Noise-Resistant Particle Swarm Optimization for the Learning of Robust Obstacle Avoidance Controllers using a Depth Camera

Iñaki Navarro Ezequiel Di Mario Alcherio Martinoli

***Abstract*— The Ranger robot was designed to interact with children in order to motivate them to tidy up their room. Its mechanical configuration, together with the limited field of view of its depth camera, make the learning of obstacle avoidance behaviors a hard problem. In this article we introduce two new Particle Swarm Optimization (PSO) algorithms designed to address this noisy, high-dimensional optimization problem. Their aim is to increase the robustness of the generated robotic controllers, as compared to previous PSO algorithms. We show that we can successfully apply this set of PSO algorithms to learn 166 parameters of a robotic controller for the obstacle avoidance task. We also study the impact that an increased evaluation budget has on the robustness and average performance of the optimized controllers. Finally, we validate the control solutions learned in simulation by testing the most robust controller in three different real arenas.**

I. Introduction

Performance evaluations of robotic controllers are inherently noisy, with sources of randomness ranging from sensor and actuator noise, varying initial conditions, and manufacturing tolerances to changes in the environment. Furthermore, as reported in [1], carrying out obstacle avoidance with short-range, noisy sensors results in performance distributions that are not Gaussian and in many cases have very large standard deviations.

Population-based optimization techniques have been successfully leveraged for coping with noisy fitness distributions [2]. We can find examples of successful operation under noise for Particle Swarm Optimization (PSO) [3], [4], Genetic Algorithms [5], and Evolutionary Strategies [6].

We focus our research on the PSO algorithm [7], because of its potential for an efficient distributed implementation, adding robustness to failure of individual robots in the learning process, and speeding it up through parallel evaluation. PSO has been applied to different problems in robotics, such as odor source localization [8], flocking [9], robotic search [10], and obstacle avoidance [11].

In this article, we use the Ranger robot [12], a small limited robotic platform designed to interact with children, since it has various characteristics which make it interesting for this research. First, it is equipped with a depth-camera used as unique input for our controller. The richness in the resulting perceptual input speaks for a highly optimized, fine-grained use of this single on-board sensor. If this optimized use is achieved through bottom-up learning, this in turns means being able to handle a high-dimensional search space. Second, it has a square footprint, and two motors in differential drive configuration placed in the front part (see Fig. 1). This mechanical configuration, together with the limited field of view of the depth camera, makes the learning of obstacle avoidance behaviors a hard problem. Obstacle avoidance should be therefore considered here as a challenging benchmark for our learning techniques.

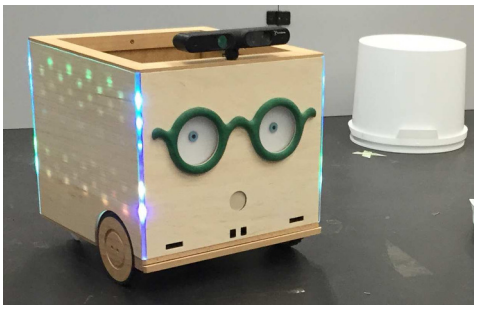


Fig. 1. The Ranger robot.

The maneuvering limitations of the Ranger robot in combination with the previously mentioned sources of randomness can result in not only learned controllers with acceptable mean performance, but also in a significant number of outliers characterized by very low performances. Thus, this article focuses on how to learn robust controllers which consistently show good performances by exploring four different PSO algorithms. By robustness here, we mean that the worst evaluation of a given controller should be as high as possible, i.e., that the controller succeeds in (almost) every case.

In our previous work [13], [14], we presented and explored various noise-resistant PSO algorithms for multi-robot learning based on Optimal Computing Budget Allocation (OCBA), a statistical sample allocation method introduced by Chen et al. [15]. We presented both distributed and centralized versions of a PSO algorithm based on OCBA, suitable for resource-constrained mobile robots due to its low requirements in terms of memory and communication. We showed that our PSO-OCBA algorithms outperform other techniques [11] for dealing with noise in robotic learning, achieving a more consistent progress and a better estimate of the ground-truth performance of candidate solutions.

The first of the four PSO algorithms adopts a naïve evaluation approach: each candidate solution is evaluated a

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

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

fixed number of times and the resulting scores are aggregated in a simple average to improve the performance estimation. The second, is the PSO-OCBA centralized version described in [13], [14], which allocates the number of evaluations to maximize the probability of correct selection of good candidates, and estimates the performance by averaging the results of the corresponding evaluations. The third and fourth algorithms are modifications of the first and the second, respectively, with the aim of increasing the robustness of the resulting controllers. Their implementation leverages a different aggregation function for generating an appropriate estimate of the fitness of the candidate solutions.

