A Distributed Noise-Resistant Particle Swarm Optimization Algorithm for High-Dimensional Multi-Robot Learning

Ezequiel Di Mario Iñaki Navarro Alcherio Martinoli

***Abstract*— Population-based learning techniques have been proven to be effective in dealing with noise in numerical benchmark functions and are thus promising tools for the high-dimensional optimization of controllers for multiple robots with limited sensing capabilities, which have inherently noisy performance evaluations. In this article, we apply a statistical technique called Optimal Computing Budget Allocation to improve the performance of Particle Swarm Optimization in the presence of noise for a multi-robot obstacle avoidance benchmark task. We present a new distributed PSO OCBA algorithm suitable for resource-constrained mobile robots due to its low requirements in terms of memory and limited local communication. Our results from simulation show that PSO OCBA outperforms other techniques for dealing with noise, achieving a more consistent progress and a better estimate of the ground-truth performance of candidate solutions. We then validate our simulations with real robot experiments where we compare the controller learned with our proposed algorithm to a potential field controller for obstacle avoidance in a cluttered environment. We show that they both achieve a high performance through different avoidance behaviors.**

I. Introduction

The high-dimensional optimization of robotic controllers is an expensive process because, even in simulation, performance evaluations require more time than any other computation in the optimization process. Moreover, in the context of multiple robots with limited sensing and actuating capabilities, there are several sources of uncertainties, such as sensor and actuator noise, varying initial conditions, manufacturing tolerances, or changes in the environment, that can increase the variance of performance measurements [1].

Population-based learning techniques have been proven to be effective in dealing with noise in fitness evaluations [2]. Within this family of algorithms, we can find examples of successful performances under noise for Particle Swarm Optimization [3], [4], Genetic Algorithms [5], and Evolutionary Strategies [6].

We focus this research on the Particle Swarm Optimization (PSO) algorithm [7], as it is well suited for distributed multi-robot implementations due to its distinct individual and social components [8]. Examples of applications of PSO to mobile robots are odor source localization [9], [10], robotic search [11], and obstacle avoidance [8].

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 [3]. 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 [8], [12].

In our previous work [1], we have analyzed different techniques for dealing with noise in PSO and identified potential sources for improvement. In this article, we present a new distributed noise-resistant PSO algorithm for multi-robot learning. It is based on Optimal Computing Budget Allocation (OCBA), a statistical sample allocation method introduced by Chen et al. [13]. OCBA has previously been applied to PSO on numerical benchmark functions [4], [14]. Here, we first detail how we apply PSO OCBA in a centralized manner to a multi-robot obstacle avoidance benchmark task. Then, we propose a new distributed version which requires only local information and communication, and compare it to other algorithms in the same task.

Obstacle avoidance was used in one of the earliest works of evaluative adaptation with Genetic Algorithms applied to real robots [15], and it has also been employed to test other learning algorithms such as PSO [8] and Reinforcement Learning [16]. We chose obstacle avoidance as a benchmark task because it can be implemented with different number of robots, requires basic sensors and actuators, and the performance metric can be defined to be fully evaluated with on-board resources. Thus, this task can serve as a benchmark for testing multi-robot learning algorithms in the same way that standard benchmark functions are used in numerical optimization, such as DeJong's test suite [17].

In addition, by increasing the density of obstacles in the arena, we can make the obstacle avoidance task increasingly challenging in terms of performance and noise [18], even for traditional controllers, as we will later show. Therefore, we will also use the obstacle avoidance benchmark to compare the controller obtained with our proposed algorithm to a potential field controller [19], a common approach in mobile platforms with limited sensing.

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 controllers. Section III provides some background on PSO and OCBA in order to facilitate the explanation of the subsequent algorithms. Section IV describes the algorithms incorporating OCBA to PSO for multi-robot implementations and their differences with the ones from the literature. Section V presents the results from applying the different algorithms for learning in simulation. In Section VI, we show the results from real

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

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

robot experiments where we compare the learned controller to a potential field one. Finally, Section VII 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 [8], [15], [16], [18], [20], and it requires basic sensors and actuators available in most mobile robots.

