on the number of robots will be set by the communication links among the robots. If each robot has to wait for information from other robots, then the system will break down as the number of robots increases. Instead, there needs to be a way to share information among the robots without requiring lots of communication traffic.
- The search algorithm must allow for contiguous movement of the robots. There are search algorithms that work well in simulation (e.g., genetic algorithms) but require "step" changes in the solutions at each iteration. This would not be feasible in this application.

This paper describes the results from programming a suite of robots to search for the brightest spot of light in a room. The algorithm is based on the Particle Swarm Optimization (PSO) algorithm that has proven successful in optimization type searches. The algorithm, described in [Hereford 2006], allows each bot to calculate its new position based on its present measurement and present position. In section 2, we give a brief overview of the modified PSO algorithm and summarize simulation results. In section 3, we describe the robots that we will use to implement the PSO (mitEBots). Section 4 gives the results of experimental runs using the

PSO with the bots. The results show that the PSO + bot swarm is able to find a target, even in the presence of obstacles. Section 5 gives the conclusions for this paper.

II. BACKGROUND

A. Classic PSO

In Particle Swarm Optimization (PSO) [Eberhart 1995, Eberhart 2001, Eberhart 2004], the potential solutions, called particles, "fly" through the problem space by following some simple rules. All of the particles have fitness values based on their position and have velocities which direct the flight of the particles. PSO is initialized with a group of random particles (solutions), and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) the particle has achieved so far. This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the population. This best value is a global best and called gbest.

After finding the two best values, the particle updates its velocity and positions with following equations:

$$v_{n+1} = w_i v_n + c1*rand*(pbest_n - p_n) + c2*rand*(gbest_n - p_n)$$
$$p_{n+1} = p_n + v_{n+1}$$

w_i is the inertia coefficient which slows velocity over time; v_n is the particle velocity; p_n is the current particle position in the search space; $pbest_n$ and $gbest_n$ are defined as the "personal" best and global best; rand is a random number between (0,1); c1, c2 are learning factors. The stop condition is usually the maximum number of allowed iterations for PSO to execute or the minimum error requirement. As with the other parameters, the stop condition depends on the problem to be optimized.

In summary, the advantages of the PSO over other algorithms are that (a) it is computationally simple and efficient; (b) each agent only needs to know its own local information and the global best to compute the new position, so there is a minimal amount of data transfer among the agents; and (c) the results from all agents in the population are not required to form the next generation, so a centralized processor is not required.

Because of the required search algorithm characteristics listed in section I, we chose the PSO as the starting point for the search control algorithm. The PSO is computationally simple. It requires only four multiplies and four add/subtracts to update the velocity and then one add to update the position. There is no complicated iterative equation solving required and no exponential or trig functions to implement. The PSO is also a distributed algorithm. Each agent/particle/bot can update its own velocity and position. The only external information is the global best – the best value by any particle within the population. The calculation of the global best can be done with a simple comparative statement. Thus, each bot does not need to know the results from each member of the population as in many traditional schemes.

The PSO also allows the contiguous movement of the bots. The updated position is relative to the current position so there is no jump changes in position or random movements. If there are constraints on the movement of the bot during each iteration, then limitations can be placed on the maximum and minimum velocity that is allowed for each particle/Bot.

B. Modified PSO

The idea of the distributed PSO (dPSO) algorithm is to have each particle/bot make measurements, update its own position and velocity, update its own personal best measurement (pbest) and personal best location (if necessary), and broadcast to the other bots if it has found a global best measurement/position. Thus, all velocity and position updates are done locally by each bot and only global best updates are broadcast. This reduces the amount of information that is communicated among the bots and increases the scalability of the algorithm.

There are several modifications that have to be made to the classic PSO algorithm for it to work effectively for the distributed search scenario. First, each bot has limited mobility (a limited turning radius, a small maximum velocity and it can not move backwards). The classic PSO assumes that the particles can move in any direction and with any velocity up to a stated Vmax. The bots, however, have to move within a limited cone.

A second modification is introduced to reduce the amount of communications traffic; the dPSO only broadcasts a new gbest location after it has been determined that the bot has indeed found an overall global best. Frequently, especially at the beginning of a search, more than one bot will find a new global best within a particular time interval. Rather than broadcast both the new value of gbest and the location, the local bot will only broadcast the gbest value. If it is determined that it indeed has the gbest, it will then broadcast the location of the global best so the other bots can use the new value in their velocity update equations.

