PARTICLE SWARM OPTIMIZATION FOR UNSUPERVISED ROBOTIC LEARNING

Jim Pugh and Alcherio Martinoli

Swarm-Intelligent Systems Research Group,
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
{jim.pugh, alcherio.martinoli}@epfl.ch

Yizhen Zhang

Engineering Design Research Laboratory,
California Institute of Technology
Pasadena, California 91125, USA
yizhen@caltech.edu

ABSTRACT

We explore using particle swarm optimization on problems with noisy performance evaluation, focusing on unsupervised robotic learning. We adapt a technique of overcoming noise used in genetic algorithms for use with particle swarm optimization, and evaluate the performance of both the original algorithm and the noise-resistant method for several numerical problems with added noise, as well as unsupervised learning of obstacle avoidance using one or more robots.

Index Terms — particle swarm optimization, unsupervised learning, noisy optimization, swarm robotics

1. INTRODUCTION

Particle swarm optimization (PSO) is a promising new optimization technique which models the set of potential problem solutions as a swarm of particles moving about in a virtual search space. The method was inspired by the movement of flocking birds and their interactions with their neighbors in the group. PSO can be used for shaping an Artificial Neural Network (ANN) controller by having the parameter set be the weights, and the evaluative function be a measure of the performance of a desired robot behavior. Thus far, there has been little exploration of how the PSO algorithm is affected by noise in the fitness evaluation, and no work on the effects of non-Gaussian noise, such as that which appears in unsupervised robotic learning.

Human intuition and current engineering design methods are often not well-adapted to design robotic controllers. Depending on computational constraints of the robotic platform and therefore on the type of the controller used, reverse engineering the perception-to-action map for even simple robotic behavior can be a non-trivial process due to the large size of the parameter search space. A solution to this problem is using robust machine-learning techniques.

In this paper, we will explore the performance of PSO in noisy environments, focusing on unsupervised robotic learning, and suggest an augmentation of the original method which can yield superior performance. Section 2 provides some background on PSO, optimization in the presence of noise, and unsupervised robotic learning. Section 3 explains the modifications in the noise-resistant method and evaluates the PSO techniques on several standard test functions with Gaussian noise added. Section 4 evaluates the original PSO and noise-resistant PSO on the case study of unsupervised learning of obstacle avoidance in several scenarios with one or two robots. Section 5 discusses the implications of the results and future work.

2. BACKGROUND

The original PSO method was developed by James Kennedy and Russel Eberhart ([7] [3]). Every particle in the swarm begins with a randomized position ($x_{i,j}$) and randomized velocity ($v_{i,j}$) in the n-dimensional search space, where i represents the particle index and j represents the dimension in the search space. Candidate solutions are optimized by flying the particles through the virtual space, with attraction to positions in the space that yielded the best results. Each particle remembers at which position it achieved its highest performance ($x^*_{i,j}$). Each particle is also a member of some neighborhood of particles, and remembers which particle achieved the best overall position in that neighborhood (given by the index i'). This neighborhood can either be a subset of the particles (local neighborhood), or all the particles (global neighborhood). For local neighborhoods, the standard method is to set neighbors in a pre-defined way (such as using particles with the closest array indices as neighbors modulo the size of the swarm, henceforth known as a "ring topology") regardless of the particles' positions in the search space. The equations executed by PSO at each step of the algorithm are

$$\begin{aligned} v_{i,j} &= w \cdot (v_{i,j} + pw \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) \\ &\quad + nw \cdot rand() \cdot (x^*_{i',j} - x_{i,j})) \\ x_{i,j} &= x_{i,j} + v_{i,j} \end{aligned}$$

where w is the inertia coefficient which slows velocity over time, pw is the weight given to the attraction to the previous best location of the current particle and nw is the weight given to the attraction to the previous best location of the particle neighborhood. $rand()$ is a uniformly-distributed random number in $[0, 1]$.

PSO has been shown to perform as well as or better than genetic algorithms (GA) in several instances. Eberhart and Kennedy found PSO performs on par with GA on the Schaffer f6 function [3, 7]. In work by Kennedy and Spears [8], a version of PSO outperforms GA in a factorial time-series experiment. Fourie showed that PSO appears to outperform GA in optimizing several standard size and shape design problems [6].