We use the metric of performance introduced in [15], which was also used in [8], [18], [20]. It consists of three factors, all normalized to the interval [0, 1]:

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

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

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

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

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k, $i_{max,k}$ 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.

We conduct experiments in a square arena of 2 m x 2 m with walls, where a different number of cylindrical obstacles of diameter 10 cm are added (0 to 15 obstacles). In simulation, the obstacles are randomly repositioned before each fitness evaluation, and the initial robots' positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. In real robot experiments, the obstacles are randomly positioned the first time, and then kept in this fixed position for the rest of the experiments, while the robots are manually repositioned in random locations between evaluations.

All experiments are conducted with four Khepera III robots. The Khepera III mobile robot is a differential wheeled vehicle 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. Robots can be seen in Figure 1 in the arena with 15 obstacles. The learning is performed in simulation using Webots [21], a high-fidelity submicroscopic simulator that models dynamical effects such as friction and inertia.

In addition to the learned controllers, in this article we employ a manually designed controller based on potential fields. This controller has two purposes: firstly, to show that the obstacle avoidance task in this cluttered environment and with this limited sensing range is challenging in terms of performance and noise, independently of the learning; and secondly, to provide a baseline performance value for comparisons with the learned controllers. The total virtual force generated by the potential field controller is the weighted sum of a constant vector moving the robot forwards and repulsive proportional forces to obstacles measured by any of the infra-red proximity sensors. This resulting virtual force is translated into wheel speeds, and the different parameters are optimized manually for maximization of the performance metric.

Fig. 1. Arena with 15 obstacles and four Khepera III robots performing one of the obstacle avoidance algorithms learned.

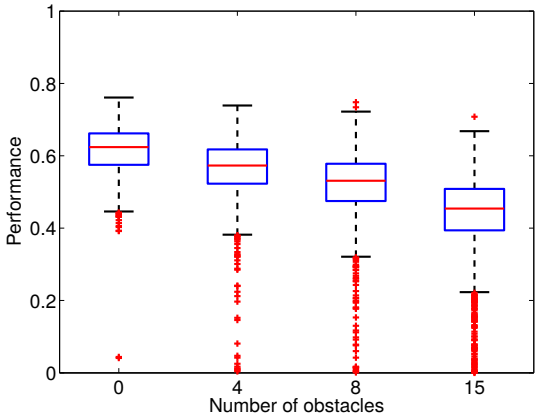


Fig. 2. Performance of the potential field controller tested with an increasing number of obstacles in the arena. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

Figure 2 shows the performance of this controller for 2000 runs in simulation for different obstacle densities. As we increase the number of obstacles in the arena from 0 to 15, the median performance decreases, and both the variance and the number of outliers increase. Outliers represent situations in which the robots get stuck, meaning that with 15 obstacles the avoidance task becomes quite challenging.

For the learning, 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 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.

```
Initialize particles
for N_i iterations do
    for N_p particles do
        Evaluate particle position
        Update personal best
        Update neighborhood best
        Update particle position
    end for
end for
```

Fig. 3. Pseudocode for the standard PSO algorithm.

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

The high-dimensional optimization problem to be solved by the PSO learning 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. Background

PSO is a relatively new 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 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 (t_e =30 s) 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 PSO is shown in Figure 3.

OCBA is a technique based on Bayesian statistics for allocating samples to different candidate solutions introduced by Chen et al. [13]. 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 correctly selecting candidate b as the best (the one with the lowest mean):

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

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 [13].

OCBA has previously been applied to PSO on numerical benchmark functions [4], [14], where it outperformed other techniques for dealing with noise. In the following section, we will describe how we apply OCBA to PSO to allocate samples for a multi-robot task in a centralized manner, and then introduce a distributed version which requires only local information and communication.