In the four aforementioned algorithms, we also study the impact of an increased evaluation budget in the controllers' robustness and mean performance.

The remainder of this article is organized as follows. In Section II, we describe the obstacle avoidance task that will be used to compare the algorithms, and the robotic platform used. Section III describes the four PSO algorithms used in this paper. Section IV presents the results obtained by learning in simulation, and evaluates the most robust controller on a real robot. Finally, Section V concludes the paper.

II. Benchmark Task

We have chosen obstacle avoidance as a task to illustrate robotic learning because it is a fundamental task popular in the robotic learning literature [11], [16]–[19], and it is a challenging task for the concrete robotic platform used in this study.

We use a metric of performance based on the work in [16], which was also used in [11], [18], [19]. It consists of three factors, all normalized to the interval betwen 0 and 1:

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

$$f_v = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \frac{\max\{v_{l,k} + v_{r,k}, 0\}}{2} \quad (2)$$

$$f_t = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \frac{|v_{l,k} - v_{r,k}|}{2} \quad (3)$$

$$f_i = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} i_{min,k} \quad (4)$$

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k, $i_{min,k}$ is the normalized proximity distance value of the closest distance measured at time step k, and N_{eval} is the number of time steps in the evaluation period. This function rewards robots that move forwards 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 the resulting series is averaged over the total number of time steps in the evaluation period.

We conducted our experiments using the Ranger robot [12] (see Fig. 1), a wooden robotic box equipped with wheels and sensors aimed to interact with children and motivate them to tidy up their room [20]. It has a $30\,cm$ width, and it weights approximately $3.5\,kg$. Its differential drive motors are positioned in the front of the robot, complemented by two castor wheels in the back. It has two wheel encoders, which are used to measure the wheel speeds for the fitness calculations. Its sensing capabilities were augmented with a depth camera *Primesense Carmine 1.09*, placed in the front upper part of the robot, which in our case is the only external input for the controller. This depth sensor has a resolution of $640 \times 480\,pixels$, a horizontal field of view of $1\,rad$, and a nominal range of $0.35 - 1.4\,m$.

In this work, the depth sensor is configured at a reduced resolution of $160\,columns \times 120\,rows$. Only the rows in the range $[61, 108]$ (being 1 the top row of the sensor) are used both for the controller inputs and the fitness calculation. Each distance measurement above $1\,m$ is converted to $1\,m$, and all the measurements are normalized to the interval [0, 1] (in this specific case already normalized if expressed in meters). Given the technology used by the depth sensor, objects under $35\,cm$ or areas near sharp edges cannot be measured. This effect was not modeled in our simulations.

Calculating f_i corresponds to obtaining the minimum normalized measurement among every pixel between rows 61 and 108.

We reduce the normalized pixel depth information of rows $[61, 108]$ into 80 virtual sensors to be used as only external inputs of the controller. Each virtual sensor is the result of obtaining the minimum of the normalized values of a group of 96 depth-pixels. Each group of 96 pixels is composed of two adjacent columns of the 48 pixels between rows 61 and 108. The different virtual sensors give horizontal resolution of distance information, but filter out the vertical resolution.The reason for performing this kind of reduction is that the robot only moves in a 2D plane.

The controller used is a recurrent artificial neural network of two units with sigmoidal activation functions. The outputs of the units determine the wheel speeds. Each neuron has 83 input connections: the 80 virtual 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 166 weight parameters in total. These 166 parameters define the dimensionality of the learning space of the algorithms.

The high-dimensional optimization problem to be solved by the PSO algorithms is to choose the set of weights of the artificial neural network controller such that the fitness function f as defined in Eq. 1 is maximized.

III. Algorithms

PSO is a metaheuristic originally introduced by Kennedy and Eberhart [7], which was inspired by the movement of flocks of birds and schools of fish. It models candidate solutions as a swarm of particles moving in a high-dimensional space. The position of each particle represents a set of weights of a controller. 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.