Considering the amount of literature on evolutionary algorithms, there have been relatively few publications on dealing with noisy fitness evaluation. Beyer showed that noise causes a decrease in convergence velocity and a residual location error in the final solution [2]. Fitzpatrick and Grefenstette found that in noisy environments, it can be preferable to sacrifice the accuracy of evaluations (by taking few samples of the noisy fitness value) to increase either population size or number of generations of a genetic algorithm [4]. Miller and Goldberg expanded on this idea to find a lower bound for the optimal number of samples [11]. In work by Stagge [15], a modification of GA where only high-performing candidate solutions were resampled is introduced. A similar technique is used in [1] by Antonsson et al. for evolving robots, where the parent population is reevaluated at each iteration.

There has been very little exploration of the effects of noisy fitness evaluation on PSO. Parsopoulos and Vrahatis found that PSO worked much of the time, but with degraded performance on a set of numerical problems with multiplicative Gaussian noise [13]. No modification to the algorithm was suggested to cope with noise.

Evolutionary algorithms have been used extensively for unsupervised learning of robotic behavior. Standard GA has been shown to be effective in evolving simple robotic controllers [5]. To the best of our knowledge, there has been no work on using PSO for unsupervised learning in robotics.

3. NOISE-RESISTANT PSO

We take our inspiration from the modification of GA presented by Antonsson et al. in [1], since the technique is very simple, and it has been used for robot evolution in the past. The workings of the algorithm are described in Fig. 1. High-performing solutions which remain in the population over multiple generations are evaluated multiple times; the final performance is taken to be an aggregate of all evaluations. This should decrease the noise of that candidate solution over time. For combining the multiple evaluations of a high-performing solution, the "aggregation function" described in [14] is used:

$$\mathcal{P}_s(\bar{\mu}) = (\frac{1}{n}\sum_{i=1}^{n}\mu_i^s)^{\frac{1}{s}} \quad (1)$$

where μ_i are the performance values over n evaluations, and s is the "degree of compensation"; s determines how much weight is given to high-performing values versus low-performing values. For instance:

$$\begin{aligned} \mathcal{P}_{-\infty} &= \lim_{s\to-\infty}\mathcal{P}_s = \min(\bar{\mu}) \\ \mathcal{P}_{\infty} &= \lim_{s\to\infty}\mathcal{P}_s = \max(\bar{\mu}) \\ \mathcal{P}_1 &= \text{avg}(\bar{\mu}) \end{aligned}$$

In [1], $\min()$ was used to aggregate the performance values, and so the final performance was the worst performance over all evaluations (GAmin). This ensures that a poor-performing solution which randomly achieved an initial good performance will be removed after very few generations, not allowing it to significantly influence the genetic code of the population.

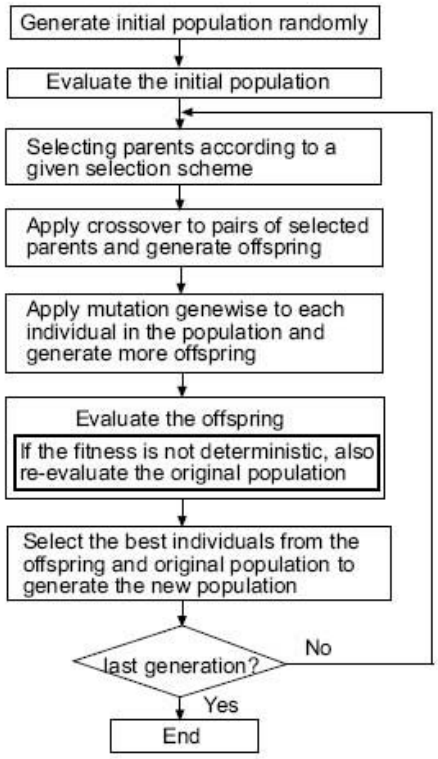

Figure 1. Evolutionary optimization loop used by GA

We can form an analogy between the parent set of GA and the x^* positions in PSO; both are sets of high-performing

candidate solutions which have the potential to exist over multiple iterations of the algorithms. Therefore, we can modify PSO in much the same way as GA: at each iteration of PSO, the x^* positions are reevaluated (see Fig. 2). For determining the final fitness, we explore using both the original method of taking the worst performance (PSOmin) as well as averaging all performances (PSOavg).