Simulation results showed that the dPSO is a very good way of coordinating simple bots for a search operation. For target locations in the middle of the search space, bots utilizing the dPSO find the target 99 or 100 % of the time.

A lower maximum velocity for each bot leads to slightly more effective searches (2-3 % more effective), but it then takes longer (on average) to reach the target. The algorithm is scalable to large numbers of bots since the communications requirements do not increase as the number of bots is increased.

We note that the dPSO has potential to also be used as a parallel (multiprocessor) version of the PSO. However, the dPSO has not been optimized to provide speed or good test results with complex search functions like other parallel PSO techniques [Chang 2005]. Instead, the goal of the dPSO is to minimize the communications among the particles/bots and thus make it easily scalable to large numbers of particles.

C. Related Work

Other authors have investigated using multiple (simple) robots for search type applications. Hayes et al. report using autonomous mobile robots for beacon localization [Hayes 2000] and plume tracking/odor source localization [Hayes 2002]; they base their search techniques on biological principles (surge "upwind") but do not use the PSO algorithm directly. Doctor et al. [Doctor 2004] discuss using the PSO for multiple robot searches. Their focus is on optimizing the parameters of the search PSO and do not consider the scalability of the standard PSO to large numbers of robots. Pugh et al. [Pugh 2005, Pugh 2006] explore using PSO on problems in noisy environments, focusing on unsupervised robotic learning. They use the PSO to evolve a robot controller that avoids obstacles, but it does not appear that the distributed robots communicate with each other.

There is at least one other research group that is investigating using mobile robots as particles within a PSO search framework. Jatmiko et al. [Jatmiko 2006] use mobile robots for plume detection and traversal. They utilize a modified form of the PSO to control the robots and consider how the robots respond to search space changes such as turbulence and wind changes. They do not, however, consider how well the process will scale to large numbers of robots. Also, they have not yet published results of implementing their PSO variants in actual hardware robot swarms.

III. Description of mitEBots

To embed the dPSO algorithm into a swarm of robots, each robot in the swarm must have a certain minimal set of abilities. Each robot must be able to (a) make a measurement of the surrounding environment, (b) perform calculations, (c) determine accurate position information, (d) move to a new location in the search space and (e) (wirelessly) communicate with other bots. Each of these operations is required to implement the dPSO operations of function evaluation (make measurement), velocity and intended position update (simple calculations), move to the new position (move and position information), and communicate the global best value and location to other bots in the neighborhood or swarm (communication with other bots).

In addition to the required characteristics above, there are several desired characteristics for each bot. Each bot should be small, both in size and in weight, which also means the processor must be small and have limited computational capability. The bot must be self-contained in terms of power. Each bot may have an array of sensors that can make measurements. As mentioned above, the sensors could measure temperature, chemical density, radiation, sound level or electromagnetic radiation within a specific band.

We have built a set of robots to demonstrate a proof-of-principle experiment for the dPSO. We call our robots mitEBots. The mitEBots use Motes [Culler 2004] to provide sensing, processing, and communications and an electric car to provide movement. They are based on the CotsBots design [Bergbreiter 2003], but the mitEBots designed by Murray State University feature a Cricket mote [Teller 2003] as the design center point. Like the CotsBots, the primary locomotive hardware is a radio-controlled car that has been disemboweled of its electronics. There is a motor board to interface the Mote to the car's steering and drive motors. Each of the mitEBots contains a simple processor (Atmega 128L 8-bit microprocessor, 4 MHz clock) so each bot can perform calculations locally.

The Cricket board, developed at MIT's Computer Science and Artificial Intelligence Laboratory, attaches to a sensor board and comes equipped with a combination of RF and ultrasound technologies to provide location information. Three beacons (other Cricket motes) are mounted in the search space so each mitEBot can determine its x, y (and z if desired) position. Even in the presence of several competing beacon transmissions, Cricket achieves good precision and accuracy. Its decentralized architecture makes the system easy to deploy.

An alternative method to determine position information is suggested by Spears et al. [Spears 2006] that is based on trilateration. In the trilateration method, each bot is equipped with a radio frequency (RF) transceiver and three ultrasound transceivers. Each bot is able to determine how far away it is from neighboring bots and thus determine its position, assuming that the positions of the other bots are known. This method has the attractive feature of eliminating the need for stationary beacons at known

locations. However, it would require more communications packets be exchanged among the bots and there is a concern that position errors would grow over time since there are no fixed reference points.

