# Distributed Particle Swarm Optimization using Optimal Computing Budget Allocation for Multi-Robot Learning

Ezequiel Di Mario, Iñaki Navarro and Alcherio Martinoli
Distributed Intelligent Systems and Algorithms Laboratory
School of Architecture, Civil and Environmental Engineering
École Polytechnique Fédérale de Lausanne
{ezequiel.dimario, inaki.navarro, alcherio.martinoli}@epfl.ch

***Abstract*—Particle Swarm Optimization (PSO) is a population-based metaheuristic that can be applied to optimize controllers for multiple robots using only local information. In order to cope with noise in the robotic performance evaluations, different re-evaluation strategies were proposed in the past. In this article, we apply a statistical technique called Optimal Computing Budget Allocation to improve the performance of distributed PSO in the presence of noise. In particular, we compare a distributed PSO OCBA algorithm suitable for resource-constrained mobile robots with a centralized version that uses global information for the allocation. We show that the distributed PSO OCBA outperforms a previous distributed noise-resistant PSO variant, and that the performance of the distributed PSO OCBA approaches that of the centralized one as the communication radius is increased. We also explore different parametrizations of the PSO OCBA algorithm, and show that the choice of parameter values differs from previous guidelines proposed for stand-alone OCBA.**

## I. Introduction

There are several sources of randomness that make performance evaluations of robotic controllers inherently noisy. In addition to the obvious sensor and actuator noise, there are other factors such as varying initial conditions, manufacturing tolerances, or changes in the environment that can increase the uncertainty in performance measurements.

Population-based learning techniques have been proven to be effective in dealing with noise in fitness evaluations [1]. Within this family of algorithms, we can find examples on the successful performance under noise for Particle Swarm Optimization [2], [3], Genetic Algorithms [4], and Evolutionary Strategies [5]. Therefore, these techniques are promising tools for the design of high-performing robotic controllers.

We focus this research on the PSO algorithm [6], which allows a distributed implementation in each robot, speeding up the optimization process and adding robustness to failure of individual robots. PSO has been applied to several problems in the robotics domain, such as odor source localization [7], [8], robotic search [9], and obstacle avoidance [10].

Regarding the influence of noise on PSO, Parsopoulos and Vrahatis showed that standard PSO was able to cope with noisy and continuously changing environments, and even suggested that noise may help to avoid local minima [2]. Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions and for certain scenarios of limited-time learning in presence of noise [10], [11].

In this article, we describe a distributed noise-resistant PSO algorithm for multi-robot learning based on Optimal Computing Budget Allocation (OCBA), a statistical sample allocation method introduced by Chen et al. [12]. OCBA has previously been applied to PSO on numerical benchmark functions in a centralized manner [3], [13], [14], where it outperformed other techniques for dealing with noise. Here, we will describe how we apply OCBA to PSO for a multi-robot task in a centralized manner, and then introduce a distributed version which requires only local information and communication. We also propose OCBA parameter values that differ from the guidelines for stand-alone OCBA based on experimental results in simulation.

OCBA has also been applied in the context of evolutionary algorithms [15], [16]. Schmidt et al. [15] used OCBA on an evolutionary algorithm for ranking selection on noisy functions. They allocated as many evaluations as needed to obtain an arbitrary quality of the estimations. In [16], the authors used a version of OCBA to select a set of best potential solutions on each iteration of an Evolution Strategy (ES) algorithm and showed with experimental results the effect of different values of the OCBA parameter initial number of samples ($n_0$), which we also analyze in this article. The difference between the ES OCBA and PSO OCBA is that in the ES it is only needed to estimate the performance of the subset with the best solutions while PSO requires to estimate the performance of all candidate solutions.

The idea behind the application of OCBA to PSO is to fix some issues of previous algorithms which affect their performance under the presence of noise [17]. Standard PSO has no explicit mechanism for dealing with noise. The naïve approach of evaluating every new candidate a fixed number of times results in a better performance estimation for new candidates, but invests as many resources in good candidates as in poor ones which could be immediately discarded [13]. Another disadvantage of this method is that the number of repetitions of each evaluation is fixed and should be selected based on the amount of noise, which must be known in advance. The

noise-resistant approach that evaluates best candidates multiple times [10] has the advantage of placing more computation on the most promising solutions and therefore achieves a high performance, but it is sensitive to "lucky" good evaluations of bad new solutions, which might displace a consistently better old solution, generating random performance drops during the learning [17].