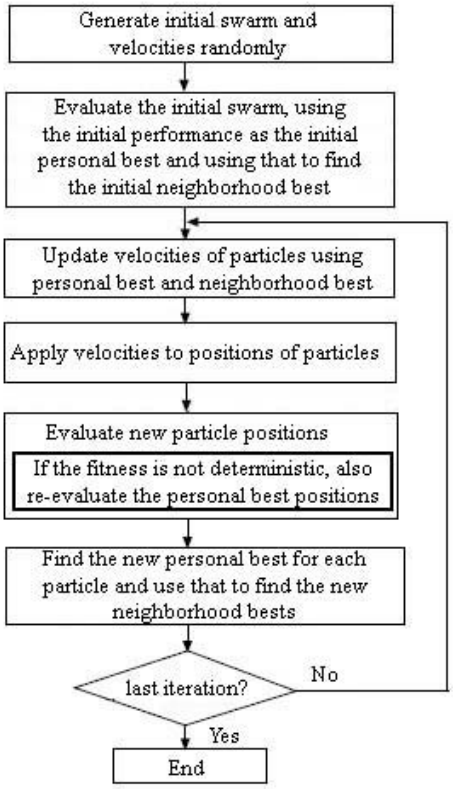

Figure 2. Evolutionary optimization loop used by noise-resistant PSO

3.1. Test Function Evaluation

For an initial evaluation, we try to minimize several standard test functions used in [9] with additive Gaussian noise. We optimize using GA, GAmin, PSO, PSOmin, and PSOavg. Functions can be found in Table 1, along with the number of iterations run on each algorithm. The number of iterations was chosen based on empirical evidence for how long the algorithms took to converge.

The functions are defined as follows:

$$f_1(\bar{x}) = \sum_{i=1}^{n} x_i^2 \quad (2)$$

$$f_2(\bar{x}) = \sum_{i=1}^{n-1} \left[100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\right] \quad (3)$$

Table 1. Test Functions

Function	Function Name	Number of Iterations
f_1	Sphere	400
f_2	Generalized Rosenbrock	20000
f_3	Rastrigin	400
f_4	Griewank	400

$$f_3(\bar{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right] \quad (4)$$

$$f_4(\bar{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) \quad (5)$$

For all functions, $n = 30$, and x_i was constrained to $[-5.12, 5.12]$, the range used in [9]. Noise was introduced by adding a 0-mean Gaussian random variable:

$$f_j'(\bar{x}) = \mathcal{N}(0, \sigma^2) + f_j(\bar{x})$$

We use sigma values of 0.03, 0.5, 1.0, 5.0, and 20.0. The parameters used for GA and PSO are given in Table 2. For the PSO neighborhood, we use a ring topology and assign the nearest particle on each side to be a neighbor. We chose this instead of a global neighborhood because preliminary results showed that better performance was achieved with this type of neighborhood on the tests we are doing. For GA, propagation is done using a Roulette Wheel scheme. Mutation applies a numerical adjustment to a gene, selected uniformly randomly over a fixed range. Once the candidate solutions have been evaluated, the best-performing half of the particles from the children and previous parent set are selected to be the new parent set.

Table 2. GA and PSO Parameters for Test Functions

GA		PSO	
Population Size	20	Swarm Size	20
Crossover Probability	0.6	pw	2.0
Mutation Probability	0.05	nw	2.0
Mutation Range	[-0.5, 0.5]	w	0.6

For GAmin, PSOmin, and PSOavg, because good candidate solutions are reevaluated, twice as many performance evaluations are executed as in standard GA or PSO. To compensate for this, these algorithms only run for half of the listed iterations (e.g.,200 instead of 400 for f_1').

3.2. Test Function Results

The results of the algorithms on the test functions can be seen in Tables 3-6. We see similar results for the first three

functions; with no noise, the standard algorithms perform as well or better than the noise-resistant algorithms, since they execute more iterations. As noise is increased, performance of the standard algorithms rapidly degrades, while the noise-resistant ones maintain reasonable results. PSO seems to outperform GA on f_1' and f_3' with noise, and the performance is comparable on f_2' with noise. PSOmin and PSOavg obtain the best results in the presence of noise, with neither offering clear superior performance; PSOmin does slightly better on f_3' and PSOavg does slightly better on f_1'.

The performance on f_4' differs from that of the other functions. Performance without noise is similar, with all functions converging to values near zero. The function appears very sensitive to noise, however, and all algorithms achieve very poor performances with standard deviation of 0.5 or higher. With standard deviation of 0.03, only standard PSO and PSOavg perform fairly well, with PSOavg outperforming PSO.