The movement of particle i in dimension j depends on three components: the velocity at the previous step weighted

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate particle position
5:       Update personal best
6:       Update neighborhood best
7:       Update particle position
8:    end for
9: end for
```

Fig. 2. Pseudocode for the canonical PSO algorithm.

by an inertia coefficient w, 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. 5). $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$v_{i,j} = w \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \quad (5)$$

Each particle evaluation consists of a robot moving in the arena for a fixed time running the controller with the weights given by that particle's position. Particle evaluations are performed in parallel, which means that each robot is testing a different controller at any given time. The fitness corresponding to a particle is equivalent to the performance of the robot measured with function f from Eq. 1. The pseudocode for the canonical PSO is shown in Figure 2. The four algorithms studied in this article correspond to variations of it.

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 [21]. The only exception is the number of particles (N_p) which was reduced to 48 instead of using the dimension of the problem ($N_p = D = 166$). This reduction was done to decrease the computational cost of the simulations. Based on experimental evidence not reported in the article, we found that for this particular problem and experimental settings, the complexity of the fitness landscape grew sublinearly in respect to the number of inputs of the networks. This is probably due to the high correlation between topologically close inputs, and to the limited actions that the robot can take. Thus, reducing N_p did not have a strong effect in the convergence of the PSO algorithms.

The most common approach to deal with noise consists of evaluating the fitness function several times in order to obtain a better estimate. In this article, we explore the influence of the evaluation budget per iteration. We explore three different budgets (B_1, B_2, and B_3), which can be fairly compared across the four different algorithms. The comparison is fair in the sense that for a given budget B_i, the four algorithms have the same number of evaluations per iteration and in total.

TABLE I
PARAMETERS COMMON TO ALL PSO ALGORITHMS

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	48
Evaluation span t_e	60 s
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Neighborhood size N_n	3
Dimension D	166
Inertia w	0.8
V_{max}	20
Iterations N_i	40

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate M times particle position
5:       Compute estimated performance of particle position
6:       Update personal best
7:       Update neighborhood best
8:       Update particle position
9:    end for
10: end for
```

Fig. 3. Pseudocode for *PSO rep* and *PSO rep dec* algorithms. In *PSO rep*, line 5 estimates the performance by calculating the mean of the M samples, while in *PSO rep dec* the performance is obtained by calculating the first decile of the M samples.

A. PSO rep

PSO rep corresponds to the naïve approach of evaluating every new candidate a fixed number of times (M) [22], and obtaining the estimated performance as the mean of the M evaluations. The pseudocode for *PSO rep* is shown in Figure 3, in which line 5 corresponds to performing the mean of the M samples previously evaluated.

In Table II, budget parameters for *PSO rep* are shown. The budget per iteration for *PSO rep* corresponds to $N_p \cdot M$.

PSO rep results in a good performance estimation for new candidates, but invests as many resources in good as in poor candidates which could be immediately discarded [22]. Another disadvantage 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.

B. PSO ocba

The idea behind the application of OCBA to PSO is to fix the mentioned issues of *PSO rep* which affect its performance under the presence of noise. OCBA is a technique based on Bayesian statistics for allocating samples to different candidate solutions introduced by Chen et al. [15]. Given k candidates with means $\{\bar{X}_1, \ldots, \bar{X}_k\}$ and variances $\{\sigma^2_1, \ldots, \sigma^2_k\}$, and a total number of samples T, OCBA aims at maximizing the probability of correctly selecting candidate b as the best (the one with the lowest mean):

$$P\{CS\} = P\{\bar{X}_B < \bar{X}_i, i \neq b\} \quad (6)$$

TABLE II

BUDGET PARAMETERS FOR *PSO rep*, AND *PSO rep dec*

Parameter	B_1	B_2	B_3
Evaluations of new candidates M	5	10	20
Iteration budget $B_{it} = N_p \cdot M$	240	480	960

by applying the following allocation rules:

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i / \delta_{b,i}}{\sigma_j / \delta_{b,j}} \right)^2, i \neq j \neq b \quad (7)$$

$$N_b = \sigma_b \sqrt{\sum_{i=1, i \neq b}^{k} \frac{N_i^2}{\sigma_i^2}} \quad (8)$$

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 7 and 8 is that candidate i will get more samples N_i when it has larger variance σ_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 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 [15]. OCBA was previously applied to PSO on numerical benchmark functions [4], [22], where it outperformed other techniques for dealing with noise.

In *PSO ocba*, previously introduced in [13], [14], 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 at the same time a larger proportion of the allocation budget to accurately test new candidates.