IV. Learning Algorithms

As a baseline for our work, we will use the following three algorithms previously presented in the PSO literature:

- *PSO std*, a standard PSO as described in Figure 3, Section III.
- *PSO rep*, the naïve approach of evaluating every new candidate a fixed number of times [14], in this case 10 (determined experimentally on preliminary runs),
- *PSO pbest*, the noise-resistant variant by Pugh et al. [8] which re-evaluates personal bests at each iteration.

Table I shows the parameters that are common to all PSO algorithms used in this paper. They are set following the guidelines for limited-time adaptation presented in [22]. Table II shows the parameters that vary between the three previously mentioned variants, and serves as a quick summary of the differences between them.

In order to perform fair comparisons, the total number of evaluations for each algorithm was held constant, which implies that the number of iterations was set to be inversely proportional to the number of evaluations performed at each iteration. This is why 500 iterations of *PSO std* were performed, 50 for *PSO rep*, and 250 for *PSO pbest*.

The idea behind the application of OCBA to PSO is to fix some issues of the previous algorithms which affect their

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	0.8
V_{max}	20

TABLE II

PARAMETERS FOR *PSO std*, *PSO rep*, AND *PSO pbest*

Parameter	*PSO std*	*PSO rep*	*PSO pbest*
Evaluations of new candidates	1	10	1
Re-evaluations of pbests	0	0	1
Iterations N_i	500	50	250

performance under the presence of noise. 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 [14]. 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 [8] 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 [1].

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 pseudocode for the centralized version of PSO OCBA, *PSO ocbaC*, is shown in Figure 4. Most steps are similar to *PSO rep*, but instead of evaluating every new position 10 times in the evaluation step, it allocates the same evaluation budget of 240 samples (24 particles times 10 evaluations) 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 (48 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 parameters for *PSO ocbaC* are shown in Table III. Again, the number of iterations was calculated to have the same total number of evaluations as the other algorithms, and in this case it is 50, the same value as *PSO rep* since they both perform 240 evaluations per iteration.

```
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. 4. Pseudocode for the *PSO ocbaC* algorithm.

TABLE III

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 Δ	4	1

The pseudocode for the distributed version of PSO OCBA, *PSO ocbaD*, is shown in Figure 5. In this case, each particle is 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. [8], 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 Equations 7 and 8. 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 III. 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

```
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. 5. Pseudocode for the *PSO ocbaD* algorithm.

samples Δ 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 following equations:

$$\bar{X}_n = \frac{(n-1)\bar{X}_{n-1} + X_n}{n} \quad (9)$$

$$\sigma_n^2 = \frac{(n-2)}{(n-1)}\sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} \quad (10)$$

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.

V. Performance of Algorithms in Simulation

We compared the performance of the algorithms on the previously described obstacle avoidance benchmark with 15 obstacles. We chose 15 obstacles based on previous work where we showed that learning in the most cluttered environment generalizes well to simpler ones [23]. Due to the stochastic nature of PSO, we repeated each algorithm for 20 runs for statistical significance.

Figure 6 shows the progress of the learning for a single run of each of the five algorithms. We selected these runs because we considered them to be representative of the behavior of each algorithm, and they will allow us to discuss certain characteristics that cannot be appreciated in the aggregated results for all runs presented later in this section.

The red curve in Figure 6 is the performance of the best solution found so far as estimated by the algorithm and stored in its internal state. 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, during the learning we store the positions of the best solutions found at each iteration. After the learning is finished, we perform 100 a-posteriori evaluations of each of the stored best solutions. We then calculate the mean of the 100 evaluations at each iteration and consider this as the "ground truth" performance of that solution (blue curve).

In general, the estimated performance is higher than the ground truth due to the fact that the learning tries to maximize the performance, and therefore evaluations with positive noise (values higher than the mean) are more likely to be selected than evaluations with negative noise.