Table 3. Performance on f_1' over 20 runs (mean/standard deviation)

σ	GA	GAmin	PSO	PSOmin	PSOavg
0.0	0.02/0.01	0.80/0.51	0.00/0.00	0.04/0.06	0.05/0.04
0.03	0.12/0.03	0.69/0.55	0.07/0.03	0.14/0.20	0.09/0.05
0.5	1.06/0.24	1.82/1.35	1.13/0.48	0.59/0.25	0.59/0.58
1.0	2.95/0.65	3.80/1.96	2.01/0.84	1.15/0.41	1.08/0.65
5.0	50.0/10.4	26.1/6.59	6.82/1.63	4.43/0.90	3.79/1.46
20.0	115/20.4	69.5/16.1	19.1/3.69	14.6/2.91	10.2/1.85

Table 4. Performance on f_2' over 20 runs (mean/standard deviation)

σ	GA	GAmin	PSO	PSOmin	PSOavg
0.0	34.6/18.9	33.1/19.6	7.38/3.27	13.3/7.35	15.2/6.32
0.03	44.8/27.5	38.8/23.9	42.4/25.9	28.9/21.5	23.8/14.6
0.5	60.4/30.7	47.1/45.8	69.4/47.0	25.0/16.4	42.2/29.0
1.0	67.2/42.2	45.0/29.6	66.9/41.2	27.2/18.2	38.2/25.3
5.0	111/70.0	52.7/37.0	87.4/41.0	38.2/24.0	36.8/23.5
20.0	133/62.8	49.8/34.3	134/83.8	47.2/22.4	48.5/24.6

4. CASE STUDY

4.1. Unsupervised Robotic Learning

Our case study is to evolve high-performing controllers for robots learning obstacle avoidance behavior in an unsupervised fashion. Unsupervised robotic learning is an interesting test, because it is inherently noisy, due to sensor and actuator noise and the local perception of the robots. The noise is not necessarily Gaussian, and therefore may yield significantly different results than a scenario with Gaussian noise. We use Webots, an embodied simulator, for our robotic simulations [10], using the Khepera robot model [12]. The robot(s) operate in a 1.0 m x 1.0 m square arena, with the corners cut off by small diagonal blocks (see Fig. 3). The robotic controller is a single-layer discrete-time artificial neural network of two neurons, one for each wheel speed, with sigmoidal output functions. The inputs are the eight proximity sensors (six in front, two in back), as well as a recursive connection from the previous output of the neuron and lateral inhibitions (see Fig. 4). Sensors have a maximum range of 5.0 cm, and sensor output varies linearly from 0.0 at maximum range to 5.11 at minimum range (0.0 cm) with 10% noise. Slip noise of 10% is applied to the wheel speed. The time step for neural updates is 128 ms. We base our fitness function for obstacle avoidance on that proposed by Floreano and Mondada [5]. The fitness function is given by:

Table 5. Performance on f_3' over 20 runs (mean/standard deviation)

σ	GA	GAmin	PSO	PSOmin	PSOavg
0.0	157/21.8	154/24.4	48.3/14.4	78.0/19.9	89.4/40.6
0.03	160/26.7	151/33.8	61.0/20.2	89.3/46.1	76.9/30.5
0.5	170/34.5	169/32.9	56.3/18.7	82.9/39.9	72.0/38.5
1.0	163/20.0	175/28.9	65.5/30.3	82.1/35.5	87.9/39.6
5.0	180/32.2	178/38.0	62.1/11.0	77.8/39.3	93.0/41.7
20.0	213/29.0	182/24.9	108/26.8	76.1/16.1	96.7/41.8

Table 6. Performance on f_4' over 20 runs (mean/standard deviation)

σ	GA	GAmin	PSO	PSOmin	PSOavg
0.0	0.01/0.01	0.07/0.04	0.01/0.03	0.01/0.01	0.01/0.01
0.03	1.04/0.01	0.95/0.21	0.21/0.26	0.68/0.45	0.10/0.09
0.5	1.06/0.01	1.06/0.01	1.04/0.01	1.12/0.01	1.09/0.01
1.0	1.06/0.01	1.07/0.01	1.04/0.01	1.12/0.01	1.10/0.01
5.0	1.06/0.01	1.06/0.01	1.04/0.01	1.12/0.01	1.10/0.01
20.0	1.06/0.01	1.06/0.01	1.04/0.01	1.13/0.01	1.10/0.01

$$F = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i)$$
$$0 \leq V \leq 1$$
$$0 \leq \Delta v \leq 1$$
$$0 \leq i \leq 1$$

where V is the average absolute wheel speed of both wheels, Δv is the average of the difference between the wheel speeds, and i is the average activation value of the most active proximity sensor over the evaluation period. These factors reward robots that move quickly, turn as little as possible, and spend little time near obstacles, respectively. The evaluation period of the fitness tests for these experiments is 240 steps, or approximately 30 seconds. Between each fitness test, the position and bearing of the robots are randomly set by the simulator to ensure the randomness of the next evaluation.