PSO ocba pseudocode is shown in Fig. 4. Most steps are similar to *PSO rep*, but instead of evaluating every new position a fixed number of times M in the evaluation step, it allocates the iteration budget (B_{it}) in a different manner. First, n_0 samples of the new positions are taken to estimate their mean and variance (in our case, $n_0 = 2$). Then the remaining samples are allocated among all the new positions and all the personal bests ($2N_p$ candidates total) using Equations 7 and 8. 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 estimated performance is calculated for each particle position by obtaining the mean of the evaluation samples.

In Table III, the specific parameters of *PSO ocba* algorithm are defined for the three budgets studied in this paper.

```
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 estimated performance 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. 4. Pseudocode for the *PSO ocba* and *PSO ocba dec* algorithms. In *PSO ocba* line 10 estimates the performance by calculating the mean of the samples, while in *PSO rep dec* the performance is obtained by calculating the first decile of the samples. In both versions the variance is also calculated in line 10.

TABLE III

BUDGET PARAMETERS FOR *PSO ocba*, AND *PSO ocba dec*

Parameter	B_1	B_2	B_3
Iteration budget B_{it}	240	480	960
Initial number of samples n_0	2	2	2
Additional number of samples Δ	4	4	4

C. PSO rep dec

PSO rep dec is a modification of *PSO rep* with the aim of increasing the robustness of the resulting controllers. Robust controllers are those with high values of their worst evaluation for a set of evaluation samples. The hypothesis here is that using the mean might not be the best statistical aggregation function for estimating the performance of a candidate solution in such noisy settings.

One possibility is to use the minimum of the M samples to calculate the estimated performance. By running some preliminary tests, we have seen that the learning process is very hard, since the minimum is a too sensitive aggregation function to single bad samples and the evaluations have large standard deviation. Instead, *PSO rep dec* computes the first decile of the M evaluations for the estimation of the performance. Thus, the algorithm learns controllers with the first decile containing the lowest performance set. The pseudocode is the same as for *PSO rep*, shown in Fig. 3, but in which line 5 corresponds to calculating the first decile of the M samples previously evaluated. We calculate the first decile for a set of M sorted (from the lowest to the highest)

samples as follows. If $M/10$ is an integer, then:

$$D_1 = \frac{val(M/10) + val(M/10+1)}{2} \quad (9)$$

where $val(i)$ is the value of the i^{th} sample of the sorted set. If $M/10$ is not an integer, then:

$$D_1 = val(\lceil M/10 \rceil) \quad (10)$$

The three budgets explored are the same as in *PSO rep* (see Table II). For the budget B_1, $D_1 = val(1)$, while for B_2, $D_1 = \frac{val(1)+val(2)}{2}$, and for B_3, $D_1 = \frac{val(2)+val(3)}{2}$.

D. PSO ocba dec

PSO ocba dec is a variation of *PSO ocba*, in which as in *PSO rep dec* the first decile of the evaluated samples is used as estimated performance for the learning, aiming to increase the robustness of the resulting controllers. It corresponds to a more adaptive approach for the allocation of the budget than in the naïve *PSO rep dec* algorithm.