Murray State University has built and tested three mitEBots for use with the PSO algorithm. (See Figure 1.) Each mitEBot can only turn a maximum of 36 degrees to the right or left and has a turning radius of approximately 17 cm. The maximum velocity is approximately 1.0 m/s when the batteries are fully charged but the bots do not move at maximum velocity during the search operations. Each Bot is approximately 16 cm long, 8.5 cm wide, and 7 cm tall. Two of the mitEBots weigh 364 grams and the third bot (built using a different model of remote-control car) weighs 387 grams which is about three times the weight of a handheld scientific calculator. No steps were taken to minimize the weight of the mitEBots as shown by the fact that three different battery systems are used: AA for the cricket mote, AAA for the motors and 9V for the interface board.

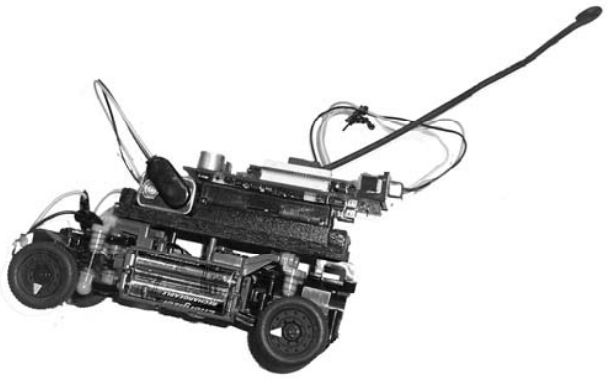

Figure 1: Murray State's mitEBot.

IV. RESULTS

A. Test conditions

To investigate the effectiveness of the dPSO algorithm, we did several experiments. In the experiments, a diffuse light source was placed near the ceiling of a dark room and pointed downward. Bots with light sensors were placed at various starting positions about 2 m apart to see (a) how often and (b) how quickly they could find the brightest spot of light in the room.

The layout of the test area is shown in Figure 2. The boxes in the middle represent the obstacles that we placed in the search space for the second half of the testing. The circles indicate the starting positions of the bots and the highest concentration of light is immediately to the left of the vertical obstacle. Since we are using a diffuse light source, the global best is actually a rectangle approximately .25 m by .3 m.

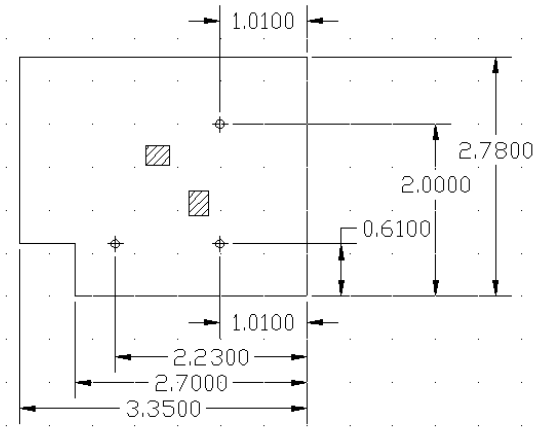

Figure 2: Graphical representation of test area. Dimensions shown are in meters.

B. Algorithm adjustments

To apply the dPSO to a robotic swarm search, we had to make several adjustments to the algorithm. First, the bots determine their position by triangulating from three cricket motes set up as beacons. For the bots to obtain accurate position measurements they must receive several data packets from each beacon. Missed packets from the beacons cause faulty distance measurements which leads to inaccurate position information for the local bot. To correct for any missed packets, the bots are programmed to move to the next position and then wait for two consecutive "clean" measurements (distance measurements within 2 cm) from all three beacons. This wait leads to relatively long search times.

A second algorithm adjustment had to be made because of mechanical faults in each bot. In general, the bots move in arcs even when the algorithm directs them to move in straight lines. Each bot moves toward the desired position in the search space but upon arrival at the destination point, the orientation direction is usually skewed relative to the movement. To compensate, we update the bot's orientation angle at each iteration based on the bot's current position and its previous position. This eliminates the buildup of orientation errors that occur when a strict dead reckoning system for orientation is used. We also orient the front wheels on the bot to point straight ahead after each move.