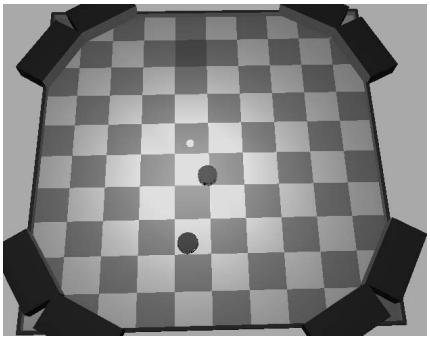

Figure 3. Robot arena with Khepera robots

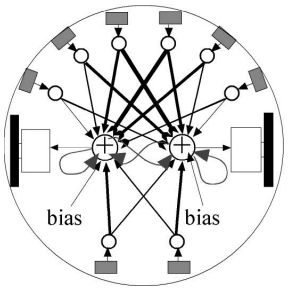

Figure 4. Representation of the artificial neural network used for the robot controller. Grey boxes represent proximity sensor inputs while the white boxes on the side represent the motor outputs. Curved arrows are recurrent connections and lateral inhibitions.

The parameters we use for GA and PSO for the unsupervised robotic learning are given in Table 7. These parameters are based upon those used in [1] for GA and [9] for PSO and adjusted based on initial empirical results to improve performance. For the PSO neighborhood, we again use a ring topology and assign the nearest particle on each side to be a neighbor. The initial neural weights are uniformly randomly generated within [-20, 20], but their value is not limited to that range by the algorithms.

We tested the unsupervised learning in three different scenarios. In scenario 1, there is a single robot evolving the controller in the arena, with no other obstacles. In scenario 2, there is again a single robot evolving the controller, but there is also another robot running a pre-evolved high-performance obstacle avoidance routine; we suspect this may increase the noise of the evolution, as the second robot acts as a moving obstacle. In scenario 3, there are two robots jointly evolving a common shared controller. We assume that all performance information can be communicated between the robots, and that they can and do evaluate different candidate solutions simultaneously. This should allow for twice the overall evaluation speed of a single robot, and thus twice the evolution speed, but may have a penalty in further increasing the noise of the evolution.

Table 7. GA and PSO Parameters for Unsupervised Learning

GA		**PSO**	
Population Size	60	Swarm Size	60
Crossover Probability	0.2	pw	2.0
Mutation Probability	0.15	nw	2.0
Mutation Range	[-5.0, 5.0]	w	0.6

4.2. Unsupervised Robotic Learning Results

The results for the best evolved controllers can be seen in Fig. 5. Best controllers were selected by evaluating every controller in the final population/swarm five times and selecting the controller with the best average performance. This controller was then evaluated 30 times, and the final best performance taken as the average of these performances. For all charts, error bars represent the standard deviation of performances over the different evolutions (20 evolutions per algorithm per scenario).

For all algorithms, performance consistently decreases from scenario 1 to scenario 2 to scenario 3. This follows our predictions about the increased noise. GAmin outperforms GA on all three scenarios. In Scenario 1, GA would sometimes converge to very good solutions (fitness of 0.7) and sometimes to poor solutions (fitness of 0.3), causing a large standard deviation in performance. Performances of the different PSO methods are fairly comparable; it is only in Scenario 3 that we see a distinct improvement using noise-resistant methods. GAmin and PSOmin achieve similar performance in all scenarios. PSOavg performs the best on all three scenarios, but within the margin of error of the other noise-resistant methods. Also, it should be noted that the performance of PSOavg on Scenario 3, although less than Scenario 1, is within the margin of error.