The pseudocode is the same as for *PSO ocba*, shown in Fig. 4, but in which in line 10 the estimated performance is calculated as the first decile of the already obtained samples, using also Equations 9 and 10.

The other main difference is the way the OCBA algorithm is used. Instead of using the mean and variance, the first decile and variance are used. Thus, in Eq. 7, $\delta_{i,j}$ is defined as the difference between the first decile of candidate i and the first decile of candidate j ($\delta_{i,j} = D_1(X_i) - D_1(X_j)$). OCBA will allocate more evaluations on candidates with high variance and large first decile.

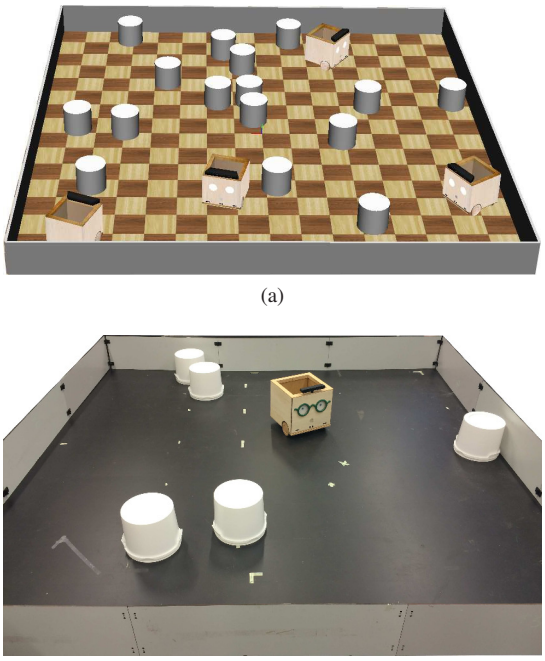
The three budgets studied for this algorithm are the same as in *PSO ocba* (see Table III).

IV. Experiments and Results

The learning experiments are performed in simulation using four Ranger robots, in a square arena of $4m$ x $4m$ with walls $0.3m$ height, where 10 cylindrical obstacles of $25cm$ diameter and $24cm$ height are added (see Fig. 5a). Before each fitness evaluation the static obstacles are randomly repositioned, and the initial robots' poses are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. Each fitness evaluation lasts $60s$. For our learning experiments, we use Webots [23], a high-fidelity submicroscopic simulator that models dynamical effects such as friction and inertia.

For each algorithm and budget size under study we perform 20 learning runs for statistical significance. 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, we perform 100 a-posteriori evaluations of the best solution at each iteration. This allows us to compare the controllers both in terms of average performance and lowest performance (robustness).

In order to compare the learning algorithms we can look at Figure 6, where the performance of the learned controllers is presented in boxplots grouped by budget. Each boxplot



(a)

(b)

Fig. 5. (a) Arena used during the learning in simulation. (b) Arena *env2* used in real robot experiments. In both cases the cylindrical obstacles are randomly placed at the beginning of each evaluation.

represents the performance of the 20 controllers learned per algorithm, each evaluated 100 times. We can see that the algorithm resulting in controllers with highest median is *PSO ocba* for every budget. In addition, the OCBA algorithms outperform the algorithms based on naïve allocation, both in robustness and median for the different budgets.

For smallest budget (B_1, Fig. 6a), both PSO algorithms based on OCBA are better in terms of robustness, showing the optimal allocation given by OCBA when the budget is limited. For B_1, the algorithm resulting in most robust controllers is *PSO ocba*, and not *PSO ocba dec* as we could expect. This might be due to a too limited budget to properly estimate the first decile. On the other hand, for budgets B_2 (Fig. 6b) and B_3 (Fig. 6c) *PSO ocba dec* slightly outperforms *PSO ocba dec* in terms of robustness of the learned controllers. Furthermore, *PSO rep dec* results in slightly more robust controllers than *PSO rep* only when the budget is the largest (B_3).