OCBA automatically adjusts the evaluation budget between old and new solutions to maximize the probability of correct selection of good candidates. In addition, as the iterations increase, good candidates tend to accumulate a large number of samples, thereby producing accurate performance estimates of the best solutions, and leaving a larger proportion of the allocation budget to accurately test new candidates.

The remainder of this article is organized as follows. Section II provides some background on PSO and OCBA in order to facilitate the explanation of the subsequent algorithms. Section III describes the algorithms incorporating OCBA to PSO for multi-robot implementations and their differences with the ones from the literature. In Section IV, we describe the obstacle avoidance task that will be used to compare the algorithms and controllers. Section V presents the results from applying the different algorithms and parametrizations for learning in simulation. Finally, Section VI concludes the paper.

## II. Background

PSO is a relatively new metaheuristic originally introduced by Kennedy and Eberhart [6], which was inspired by the movement of flocks of birds and schools of fish. PSO is well suited for distributed implementations due to its distinct individual and social components and the use of the neighborhood concept [10]. It models candidate solutions as a swarm of particles moving in a high-dimensional space. Each particle stores its own personal best position and the position of the best in its neighborhood, which are used to guide the particle's movement.

At each iteration, the position of particle $i$ in dimension $j$ ($x^*_{i,j}$) is updated by adding a velocity $v_{i,j}$ (Eq. 1). This velocity depends on three components: the velocity at the previous iteration weighted by an inertia coefficient $w_I$, a randomized attraction to its personal best $x^*_{i,j}$ weighted by $w_P$, and a randomized attraction to the neighborhood's best $x^*_{i',j}$ weighted by $w_N$ (Eq. 2). $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$x_{i,j} := x_{i,j} + v_{i,j} \quad (1)$$

$$v_{i,j} := w_I \cdot v_{i,j} + w_P \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_N \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \quad (2)$$

The pseudocode for PSO is shown in Fig. 1.

Pugh et al. [11] introduced a distributed, noise-resistant, averaging variation of PSO which operates by re-evaluating personal best positions and averaging them with the previous evaluations. We will refer to this variant as *PSO pbest* and

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate new particle position
5:       Share personal best within neighborhood
6:       Update particle position (Eqs. 1 and 2)
7:    end for
8: end for
```

Fig. 1. Pseudocode for the standard PSO algorithm.

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate particle new particle position
5:       Re-evaluate personal best
6:       Aggregate personal best with previous best
7:       Share personal best within neighborhood
8:       Update particle position (Eqs. 1 and 2)
9:    end for
10: end for
```

Fig. 2. Pseudocode for the *PSO pbest* algorithm.

compare it to our proposed distributed PSO OCBA. The pseudocode for *PSO pbest* is shown in Fig. 2. The difference with the standard PSO pseudocode is the addition of lines 5 and 6.

OCBA is a technique based on Bayesian statistics for allocating samples to different candidate solutions introduced by Chen et al. [12]. Given $k$ candidates with means $\{\bar{X}_1, \ldots, \bar{X}_k\}$ and variances $\{\sigma_1^2, \ldots, \sigma_k^2\}$, and a total number of samples $T$, OCBA aims at maximizing the probability of correct selection $P\{CS\}$ of candidate $b$ as the best (in a minimization problem, the one with the lowest mean):

$$P\{CS\} = P\{\bar{X}_B < \bar{X}_i, i \neq b\} \quad (3)$$

by applying the following allocation rules:

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i/\delta_{b,i}}{\sigma_j/\delta_{b,j}}\right)^2, i \neq j \neq b \quad (4)$$

$$N_b = \sigma_b \sqrt{\sum_{i=1, i\neq b}^{k} \frac{N_i^2}{\sigma_i^2}} \quad (5)$$

where $N_i$ is the number of samples for candidate $i$, and $\delta_{i,j} = \bar{X}_i - \bar{X}_j$ the difference between the means of candidate $i$ and candidate $j$. An intuitive way of interpreting Equations 4 and 5 is that candidate $i$ will get more samples $N_i$ when it has larger variance $\sigma_i^2$ and when its mean is closer to the mean of the best solution found so far (small $\delta_{b,i}^2$). To switch the type of problem from minimization to maximization, we can