It is useful to observe the average performance of the population/swarm over the evolution process to gain some insight into the workings of the algorithms. Fig. 6 shows

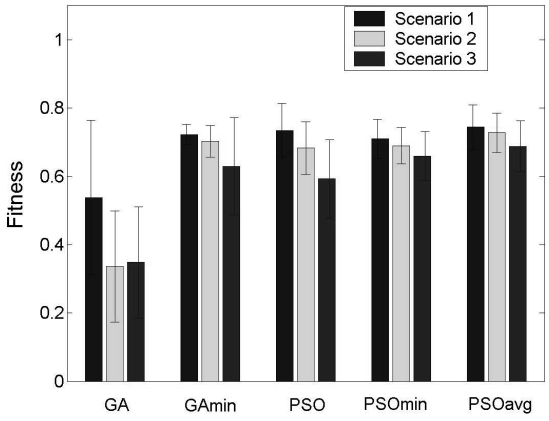

Figure 5. Average of final best performance over 20 evolutions

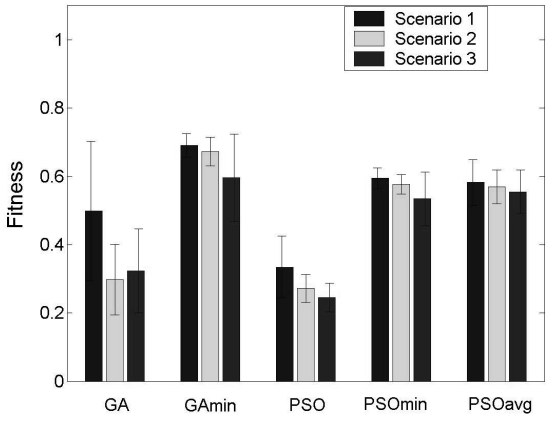

Figure 6. Average performance of final population/swarm over 20 evolutions

the average performance of the final population/swarm for each algorithm; this value was obtained by averaging the final performances of all elements in the population/swarm, without any additional reevaluations.

The average population performances of both GA methods closely reflect the best performances. However, there is a large difference between the average swarm performance and the best performance in all PSO methods. This is most pronounced in standard PSO. The likely cause of this is the local neighborhoods in PSO, which would allow for a heterogenous swarm to be maintained over the evolutionary process. This results in a large range of performances in the swarm at the end of evolution, which causes the average performance to be significantly lower than the best. GA, on the other hand, has no feature to maintain diversity, and all chromosomes will tend to converge to a homogenous solution. This will cause the average performance to closely resemble the best performance.

We can see the progression of the average population/ swarm performance over the evolutionary process in Fig. 7-10. For these figures, "step" refers to one iteration of the noise-resistant algorithms or two iterations of the standard algorithms, as the standard algorithms were run for twice as many iterations. We see in Fig. 7 and 8 that while the standard algorithms cease their evolutionary progress after few iterations, the noise-resistant algorithms continue throughout the entire process. In the cases of PSOmin and PSOavg, this would allow the average swarm performance to more closely approach the best performance, which would explain why the difference is most pronounced in standard PSO. Fig. 9 and 10 show that GA has greater growth in the early iterations of the evolution. From Fig. 10, however, we see that noise-resistant PSO has greater growth in the later stages.

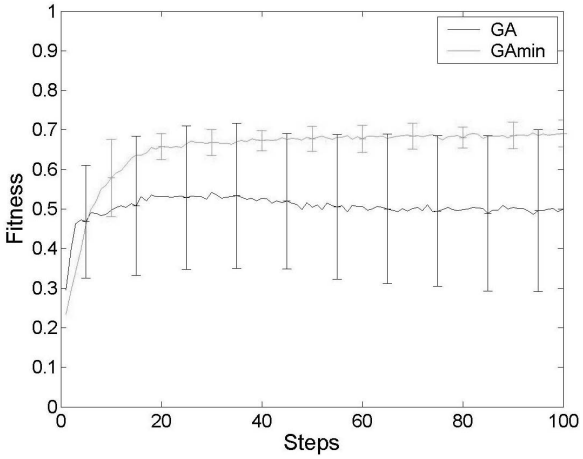

Figure 7. GA vs. GAmin, average performance of population over 20 evolutions in Scenario 1

5. DISCUSSION

Standard GA had unexpectedly poor performance in the unsupervised learning task, especially since it was shown to yield very good results in [5]. We suspect this may be due to the fast convergence rate of the algorithm; if all candidate solutions converge to homogeneity before much exploration is done, it increases the chance of pre-converging to a medium-performance local maximum as opposed to one with good performance. This problem was overcome in the noise-resistant version, since it continues to converge over the entire evolution, and was not present in PSO because of its slower convergence rate.