Unlike a simulation-only PSO, the hardware bots can get "stuck" at an obstacle or collide with another bot (particle). Once a bot got stuck or collided, we programmed the bot to back up and turn right. This allows the bot to move around long obstacles, such as a wall, even though it may require more than one cycle of backing up and turning to avoid.

C. *Qualitative results*

During the hardware experiments, we programmed each mitEBot with identical nesC programs. (The only difference is that each bot was programmed with a different identification number.) The PSO parameters used were c1 = 2, c2 = 2, and w_i = 1.0. We tried using w_i = 0.8 but it slowed the bots down considerably and led to many failed searches. The value of Vmax was set based on the hardware limitations of each bot. The value varied slightly from bot to bot but it was approximately 0.8 m/s for each one. Each bot was programmed to move in the desired direction for approximately 0.5 sec. The bot would then make a measurement, determine its new position, calculate its desired movement direction based on the PSO update, orient its wheels to that direction, if possible, and then move for 0.5 sec. The search was ended when there had been 20 iterations of the algorithm with no new global best discovered. We made some test runs using a stop condition with 10 iterations with no new gbest but we determined that 10 was insufficient.

Qualitative results for the robotic search are shown in Figures 3 and 4. Figure 3 shows the path traced out by 1 bot and Figure 4 shows the path traced out by 2 bots. Note that these are not simulation results. These are plots of the movement of the bot(s) through the search space where the (x,y) position of the bot(s) were monitored throughout the search. The x and y axes are to scale and the axes are in cm. The asterisks (*) marks in the figures represent positions where a new global best was found. The rectangular box is peak area of the search space. The plots only show the search up till the peak light value was found.

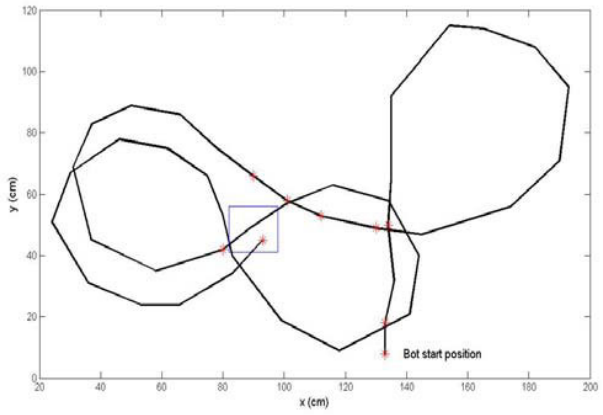

Figure 3: Path in search space with no obstacles for 1 bot.

In the one bot case (Figure 3), the bot starts in lower right corner. It moves up (north), finds new global bests and continues upward. When it starts to move away from the peak, it circles clockwise and then begins moving to the left (westward). When the light intensity measurements begin to taper off again, the bot circles counterclockwise. The circle behavior is because the bot is limited in its turning radius – it can not make a sharp turn. Thus, it must move toward the global best in a roundabout fashion. Eventually, it settles on to the peak light value.

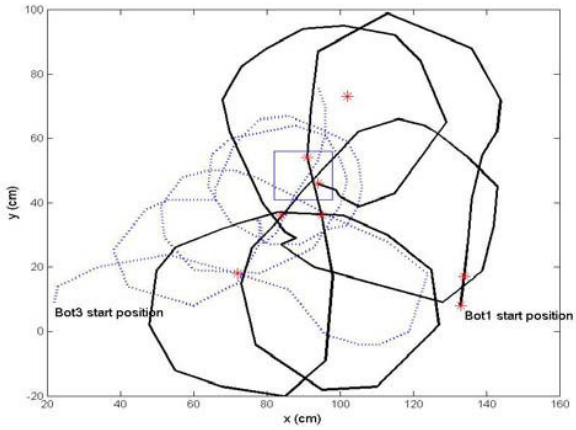

Figure 4: Path in search space with no obstacles for 2 bots in a 3 bot search.

Figure 4 shows the paths traced by two bots in a three bot search. (Only two bots are shown to simplify the figure.) In the multi bot case (Figure 4), the bots start at different locations (lower right and lower left) and proceed to move and take light intensity measurements. As seen with only 1 bot, if any of the bots starts moving away from the global best, then they circle around to return to the vicinity of the best value. In this search, there was a global best that occurred within the rectangular box, but a superior gbest was discovered within the next 20 iterations.