TABLE I
PARAMETERS COMMON TO ALL PSO ALGORITHMS

| Parameter | Value |
|---|---|
| Number of robots $N_{rob}$ | 4 |
| Population size $N_p$ | 24 |
| Evaluation span $t_e$ | 30 s |
| Personal weight $w_P$ | 2.0 |
| Neighborhood weight $w_N$ | 2.0 |
| Neighborhood size $N_n$ | 3 |
| Dimension $D$ | 24 |
| Inertia $w_I$ | 0.8 |
| $V_{max}$ | 20 |

TABLE II
PARAMETERS FOR *PSO ocbaC* AND *PSO ocbaD*

| Parameter | *PSO ocbaC* | *PSO ocbaD* |
|---|---|---|
| Iterations $N_i$ | 50 | 50 |
| Iteration budget $B_i$ | 240 | 10 |
| Initial number of samples $n_0$ | 2 | 2 |
| Additional number of samples $\Delta$ | 4 | 1 |

simply consider $\bar{X}_i = -\bar{X}'_i$ where $\bar{X}'_i$ corresponds to the mean of the maximization problem.

This allocation procedure has been proven to be optimal in the sense that it maximizes an asymptotic approximation to the probability of correct selection $P\{CS\}$ as the number of samples tends to infinity, but it was also shown to be very efficient for limited sampling budgets in numerical experiments [12].

## III. LEARNING ALGORITHMS

In this section, we will describe how we incorporate OCBA into PSO in two variants:

- *PSO ocbaC*: a centralized variant where the OCBA allocation is performed with full information,
- *PSO ocbaD*: a distributed variant where each particle performs its own OCBA allocation with information from its local neighborhood.

We will then compare their performances with *PSO pbest*, the noise-resistant variant by Pugh et al. [10] that re-evaluates personal bests at each iteration.

Table I shows the parameters that are common to all PSO algorithms used in this article. They are set following the guidelines for limited-time adaptation presented in [18].

The pseudocode for the centralized version of PSO OCBA, *PSO ocbaC*, is shown in Fig. 3. The main difference from the standard PSO shown in Fig. 1 is the evaluation step, which now involves an allocation procedure using OCBA (Fig. 3 lines 3 to 12). First, $n_0$ samples of the new positions are taken to estimate their mean and variance. Then the remaining samples are allocated among all the new positions and all the personal bests using OCBA Equations 4 and 5. Note that since all personal bests were new positions at some time, they already have at least $n_0$ samples at the moment of the OCBA allocation. The parameters for *PSO ocbaC* are shown in Table II.

The pseudocode for the distributed version of PSO OCBA, *PSO ocbaD*, is shown in Fig. 4. In this case, each particle is

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate new particle position n_0 times
5:    end for
6:    remaining budget := iteration budget - n_0 · N_p
7:    while remaining budget> 0 do
8:       Allocate Δ samples among current positions and
         personal bests using OCBA
9:       Evaluate allocated samples
10:      Recalculate mean and variance for new evaluations
11:      remaining budget := remaining budget - Δ
12:   end while
13:   for N_p particles do
14:      Update personal best
15:      Update neighborhood best
16:      Update particle position
17:   end for
18: end for
```

Fig. 3. Pseudocode for the *PSO ocbaC* algorithm.

running its own algorithm, so the pseudocode is written from the point of view of an individual particle. First, the particle takes $n_0$ samples of its new position in order to estimate its mean and variance. Next, the particle collects the mean, variance, and number of samples of all candidates in the neighborhood (new positions and personal bests). In our case, for comparison purposes, we are using the same neighborhood size as Pugh et al. [10], which is one neighbor on each side of a ring topology. Then, the particle allocates the remaining budget among the shared new positions and personal bests in the neighborhood (in this case, 6 candidates in total: own position, own personal best, 2 shared new positions, and 2 shared personal bests) using OCBA Equations 4 and 5. Finally, the particle evaluates the candidates with the number of samples given by the OCBA allocation and shares the results in the neighborhood.

The parameters for *PSO ocbaD* are displayed alongside those for *PSO ocbaC* in Table II. The main difference is that since each particle is performing its own OCBA allocation, the iteration budget $B_i$ for that allocation is $1/24$ the budget of the centralized version, and the additional number of samples $\Delta$ is reduced to 1 in order to share and receive the results from other particles after each evaluation.

The information shared by each particle is the mean, variance, and sample size of its current position and personal best position, which are required to compute the OCBA allocation. The mean, variance, and sample size can be calculated online incrementally every time a new sample is added using the