We have grouped the same boxplots by algorithm in Fig. 7 for better understanding the influence of the budget. In the case of algorithms based on fixed allocation of the evaluations (Fig. 7a and Fig. 7c), we can observe that the increase of budget improves the robustness of the controllers, but not so clearly the median value. For the algorithms based on OCBA (Fig. 7b and Fig. 7d), augmenting the budget does not have an impact in the median and robustness of the controllers. The smallest budget (B_1) is sufficient thanks to the use of the OCBA adaptive allocation.

The learning progress is shown in Fig. 8 for the different algorithms and budgets. The red curves show the mean over the 20 learning runs of the best solution as estimated by

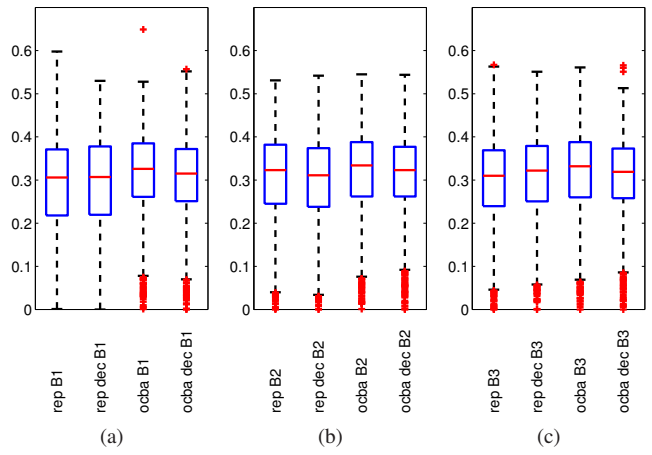


Fig. 6. Performance in simulation for the different learned controllers aggregating 100 a-posteriori evaluations from 20 runs for each learning algorithm and budget. The box represents the upper and lower quartiles, the line across the middle marks the median, the bars extend to the most extreme data points not considered outliers, and the red crosses show outliers. The lower bar is an indicator of the robustness. They are grouped by budget: (a) B_1, (b) B_2, and (c) B_3.

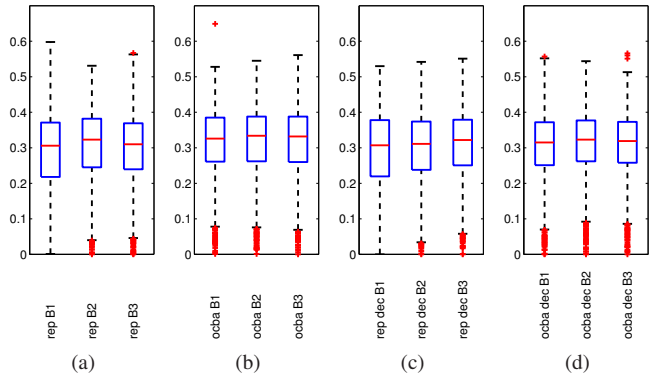


Fig. 7. Performance in simulation for the different learned controllers aggregating 100 a-posteriori evaluations from 20 runs for each learning algorithm and budget. The box represents the upper and lower quartiles, the line across the middle marks the median, the bars extend to the most extreme data points not considered outliers, and the red crosses show outliers. The lower bar is an indicator of the robustness. They are grouped by learning algorithm: (a) *PSO rep*, (b) *PSO ocba*, (c) *PSO rep dec*, and (d) *PSO ocba dec*.

the algorithm. The blue curves represent the ground truth performance obtained by 100 a posteriori evaluations and averaged over the 20 runs. As we already know from [13], *PSO rep* (Figures 8a-8c) is not able to properly estimate the performance and tends to overestimate when comparing with the ground truth, while *PSO ocba* (figures 8d-8f) can estimate properly. What we can observe here is that increasing the budget results into less estimation error in both algorithms.

When looking at *PSO rep dec* (Figures 8g-8i), we observe the same overestimation phenomenon as in *PSO rep*, and also the effect of increasing budget in reducing this overestimation. For *PSO ocba dec* (Figures 8j-8l), the algorithm is able to properly estimate the first decile for budgets B_2 and B_3, but fails to estimate properly for budget B_1 resulting in a small overestimation. This might be an indication that B_1 is not enough to properly compute the first decile, and the

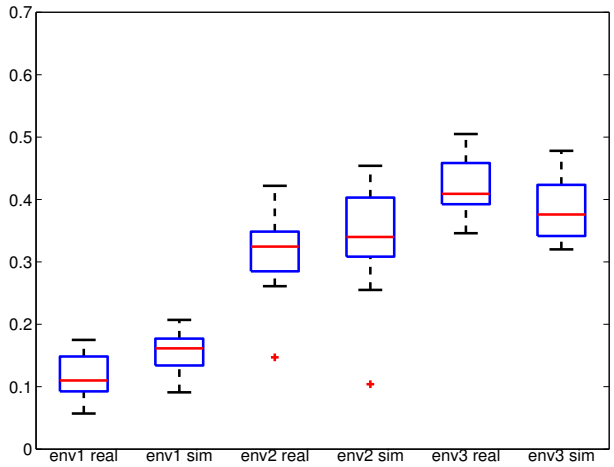


Fig. 9. Evaluation of the most robust learned controller. The performance was measured in 20 evaluations of 30 s per environment using single robot in reality and in simulation. The box represents the upper and lower quartiles, the line across the middle marks the median, the bars extend to the most extreme data points not considered outliers, and the red crosses show outliers.

reason why *PSO ocba* outperforms *PSO ocba dec* for B_1 in Fig. 6a.