D. *Quantitative results*

The PSO algorithm was tested using swarm sizes of 1, 2 and 3 bots. (The 1-bot swarm was merely for baseline comparison.) Ten runs were made for each of the three cases. We tracked how many times the bots found the brightest spot and how long it took for the bots to locate the peak.

The bots were timed by a cricket mote which was not part of the swarm. This mote transmitted a radio signal which allowed all the bots to start searching simultaneously. This mote then listened for radio communication between the bots in the swarm. If twenty iterations of the algorithm passed with no new global best, then we presume the search is over and the external mote reported the length of time it took the first bot to find the highest light value.

The quantitative results are shown in Table 1. The results are for two different search spaces: one with no obstacles and one with obstacles. The table shows the number of bots in the swarm, the number of successful runs and the average

time to find the peak. The average time is for only the successful searches.

TABLE 1: INITIAL RESULTS FROM USING PSO TO PROGRAM A SUITE OF BOTS

Runs without obstacles			
Number of bots	Successful runs (out of 10)	Average time of successful runs (sec)	Standard deviation (sec)
1	8	205.1	88.7
2	10	200.3	65.5
3	10	132	72.1

Runs with obstacles			
Number of bots	Successful runs (out of 10)	Average time of successful runs (sec)	Standard deviation (sec)
1	6	205.9	152.9
2	9	225.1	151.4
3	9	150.2	93.9

Increasing the number of bots does two things. It leads to more successful searches and (usually) reduces the time to find the peak/best value. We see that even with the obstacles in the search space, the swarm is still able to find the "target" or peak light intensity (almost) every time. In general, the searches take longer when there are obstacles in the search space but the search is still successful. The times should be used for comparison purposes and not necessarily as an absolute reference. As mentioned in section 4.B, sketchy communications with the beacons forced the bots to wait for two consecutive good data points at each iteration. This waiting time greatly increased the search time.

The only data point that is counter-intuitive is the average time values of the one bot case with obstacles. The average search time actually is less than the two bot case, though there were far fewer successful searches. This implies that the one bot case with obstacles either finds the target relatively quickly, or not at all.

The standard deviation for the test runs is relatively large, especially for the 1 and 2 bot cases, with obstacles. Generally, one really large search time in each set of 10 test runs skewed the average and standard deviation times. For the search with obstacles, in both the 1 and 2 bot cases, there was one test case with a search time in excess of 500 seconds. We attribute these extremely long search times primarily to low battery levels. Low battery levels leads to longer wait times to get clean packets from the beacons (transmission signals from the most-distant beacon are weaker) and then slower movement of the bots.

To overcome the weak signal from one of the beacons, we modified the position algorithm to allow the bot to calculate its position with distance data from only two beacons. The bot initially tried to get two consecutive good measurements from all three beacons. If after seven seconds there was still not good distance information, then the bot would use the distance measurements from two beacons to calculate its (x,y) position.

During our initial experiments, we noticed that the light sensors on the bots were mismatched. That is, different bots would read different values for the same light level. This imbalance led to some stray values for the time to find the target. Specifically, if the bot with the lowest light readings found the peak value first, then other bots would circle toward that point. When another bot (with a light sensor that recorded higher light values) moved to the same location, it would record a higher light value and the algorithm would think a "new" target had been found.

To correct for the different sensor readings, we used linear splines for each individual sensor to adjust and match the light sensor outputs of the three bots. Thus, each bot will give the same (corrected) output for the equivalent light level. The results for 10 test runs with the position and sensor algorithm corrections are shown in Table 2.

TABLE 2: RESULTS FROM USING PSO TO PROGRAM A SUITE OF BOTS WITH SENSOR AND POSITION CORRECTIONS

Runs without obstacles			
Number of bots	Successful runs (out of 10)	Average time of successful runs (sec)	Standard deviation (sec)
1	6	180.2	101
2	10	176.1	68.9
3	10	109.6	55.0

Runs with obstacles			
Number of bots	Successful runs (out of 10)	Average time of successful runs (sec)	Standard deviation (sec)
1	8	250.2	108.6
2	10	181.7	76.1
3	10	125.8	65.2

As expected, the sensor and position corrections improve the average and standard deviation of the search times. The multi-bot test runs are all finding the peak value, even in the presence of obstacles. As with the previous results, more bots leads to more successful searches and faster search times.