```
Initialize particle
for N_i iterations do
    Evaluate new particle position n_0 times
    Share evaluation results in neighborhood
    Receive and store evaluation results from neighborhood
    remaining budget := iteration budget - n_0 · N_p
    while remaining budget> 0 do
        Allocate Δ samples among current positions and personal bests in neighborhood using OCBA
        Evaluate allocated samples
        Recalculate mean and variance for new evaluations
        Share evaluation results in neighborhood
        Receive and store evaluation results from neighborhood
        remaining budget := remaining budget - Δ
    end while
    Update personal best
    Update neighborhood best
    Update particle position
end for
```

Fig. 4. Pseudocode for the *PSO ocbaD* algorithm.

following equations:

$$\bar{X}_n = \frac{(n-1)\bar{X}_{n-1}+X_n}{n} \quad (6)$$

$$\sigma_n^2 = \frac{(n-2)}{(n-1)}\sigma_{n-1}^2 + \frac{(x_n-\bar{x}_{n-1})^2}{n} \quad (7)$$

Therefore, the history from previous evaluations can be incorporated without the need to store or share the entire vector of samples, which can become large towards the end of the learning, especially in the case of good solutions (e.g., we have observed several runs where the best solution had more than 100 samples at the end). Thus, the memory and communication requirements for the distributed algorithm are significantly reduced and they remain constant for the entire learning process.

## IV. Benchmark Task

In order to test the algorithms discussed in Section III we use obstacle avoidance as a benchmark task. We chose this task because of its popularity in the robotic learning literature [10], [19]–[22], and because of its noisy performance evaluations which pose a challenge to the learning algorithms [17].

We use a metric of performance based on the work of [19], which is present in several studies on learning obstacle avoidance (e.g., [10], [21], and our own previous work [22]). The fitness function consists of three factors, all normalized to the interval [0, 1]:

$$f = f_v \cdot (1-\sqrt{f_t}) \cdot (1-f_i) \quad (8)$$

$$f_v = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \frac{|v_{l,k}+v_{r,k}|}{2} \quad (9)$$

$$f_t = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \frac{|v_{l,k}-v_{r,k}|}{2} \quad (10)$$

$$f_i = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} i_{max,k} \quad (11)$$

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step $k$, $i_{max,k}$ is the normalized proximity sensor activation value of the most active sensor at time step $k$, and $N_{eval}$ is the number of time steps in the evaluation period. This function rewards robots that move quickly ($f_v$), turn as little as possible ($f_t$), and stay away from obstacles ($f_i$). Each factor is calculated at each time step and then the product is averaged for the total number of time steps in the evaluation period.

Our experimental platform is the Khepera III, a differential wheeled robot with a diameter of 12 cm. It is equipped with nine infra-red sensors for short range obstacle detection, which in our case are the only external inputs for the controllers, and two wheel encoders, which are used to measure the wheel speeds for the fitness calculations.

The comparison between algorithms presented in this paper is performed only in simulation using Webots [23], a high-fidelity robotic simulator that models dynamical effects such as friction and inertia. The accuracy of the simulation in this setup has been previously validated with experiments on real robots in [18].

We conduct experiments in a square arena of 2 m x 2 m with walls, where 15 cylindrical obstacles of diameter 10 cm are randomly placed at the beginning of each fitness evaluation. The initial robots' positions at the beginning of each evaluation are also set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. The setup can be seen in Fig. 5, which shows 4 robots in the arena with 15 obstacles.