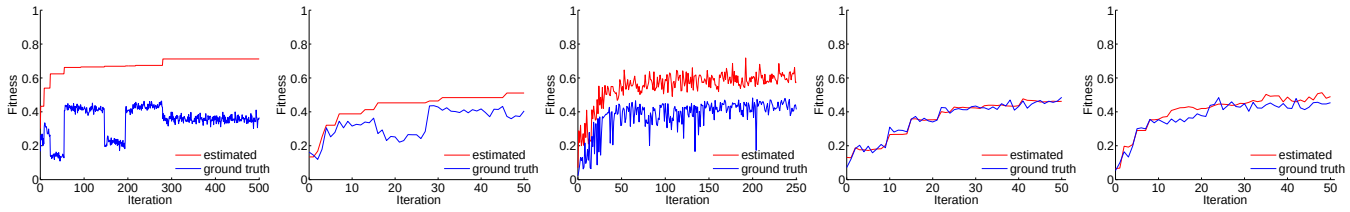
We can see that the largest discrepancy between estimated and ground truth performances occurs for *PSO std* (Figure 6a), which bases its estimate on a single sample of each solution. In addition, even though the estimated performance for *PSO std* is monotonically increasing, the actual performance is rather erratic, with jumps and drops, and stagnates for several number of iterations. These effects are partially mitigated in *PSO rep* (Figure 6b) through the multiple evaluations, but jumps, drops, and stagnation still occur.

PSO pbest (Figure 6c) has a more sustained improvement, interrupted by sudden drops. These drops are due to the fact that *PSO pbest* does not re-evaluate new positions, and therefore a lucky evaluation of a bad new solution can displace a more consistent older candidate. Both *PSO ocbaC* (Figure 6d) and *PSO ocbaD* (Figure 6e) show much more consistent improvement along each iteration, with a good estimate of the ground truth performance and few drops or jumps.

Figure 7 shows the averaged results over the 20 runs for each of the five algorithms. In this case, the individual jumps and drops are averaged out as they occur at different iterations, but the difference between the estimated and ground truth performances becomes evident for the first three algorithms. It is also clear that even though the distributed implementation with a limited communication neighborhood of *PSO ocbaD* outperforms the first three algorithms in terms of estimating the ground truth, it does not perform as well as the centralized *PSO ocbaC*.

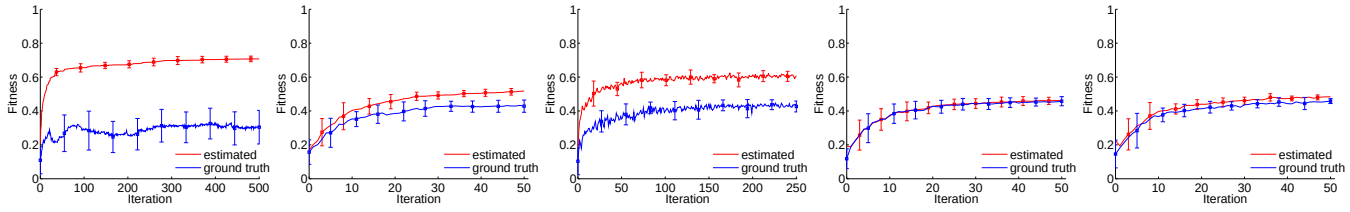
VI. Experiments with Real Robots

In order to validate the results obtained from learning in simulation we tested the best controller from the *PSO ocbaD*



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

Fig. 6. Progress during a single run for each of the five algorithms. 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. (a) PSO std. (b) PSO rep. (c) PSO pbest. (d) PSO ocbaC (e) PSO ocbaD.



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

Fig. 7. Progress averaged over 20 runs for each of the five algorithms. 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) PSO std. (b) PSO rep. (c) PSO pbest. (d) PSO ocbaC (e) PSO ocbaD.