V. CONCLUSIONS

We have developed a version of the PSO that "distributes" the processing among several, simple, compact, mobile bots.

We have called our algorithm the distributed PSO (dPSO). In the dPSO algorithm, all the calculations are done "locally", that is, on each local bot. The only data that is potentially needed from other bots is the value and location of the global best, gbest. Thus, there are no communications unless one of the bots finds a point in the search space that is better than any point found up to that time during the search.

We have tested the distributed PSO algorithm using three bots. The tests show that using the PSO to control a swarm can successfully find the target (in this case, the brightest point of light in the search space), even in the presence of obstacles.

REFERENCES

[Bergbreiter 2003] S. Bergbreiter; K. Pister, "CotsBots: An Off-the-Shelf Platform for Distributed Robotics," IROS 2003, Las Vegas, NV, October 27-31, 2003.

[Chang 2005] J. Chang, S. Chu, J. Roddick, J. Pan, "A parallel particle swarm optimization algorithm with communication strategies", Journal of Information Science and Engineering, vol. 21, pp. 809-818, 2005.

[Culler 2004] D. Culler, H. Mulder, "Smart sensors to network the world", Scientific American, pp. 84 – 91, June 2004.

[Doctor 2004] S. Doctor, G. Venayagamoorthy, V. Gudise, "Optimal PSO for collective robotic search applications", IEEE Congress on Evolutionary Computation, Portland, OR, pp. 1390 – 1395, June 2004.

[Eberhart 1995] R. Eberhart, J. Kennedy, "A new optimizer using particle swarm theory", Proceedings of the sixth international symposium on micro machine and human science, Japan, pp. 39-43, 1995.

[Eberhart 2001] R. Eberhart, Y. Shi, "Particle swarm optimization: developments, applications and resources", Proc. congress on evolutionary computation, Korea, 2001.

[Eberhart 2004] R. Eberhart, Y. Shi, Special issue on Particle Swarm Optimization, IEEE Transactions on Evolutionary Computation, pp. 201 – 301, June 2004.

[Hayes 2000] A. Hayes, A. Martinoli, R. Goodman, "Comparing distributed exploration strategies with simulated and real autonomous robots", Proc of the 5th International Symposium on Distributed Autonomous Robotic Systems, Knoxville, TN, pp. 261-270, October 2000.

[Hayes 2002] A. Hayes, A. Martinoli, R. Goodman, "Distributed Odor Source Localization", IEEE Sensors, pp. 260-271, June 2002.

[Hereford 2006] J. Hereford, "A distributed Particle Swarm Optimization algorithm for swarm robotic applications", 2006 Congress on Evolutionary Computation, Vancouver, BC, pp. 6143 – 6149, July 2006.

[Hsiang 2002] T-R Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, "Algorithms for rapidly dispersing robot swarms in unknown environments", Fifth International Workshop on Algorithmic Foundation of Robotics, December 2002.

[Jatmiko 2006] W. Jatmiko, K. Sekiyama, T. Fukuda, "A PSO-based mobile sensor network for odor source localization in dynamic environment: theory, simulation and measurement", 2006 Congress on Evolutionary Computation, Vancouver, BC, pp. 3781 – 3788, July 2006.

[Morlok 2004] R. Morlok, M. Gini, "Dispersing robots in an unknown environment", Distributed Autonomous Robotic Systems 2004, Toulouse, France, June 2004.

[Pugh 2005] J. Pugh, A. Martinoli, Y. Zhang, "Particle Swarm Optimization for unsupervised robotic learning", Proc. of the 2005 IEEE Swarm Intelligence Symposium, Pasadena, CA, June 2005.

[Pugh 2006] J. Pugh, A. Martinoli, "Multi-robot learning with Particle Swarm Optimization", Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, May 2006.

[Spears 2006] W. Spears, J. Hamann, P. Maxim, P. Kunkel, D. Zarzhitsky, D. Spears, C. and Karlsson, "Where are you?", Proceedings of the SAB Swarm Robotics Workshop, September 2006, Rome, Italy.

[Teller 2003] S. Teller, K. Chen, H. Balakrishnan, "Pervasive Pose-Aware Applications and Infrastructure", IEEE Computer Graphics and Applications, July/August 2003.