The optimization problem to be solved by the PSO learning algorithms is to choose the set of weights of an artificial neural network (ANN) controller such that the fitness function $f$ as defined in Eq. 8 is maximized. Since the ANN is a general controller architecture that was not specifically engineered for this task, it is the PSO algorithm that decides the behavior of the robot, which is why we describe the resulting behavior as learned.

The artificial neural network has a recurrent architecture consisting of two units with sigmoidal activation functions. The outputs of the units determine the wheel speeds. Each neuron has 12 input connections: the 9 infrared sensors, a connection to a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output, resulting in 24 weight parameters in total.

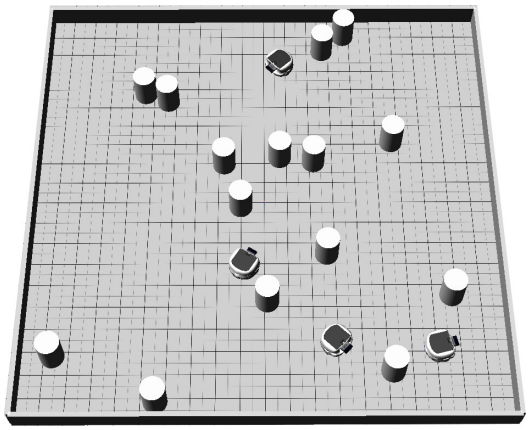


Fig. 5. Arena with 15 obstacles and four Khepera III robots performing one of the obstacle avoidance algorithms learned.

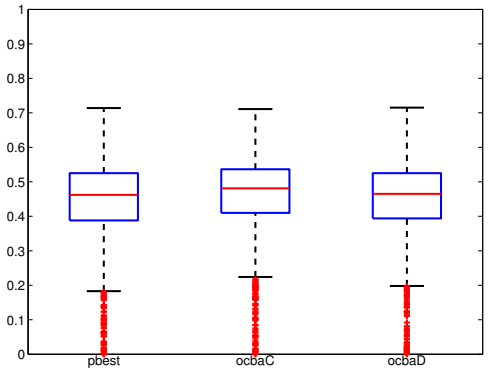



Fig. 6. Final ground truth performance with default parameter values for *PSO pbest*, *PSO ocbaC*, and *PSO ocbaD*. The box represents the upper and lower quartiles, the line across the middle marks the median, the bars extend to the most extreme data points not considered outliers, and the red crosses show outliers.

These 24 parameters define the dimensionality of the learning space of the algorithms.

Each particle evaluation consists of a robot moving in the arena for a fixed time ($t_e$ =30 s) running the controller with the weights given by that particle's position. At the end of each evaluation, robots communicate the number of samples, mean, and variance of the solution they have evaluated, and they synchronize for the start of the next evaluation. The time required for the communication and synchronization is negligible in comparison to the evaluation time of the controllers (less than 1 second vs 30 seconds).

## V. Results

For each configuration of the algorithms under study we perform 20 learning runs for statistical significance. The number of iterations for each learning experiment is 250 in the case of *PSO pbest*, and 50 for *PSO ocbaC* and *PSO ocbaD*. This results in the same total number of evaluations for every algorithm. Parameters that are not explicitly mentioned in this section take the values listed in Section III (i.e., we only mention parameters that are different from the defaults).

Due to the presence of noise, the fitness value of the best solution as reported by the algorithms may not be an accurate representation of the actual performance of the solution. Therefore, in order to accurately judge the performances for comparison purposes, we perform 100 a-posteriori evaluations of the best solution at each iteration and consider the mean of the 100 evaluations as the *ground truth* performance of that solution.

We begin by comparing the final ground truth performance of the three algorithms with their default parameter values, shown in Fig. 6. Due to the adequate selection of parameters, all algorithms achieve the desired robotic behavior with high performances, with *PSO ocbaC* slightly outperforming the two distributed algorithms. There is no statistically significant difference in ground truth performance between *PSO pbest* and *PSO ocbaD* (Mann Whitney U test, 5% significance level), but we will show next that there is a significant difference in their estimates of the ground truth.

We now compare the performances estimated by the distributed algorithms with the ground truth for different neighborhood sizes. Figures 7 and 8 show the progress of *PSO ocbaD* and *PSO pbest* for neighborhoods of size 3, 7, 15, and 24.