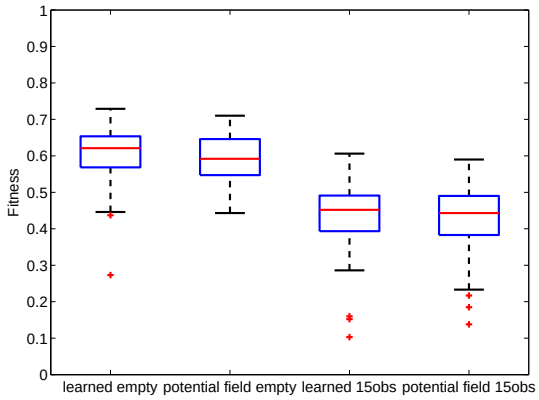


Fig. 8. Potential field and learned controllers tested with 0 and 15 obstacles. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

approach and a traditional potential field controller in the two extremes of the benchmark task presented in Section II: arena with no obstacles and arena with 15 obstacles. In the latter case, the obstacles were placed in random positions and then kept fixed for all the runs. We conducted 20 runs with 4 robots for each controller and environment, which results in 80 performance measurements per setting. The results from these experiments are shown in Figure 8.

Both controllers perform well in both environments and do not collide with walls, obstacles, or other robots. The median performance of the learned controller is slightly higher than the potential field one in both cases, although it is only statistically significant in the environment with no obstacles (Mann Whitney U test, $p = 0.02$ for no obstacles and $p = 0.80$ for 15 obstacles). This difference is due to the fact that when the potential field controller is approaching an obstacle or wall straight ahead, it gradually slows before turning, which results in lower speed factor and more time spent near obstacles (lower proximity factor). We also observed that the potential field controller oscillated in place or got stuck when there was a narrow passage between an obstacle and a wall.

On the other hand, the learned controller switched between two states: moving forwards at full speed and turning counter-clockwise in place at full speed. This behavior of switching between boundary points in the control space (*bang-bang control*) is similar to the behaviors seen when applying optimal control theory to minimize trajectory time for bounded velocity differential drive vehicles [24]. In addition, the fact that robots have a preferred turning direction (always turn counter-clockwise) avoids getting stuck, resulting in a more robust controller.

VII. Conclusion

We have applied OCBA, a statistical technique for sampling budget allocation, to improve the performance of PSO in the presence of noise for the high-dimensional optimization of controllers for multiple robots with limited resources. In addition to the centralized budget allocation, we have introduced a new distributed PSO OCBA algorithm suitable for resource-constrained mobile robots since it requires only local communication and sensing, and the amount of information shared is limited and constant for the whole learning process.

Results in an obstacle avoidance benchmark show that both PSO OCBA variants outperform other techniques for dealing with noise from the literature, achieving a more consistent progress and a better estimate of the ground-truth

performance of candidate solutions.

In order to validate our simulations, we compared on real robots the controller learned with our proposed algorithm to a potential field controller from the literature. They both achieved a high performance through different avoidance strategies.

As future work, we would like to apply this algorithm to different tasks in the multi-robot domain to see how the results obtained in this paper generalize to other robotic problems. In addition, we would like to explore the effect of different parametrizations and variations of the distributed PSO OCBA.

REFERENCES

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

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

[9] 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.

[10] L. Marques, U. Nunes, and A. T. Almeida, "Particle swarm-based olfactory guided search," *Autonomous Robots*, vol. 20, no. 3, pp. 277–287, 2006.

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

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

[13] 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.

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

[15] 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.

[16] 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.

[17] K. A. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. dissertation, University of Michigan, 1975.

[18] 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.

[19] R. C. Arkin, "Motor Schema – Based Mobile Robot Navigation," *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[20] 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.

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

[22] 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.

[23] 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 *European Conference on the Synthesis and Simulation of Living Systems, Advances in Artificial Life*, 2013, pp. 561–568.

[24] D. Balkcom and M. Mason, "Time Optimal Trajectories for Bounded Velocity Differential Drive Vehicles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 199–217, 2002.