In order to validate the learning process in simulation, we tested a selected controller using only one Ranger in three smaller arena versions of the learning one. The reduced size of the arenas and of the number of robots was due limitations in their availability and space. The arena *env1* is $2m$ x $2m$, with walls of $0.3m$ height and 3 cylindrical obstacles of same size as in the learning. It respects the same robot density as in the learning and approximates the obstacle density by rounding up the number of obstacles. *env2* is $3m$ x $3m$, with 5 cylindrical obstacles of the same type (see Fig. 5b). *env3* is also $3m$ x $3m$, without cylindrical obstacles.

The chosen controller is the most robust from all the learned controllers, and was obtained in a *PSO ocba* learning run of budget B_2. It has been evaluated 20 times per arena in reality, and in simulation on equivalent simulated arenas. Both the robot and the obstacles were randomly placed at the beginning of each evaluation, which lasted $30s$. In Fig. 9, we can see the performance of the selected controllers for the three arenas in simulation and reality. We can observe how the distribution and median of the performances are very similar between simulation and reality, and the small discrepancies might be due to small differences in the models used. The difference in performance between arenas is due to the amount of free space available.

V. Conclusion

In this article we have introduced two new PSO algorithms based on the use of the first decile of several samples for the estimation of the performance. Their aim is to increase the robustness of the generated robotic controllers. We have seen that when enough evaluation budget is available (B_3 for the fixed allocation approach and B_2 and B_3 for the OCBA one) their resulting controllers are on average slightly more

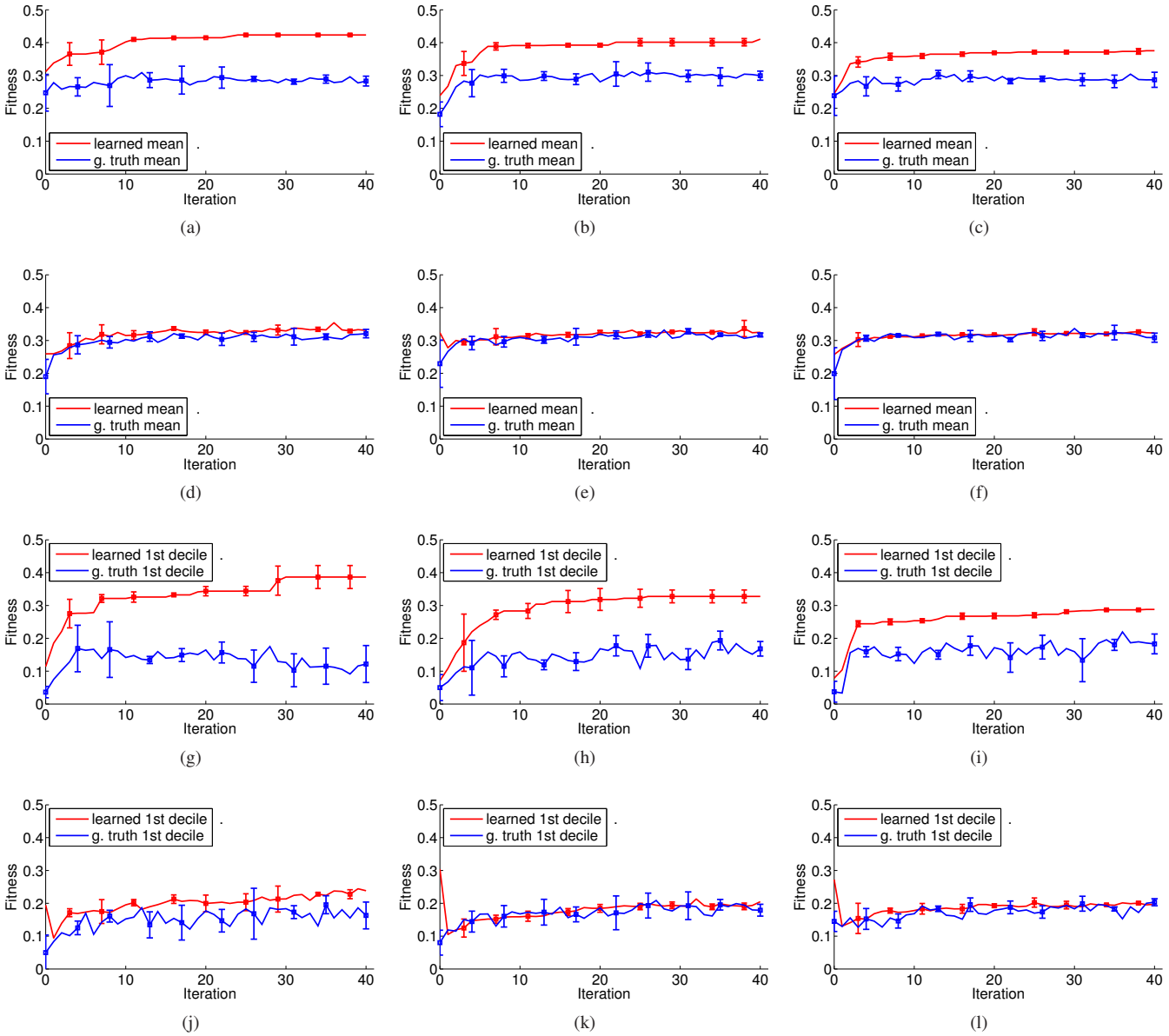


Fig. 8. Progress averaged over 20 runs for each of the four algorithms and three different budgets. The red curve represents the performance of the best solution as estimated by the algorithm and averaged over the 20 runs, and the blue curve represents the ground truth performance obtained by 100 a posteriori evaluations per run and averaged over the 20 runs. In the case of *PSO rep* and *PSO ocba* the algorithms use the mean of the samples as estimated performance for the learning, and so the ground truth corresponds to the mean of 100 a posteriori evaluations. In *PSO rep dec* and *PSO ocba dec* learning is made using the first decile as estimated performance, so the ground truth corresponds to the first decile of 100 a posteriori evaluations. Error bars represent one standard deviation. (a) *PSO rep* B_1. (b) *PSO rep* B_2. (c) *PSO rep* B_3. (d) *PSO ocba* B_1. (e) *PSO ocba* B_2. (f) *PSO ocba* B_3. (g) *PSO rep dec* B_1. (h) *PSO rep dec* B_2. (i) *PSO rep dec* B_3. (j) *PSO ocba dec* B_1. (k) *PSO ocba dec* B_2. (l) *PSO ocba dec* B_3.

robust than those obtained by using PSO algorithms based on the mean. Unfortunately, we cannot conclude that this improvement can be generalized for other sets of learning parameters or benchmark tasks.

We have also explored the effect of the evaluation budget for the four algorithms covered in this paper. Results show that increasing the budget helps to generate more robust and performant controllers in the case of the naïve repetition approaches. On the other hand, this increase in the budget has no effect on the controllers produced by PSO algorithms based on OCBA, since they are able to allocate low budgets in an adaptive way. Increasing the total budget by having a larger population size might have a positive impact on the robustness and performance of the controllers, since it allows for more exploration in the learning. This could be studied in future works.