By comparing these two figures, we can see that the difference between estimates and ground truth is much lower in the case of *PSO ocbaD* than in *PSO pbest*, i.e., *PSO pbest* reports a much higher performance than the one it actually achieves. Also, for *PSO ocbaD* the difference becomes smaller for larger neighborhood sizes, which is not the case for *PSO pbest*.

In order to measure the differences between the performance estimated by the algorithm and the ground truth from a-posteriori measurements we calculate the root-mean-squared error (RMSE):

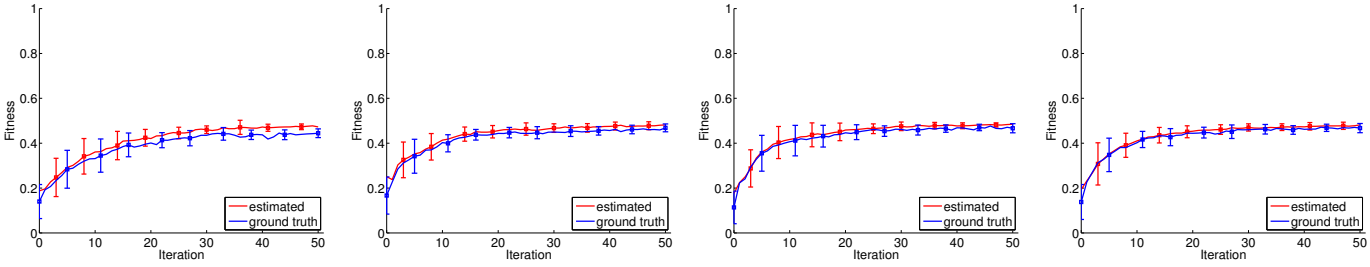
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{X}_i - X_i)^2} \qquad (12)$$

where $N = 20$ is the number of runs, $\hat{X}_i$ is the performance estimated by the algorithm for run $i$, and $X_i$ is the ground truth for run i.

Table III shows the calculated RMSE for *PSO ocbaC*, *PSO ocbaD*, and *PSO pbest* using different neighborhood sizes. *PSO ocbaC* provides the best estimate of the ground truth performance, followed by *PSO ocbaD*. The errors for both PSO OCBA variants are one order of magnitude smaller than for *PSO pbest*.

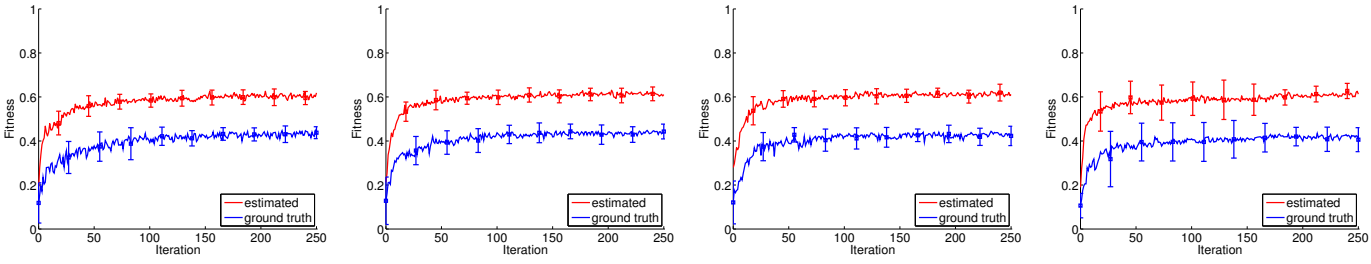
We now switch to the analysis of two parameters that are specific to the OCBA algorithm: $n_0$ and $\Delta$. $n_0$ is the number of samples used for the initial estimates of the mean and variance: low values can lead to poor initial estimates, while high values may waste function evaluations on poor solutions. In [12] values between 5 and 20 are suggested as suitable choices for stand-alone OCBA.

We ran the two PSO OCBA algorithms with three different $n_0$ values: 2, 4, and 8; results are shown in Table IV. From




(a) (b) (c) (d)

Fig. 7. Progress averaged over 20 runs for *PSO ocbaD* with increasing neighborhood size. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a-posteriori evaluations. Error bars represent one standard deviation (a) Neighborhood size 3. (b) Neighborhood size 7. (c) Neighborhood size 15. (d) Neighborhoood size 24.




(a) (b) (c) (d)

Fig. 8. Progress averaged over 20 runs for *PSO pbest* with increasing neighborhood size. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a-posteriori evaluations. Error bars represent one standard deviation (a) Neighborhood size 3. (b) Neighborhood size 7. (c) Neighborhood size 15. (d) Neighborhoood size 24.

TABLE III
ROOT-MEAN-SQUARED ERROR BETWEEN ESTIMATES AND GROUND TRUTH FOR INCREASING NEIGHBORHOOD SIZE.

| Algorithm | Neighborhood size | RMSE |
|---|---|---|
| PSO ocbaC | 3 | 0.009 |
| PSO ocbaC | 7 | 0.013 |
| PSO ocbaC | 15 | 0.012 |
| PSO ocbaC | 24 | 0.015 |
| PSO ocbaD | 3 | 0.033 |
| PSO ocbaD | 7 | 0.023 |
| PSO ocbaD | 15 | 0.024 |
| PSO ocbaD | 24 | 0.026 |
| PSO pbest | 3 | 0.19 |
| PSO pbest | 7 | 0.18 |
| PSO pbest | 15 | 0.19 |
| PSO pbest | 24 | 0.21 |

these experiments, we found that when applying OCBA in PSO, the lowest possible value of $n_0 = 2$ led to the best final estimate of the ground truth performance, which is different from the guideline for stand-alone OCBA. This result suggests that for PSO OCBA it is better to save function evaluations in the initial estimates and perform more iterations of the OCBA algorithm.

The other OCBA parameter that we analyze is $\Delta$, the additional number of samples used in the internal OCBA iterations. A smaller delta means that more OCBA calculations are performed, which is more computationally expensive but provides better estimates because mean and standard deviation

TABLE IV
ROOT-MEAN-SQUARED ERROR BETWEEN ESTIMATES AND GROUND TRUTH FOR INCREASING $n_0$.

| Algorithm | $n_0$ | $\Delta$ | RMSE |
|---|---|---|---|
| PSO ocbaC | 2 | 4 | 0.009 |