Although PSOmin and PSOavg offered comparable performances overall, there were various situations in which

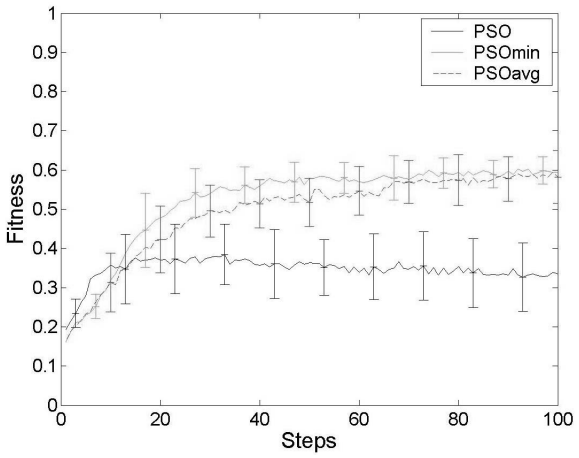

Figure 8. PSO vs. PSOmin vs. PSOavg, average performance of swarm over 20 evolutions in Scenario 1

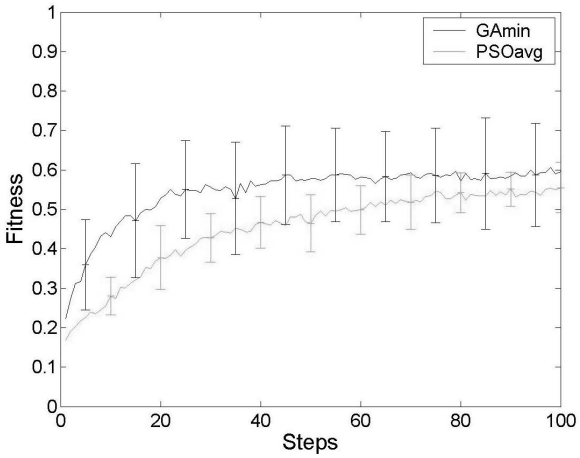

Figure 10. GAmin vs. PSOavg, average performance of population/swarm over 20 evolutions in Scenario 3

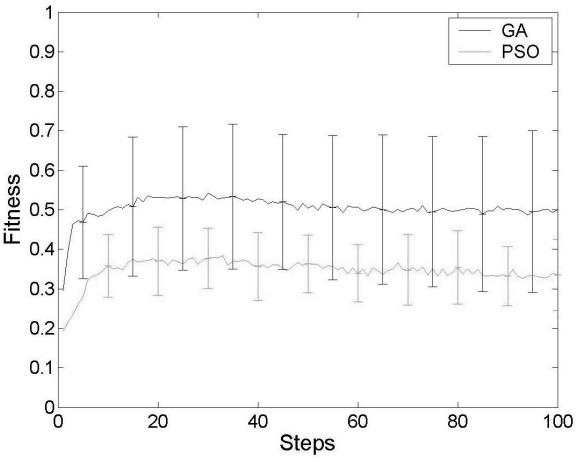

Figure 9. GA vs. PSO, average performance of population/swarm over 20 evolutions in Scenario 1

one or the other performed slightly better. What caused these variations is not clear. We hypothesize that in situations in which we are concerned only with false positives (poor solutions masquerading as good ones), PSOmin will offer superior performance, as it can more quickly correct these mistakes and remove the poor solutions. It also ensures that final solutions will be very robust and not susceptible to occasional poor performances. However, in situations with both false positives and false negatives (good solutions that get a poor performance due to external factors, e.g.,being boxed into a corner by another robot), it could be preferable to use PSOavg, since PSOmin could remove very good solutions if they happened to score badly once.

Although not explored here, PSO could offer an additional benefit over GA for unsupervised learning in multi-robot systems. The population manager for GA must use all candidate solutions to create the next generation of the population, requiring global knowledge of their performances. In PSO with a local neighborhood, a particle can be updated knowing only the performances of several other particles. This gives PSO the potential to be implemented efficiently in a distributed fashion without a global supervisor. In multi-robot systems, communication is often very expensive and sometimes not possible when there is a large distance or obstacles between robots. PSO could be adapted to allow robots to use other robots in the physical proximity as neighbors, and thereby implement a distributed version of the algorithm on a robotic swarm, with each robot responsible for one or more particles. Since we have shown that unsupervised learning can be accomplished by multiple robots simultaneously, this could prove an effective technique for very fast evolution of robotic controllers. It remains to see how this type of neighborhood will affect the optimization process.

6. CONCLUSION AND OUTLOOK

Adding the noise-resistance modification to PSO results in an improvement very similar to what is observed in GA when dealing with noisy fitness evaluations. For very low noise values or very few iterations, the standard algorithms perform better, since more evaluations are required for the noise-resistant versions, making them slower. However, as noise increases, the standard algorithms will prematurely converge on poorer results. The noise-resistant algorithms continue to improve throughout the full evolutionary process, allowing them to avoid the major decrease in performance and offer better results than the standard algorithms in these situations. This indicates that the noise-resistant

algorithm is able to at least partially overcome the convergence error described in [2].