The two algorithms based on OCBA outperform their naïve allocation budget versions in terms of the average robustness of their resulting controllers. They are also able to properly estimate the ground truth performance, although in the concrete case of *PSO ocba dec* it needs a certain minimum budget.

Furthermore, we have shown that we can apply successfully this set of PSO algorithms to very high-dimensional

problems, in this case to learn 166 parameters of a robotic controller. We validated our learning in simulation by testing the most robust controller in three different real arenas.

As a future work, we would like to implement the PSO OCBA algorithms for the learning of heterogeneous controllers for cooperative robotic behaviors as in [9].

VI. Acknowledgements

We would like to express special thanks to Francesco Mondada, Daniel Burnier, Norbert Crot from the Laboratoire de Systèmes Robotiques at EPFL, and to David Mansolino from our lab for their great help with the Ranger robot. Without their support this work would not have been possible.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[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] I. Navarro, E. Di Mario, and A. Martinoli, "Distributed vs. centralized Particle Swarm Optimization for Learning Flocking Behaviors," in *Proceedings of the European Conference on Artificial Life 2015*, P. Andrews, L. Caves, R. Doursat, S. Hickinbotham, F. Polack, S. Stepney, T. Taylor, and J. Timmis, Eds. The MIT Press, 2015, pp. 302–309.

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

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

[12] F. Mondada, J. Fink, S. Lemaignan, D. Mansolino, F. Wille, and K. Franinovi, "Ranger, an Example of Integration of Robotics into the Home Ecosystem," in *International Workshop and Summer School on Medical and Service Robotics*, 2014.

[13] E. Di Mario, I. Navarro, and A. Martinoli, "A Distributed Noise-Resistant Particle Swarm Optimization Algorithm for High-Dimensional Multi-Robot Learning," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5970 – 5976.

[14] ——, "Distributed Particle Swarm Optimization using Optimal Computing Budget Allocation for Multi-Robot Learning," in *IEEE Congress on Evolutionary Computation*, 2015, pp. 566–572.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] E. Di Mario, I. Navarro, and A. Martinoli, "The Effect of the Environment in the Synthesis of Robotic Controllers: A Case Study in Multi-Robot Obstacle Avoidance using Distributed Particle Swarm Optimization," in *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, Advances in Artificial Life, ECAL 2013*, 2013, pp. 561–568.

[20] J. Fink, S. Lemaignan, P. Dillenbourg, P. Rétornaz, F. Vaussard, A. Berthoud, F. Mondada, F. Wille, and K. Franinović, "Which Robot Behavior Can Motivate Children to Tidy up Their Toys?: Design and Evaluation of "Ranger"," in *Proceedings of the 2014 ACM/IEEE International Conference on Human-robot Interaction*, 2014, pp. 439–446.

[21] 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.

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

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