| PSO ocbaC | 4 | 4 | 0.019 |
| PSO ocbaC | 8 | 4 | 0.025 |
| PSO ocbaD | 2 | 1 | 0.033 |
| PSO ocbaD | 4 | 1 | 0.035 |
| PSO ocbaD | 8 | 1 | 0.051 |

are updated more often. The guidelines for this parameter are to select a number bigger than 5 but smaller than 10% of the number of candidates [12].

Table V shows the results from experiments with the two PSO OCBA algorithms and different values of $\Delta$, where it can be seen that lower values of $\Delta$ produce a lower error. In robotic learning the computational cost of the OCBA procedure is negligible compared to the cost of evaluations (both with high-fidelity simulation and real robot experiments), which is why the guidelines from stand-alone OCBA do not necessarily apply. Therefore, we can select the lowest possible value for $\Delta$, which is one for the distributed case and four (the number of robots in the system) for the centralized one (in order to avoid idle robots and evaluate a maximum number of solutions in parallel).

TABLE V
ROOT-MEAN-SQUARED ERROR BETWEEN ESTIMATES AND GROUND TRUTH FOR INCREASING $\Delta$.

| Algorithm | $n_0$ | $\Delta$ | RMSE |
|---|---|---|---|
| PSO ocbaC | 2 | 4 | 0.009 |
| PSO ocbaC | 2 | 8 | 0.014 |
| PSO ocbaC | 2 | 16 | 0.022 |
| PSO ocbaC | 2 | 32 | 0.017 |
| PSO ocbaD | 2 | 1 | 0.033 |
| PSO ocbaD | 2 | 2 | 0.039 |
| PSO ocbaD | 2 | 4 | 0.065 |
| PSO ocbaD | 2 | 8 | 0.056 |

## VI. CONCLUSION

To conclude, we will summarize the contributions from this article. We have described two PSO OCBA algorithms that are suitable for noisy optimization problems such as those found in robotics. The centralized version is similar to those previously applied to numerical benchmark functions, while the distributed version is the first of such kind found in the literature (to the best of our knowledge). The distributed version has limited communication and computational requirements, which makes it suitable for multi-robot distributed on-line optimization problems.

Both versions provided better estimates of the ground truth performance than a previous distributed noise-resistant PSO implementation, even though the final performance of the three algorithms was similar. The distributed version's estimate of the ground truth was slightly worse than the centralized one, but the difference was reduced when increasing the neighborhood size. This was not the case for the previous distributed noise-resistant PSO, whose estimation error was an order of magnitude worse than both PSO OCBA algorithms.

We also analyzed two parameters specific to OCBA: $n_0$ and $\Delta$. We showed that the lowest possible values ($n_0 = 2$, $\Delta =$ number of robots) performed best, and these values differ considerably from the guidelines for stand-alone OCBA present in the literature.

As a future work, we would like to implement a modified version of the distributed PSO OCBA for the learning of heterogeneous controllers for cooperative robotic behaviors as in [24].

## ACKNOWLEDGMENT

This research was partially supported by the Swiss National Science Foundation through the National Center of Competence in Research Robotics.

## REFERENCES

[1] Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain Environments: A Survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[2] K. E. Parsopoulos and M. N. Vrahatis, "Particle Swarm Optimizer in Noisy and Continuously Changing Environments," in *Artificial Intelligence and Soft Computing*, 2001, pp. 289–294.

[3] H. Pan, L. Wang, and B. Liu, "Particle Swarm Optimization for Function Optimization in Noisy Environment," *Applied Mathematics and Computation*, vol. 181, no. 2, pp. 908–919, 2006.

[4] J. Fitzpatrick and J. Grefenstette, "Genetic Algorithms in Noisy Environments," *Machine Learning*, vol. 3, no. 2, pp. 101–120, 1988.