While it had previously been shown in [5] that genetic algorithms could be used for unsupervised robotic learning, we have shown that particle swarm optimization can also be used for this purpose. PSO had comparable performance to GA on the unsupervised learning task described in this paper. Although the results obtained by PSO were superior to those obtained by GA in several instances, this could be partially due to parameter settings of the algorithms, and is therefore not conclusive evidence that PSO would consistently work better in these situations.

We hope to continue to expand on the work in this area. The costs and benefits of different aggregation functions for combining performance values needs to be explored. Unsupervised learning with more than 2 robots and using different methods of sharing information would allow us to assess the scalability potential of swarm unsupervised learning. We also want to explore moving the implementation of the PSO algorithm closer to one which could be used effectively with real robots.

7. ACKNOWLEDGEMENTS

Jim Pugh and Alcherio Martinoli are currently sponsored by a Swiss NSF grant (contract Nr. 11 P00268647/1). Yizhen Zhang is currently sponsored by the Engineering Research Centers Program of the National Science Foundation under Award Number EEC-9402726.

8. REFERENCES

[1] Antonsson E. K, Zhang Y., & Martinoli A. "Evolving Engineering Design Trade-Offs". Proc. of the ASME Fifteenth Int. Conf. on Design Theory and Methodology, September 2003, Chicago, IL.

[2] Beyer, H.-G. "Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice". Computer Methods in Mechanics and Applied Engineering, 2000, pages:239-267, vol.186.

[3] Eberhart, R. & Kennedy, J. "A new optimizer using particle swarm theory" Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on, Vol., Iss., 4-6 Oct 1995, pages:39-43

[4] Fitzpatrick, J. M. & Grefenstette, J. J. "Genetic Algorithms in Noisy Environments" Machine Learning: Special Issue on Genetic Algorithms, 1988, pages:101-120, vol.3.

[5] Floreano, D. & Mondada, F. "Evolution of Homing Navigation in a Real Mobile Robot" Systems, Man and Cybernetics, Part B, IEEE Transactions on, Vol.26, Iss.3, Jun 1996, pages:396-407

[6] Fourie, P. C. & Groenwold, A. A. "The particle swarm optimization algorithm in size and shape optimization" Struct. Multidisc. Optim., 2002, pages:259-267 vol.23

[7] Kennedy, J. & Eberhart, R. "Particle swarm optimization" Neural Networks, 1995. Proceedings., IEEE International Conference on, Vol.4, Iss., Nov/Dec 1995, pages:1942-1948 vol.4

[8] Kennedy, J. & Spears, W. M. "Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator" in Proceedings of IEEE International Conference on Evolutionary Computation, Anchorage, May 1998, pp. 78-83.

[9] Kennedy, J. & Eberhart, R. Swarm Intelligence, Morgan Kaufmann Academic Press, 2001.

[10] Michel, O. "Webots: Professional Mobile Robot Simulation" Int. J. of Advanced Robotic Systems, 2004, pages:39-42, vol.1

[11] Miller, B. L. & Goldberg, D. E. "Optimal Sampling for Genetic Algorithms" Intelligent Engineering Systems Through Artifical Neural Networks, 1996, pages:291-298, vol.6.

[12] Mondada, F., Franzi, E. & Ienne, P. "Mobile robot miniaturisation: A tool for investigation in control algorithms" Proc. of the Third Int. Symp. on Experimental Robotics, Kyoto, Japan, October, 1993, pages:501-513

[13] Parsopoulos, K. E. & Vrahatis, M. N. "Particle Swarm Optimizer in Noisy and Continuously Changing Environments" M.H. Hamza (Ed.) , Artificial Intelligence and Soft Computing, IASTED/ACTA Press, 2001, pages:289-294

[14] Scott, M. J., & Antonsson, E. K. "Aggregation functions for engineering design trade-offs". Fuzzy Sets and Systems, 99 (3) November 1998, pp. 253264.

[15] Stagge, P., "Averaging efficiently in the presence of noise" Parallel Problem Solving from Nature, PPSN 5, LNCS 1498, 1998, pages:188-197.

[16] Winfield, A.F.T. & Holland, O.E. "The application of wireless local area network technology to the control of mobile robots", Microprocessors and Microsystems, 2000, Vol. 23, pp. 597-607.