[5] D. Arnold and H.-G. Beyer, "A General Noise Model and its Effects on Evolution Strategy Performance," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 380–391, 2006.

[6] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *IEEE International Conference on Neural Networks*, 1995, pp. 1942 – 1948 vol.4.

[7] M. Turduev and Y. Atas, "Cooperative Chemical Concentration Map Building Using Decentralized Asynchronous Particle Swarm Optimization Based Search by Mobile Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4175–4180.

[8] L. Marques, U. Nunes, and A. T. Almeida, "Particle Swarm-based Olfactory Guided Search," *Autonomous Robots*, vol. 20, no. 3, pp. 277–287, 2006.

[9] J. Hereford and M. Siebold, "Using the Particle Swarm Optimization Algorithm for Robotic Search Applications," in *IEEE Swarm Intelligence Symposium*, 2007, pp. 53–59.

[10] J. Pugh and A. Martinoli, "Distributed Scalable Multi-robot Learning using Particle Swarm Optimization," *Swarm Intelligence*, vol. 3, no. 3, pp. 203–222, 2009.

[11] J. Pugh, Y. Zhang, and A. Martinoli, "Particle Swarm Optimization for Unsupervised Robotic Learning," in *IEEE Swarm Intelligence Symposium*, 2005, pp. 92–99.

[12] C. Chen, J. Lin, E. Yücesan, and S. E. Chick, "Simulation Budget Allocation for Further Enhancing the Efficiency of Ordinal Optimization," *Discrete Event Dynamic Systems: Theory and Applications*, pp. 251–270, 2000.

[13] T. Bartz-Beielstein, D. Blum, and J. Branke, "Particle Swarm Optimization and Sequential Sampling in Noisy Environments," *Metaheuristics*, vol. 39, pp. 261–273, 2007.

[14] J. Rada-Vilela, M. Zhang, and M. Johnston, "Optimal Computing Budget Allocation in Particle Swarm Optimization," in *Genetic and Evolutionary Computation Conference*. ACM, 2013, pp. 81–88.

[15] C. Schmidt, J. Branke, and S. Chick, "Integrating Techniques from Statistical Ranking into Evolutionary Algorithms," in *Applications of Evolutionary Computing*, 2006, vol. 3907, pp. 752–763.

[16] G. LaPorte, J. Branke, and C.-H. Chen, "Optimal Computing Budget Allocation for Small Computing Budgets," in *Winter Simulation Conference*, 2012, pp. 1–13.

[17] E. Di Mario, I. Navarro, and A. Martinoli, "Analysis of Fitness Noise in Particle Swarm Optimization: From Robotic Learning to Benchmark Functions," in *IEEE Congress on Evolutionary Computation*, 2014, pp. 2785–2792.

[18] E. Di Mario and A. Martinoli, "Distributed Particle Swarm Optimization for Limited Time Adaptation with Real Robots," *Robotica*, vol. 32, no. 02, pp. 193–208, 2014.

[19] D. Floreano and F. Mondada, "Evolution of Homing Navigation in a Real Mobile Robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 3, pp. 396–407, 1996.

[20] B. Huang, G. Cao, and M. Guo, "Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance," in *International Conference on Machine Learning and Cybernetics*, 2005, pp. 85–89.

[21] R. E. Palacios-Leyva, R. Cruz-Alvarez, F. Montes-Gonzalez, and L. Rascon-Perez, "Combination of Reinforcement Learning with Evolution for Automatically Obtaining Robot Neural Controllers," in *IEEE International Conference on Evolutionary Computation*, 2013, pp. 119–126.

[22] E. Di Mario, I. Navarro, and A. Martinoli, "The Role of Environmental and Controller Complexity in the Distributed Optimization of Multi-Robot Obstacle Avoidance," in *IEEE International Conference on Robotics and Automation*, 2014, pp. 571–577.

[23] O. Michel, "Webots: Professional Mobile Robot Simulation," *Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[24] E. Di Mario, I. Navarro, and A. Martinoli, "Distributed Learning of Cooperative Robotic Behaviors using Particle Swarm Optimization," in *International Symposium on Experimental Robotics 2014, Springer Tracts in Advanced Robotics*, to appear.