Distributed Particle Swarm Optimization for Limited Time Adaptation with Real Robots

Ezequiel Di Mario and Alcherio Martinoli

Abstract Evaluative techniques offer a tremendous potential for on-line controller design. However, when the optimization space is large and the performance metric is noisy, the overall adaptation process becomes extremely time consuming. Distributing the adaptation process reduces the required time and increases robustness to failure of individual agents. In this paper, we analyze the role of the four algorithmic parameters that determine the total evaluation time in a distributed implementation of a Particle Swarm Optimization algorithm. For an obstacle avoidance case study using up to eight robots, we explore in simulation the lower boundaries of these parameters and propose a set of empirical guidelines for choosing their values. We then apply these guidelines to a real robot implementation and show that it is feasible to optimize 24 control parameters per robot within 2 hours, a limited amount of time determined by the robots' battery life. We also show that a hybrid simulate-and-transfer approach coupled with a noise-resistant PSO algorithm can be used to further reduce experimental time as compared to a pure real-robot implementation.

Ezequiel Di Mario and Alcherio Martinoli

Distributed Intelligent Systems and Algorithms Laboratory, School of Architecture, Civil and Environmental Engineering, École Polytechnique Fédérale de Lausanne.

1015 Lausanne, Switzerland

e-mail: ezequiel.dimario@epfl.ch, alcherio.martinoli@epfl.ch.

1 Introduction

Human design of high-performing robotic controllers is not a trivial task for a number of reasons. In the first place, even the simplest of modern robots have a large number of sensors and actuators, which implies a large number of control parameters to optimize. Secondly, real systems often present discontinuities and nonlinearities, making it difficult to apply well-understood linear control techniques. Finally, when porting the designed controller to real robots there might be an unexpected performance drop due to a number of factors such as imperfections in fabrication, changes in the environment, or modeling inaccuracies.

Machine-learning techniques are an alternative to human-guided design that can address the previously mentioned challenges. In particular, population-based, evaluative, metaheuristic methods can deal with high-dimensional search spaces with discontinuities and nonlinearities, such as the ones used in optimization competitions.[1] Furthermore, the learning process can be implemented fully on-board real robots,[2,3] which enables automatic adaptation to the underlying hardware and environment, and has the potential to find innovative solutions not foreseen by human designers.

However, the main drawback of working with a pool of candidate solutions in an on-line, evaluative framework is the amount of time needed to evaluate all candidates, which is substantially larger than that required to generate them. Moreover, due to several sources of uncertainty, such as sensor noise, manufacturing tolerances, or lack of strict coordination in multi-robot settings, it may be necessary to re-evaluate some solutions to build sufficient statistics for meaningful adaptation. Because of these two reasons, large evaluation time and learning disruption due to uncertainties in the performance measurement, the adaptation process is considered an expensive optimization problem.

Implementing the adaptation process in a distributed fashion brings two distinct advantages. Firstly, it reduces the required total evaluation time through parallel evaluations. Secondly, it increases robustness by avoiding a critical point of failure, which is of particular interest in real multi-robot implementations.

Thus, the goal of this paper is to analyze how different algorithmic parameters in a distributed implementation affect the total evaluation time and resulting fitness. We aim to reduce the total evaluation time such that it is feasible to implement the adaptation process within the limits of the robots' battery life without renouncing the benefits of a population-based learning algorithm. Even though the focus of this work is on the algorithmic parameters, we use obstacle avoidance as a benchmark task to show the potential of these techniques for concrete multi-robot applications, in particular for dense environments and with limited on-board resources.

We then apply our proposed guidelines to a real robot distributed implementation of Particle Swarm Optimization. We use trajectory analysis to characterize the best solution strategies. We also analyze the performance differences between simulation and reality, focusing on the different noise distributions in fitness evaluations. Finally, we compare pure real robot optimization with a hybrid simulate-and-transfer method which reduces the required real robot experimental time.

2 Related Work

Particle Swarm Optimization (PSO) is a relatively new metaheuristic originally introduced by Kennedy and Eberhart.[4] PSO is inspired by the movement of flocks of birds, and represents a set of candidate solutions as a swarm of particles moving in a multi-dimensional space. Because of its simplicity and versatility, PSO has been used in a wide range of applications such as antenna design, communication networks, finance, power systems, and scheduling. Within the robotics domain, popular topics are robotic search, path planning, and odor source localization.[5]

PSO is well suited for distributed/decentralized implementations due to its distinct individual and social components and the use of the neighborhood concept. Most of the work on distributed implementations has been focused on benchmark functions running on computational clusters,[6,7] as opposed to implementations on real robots as the one used in this paper.

Moreover, implementations with mobile robots are mostly applied to odor source localization,[8,9] and robotic search.[10] In these applications, the particles' position is usually directly matched to the robots' position in the arena (as opposed to this article where we optimize a set of control parameters for the task at hand). Thus, the search is conducted in two dimensions and with few or even only one local extrema. For these reasons, even though robotic search is a challenging practical task, it does not represent a complex optimization problem.

An example of a more challenging on-line optimization problem is the work of Floreano and Mondada,[2] who used Genetic Algorithms to optimize the weights of an artificial neural network controller. The task was to navigate a path and avoid obstacles with a tethered mobile robot. Even though the population manager and other resource-intensive tasks were carried out on a dedicated off-board computer, this study was still able to show the advantages of evaluation with hardware in the loop. For example, the evolved direction of motion was a result of the interplay between the robot morphology (higher density of proximity sensors facing forward) and the environment in which it was deployed. It is worth noting that the experiment required 67 hours of total evaluation time, and it would require the same time to recreate it nowadays since the limit was not imposed by computational capabilities but rather by the wall-clock time needed to gather enough information on the quality of the candidate solutions. This observation motivates this paper's emphasis on reducing evaluation time.

Most of the research on optimization in noisy environments has focused on evolutionary algorithms.[11] In particular, Arnold and Beyer studied the effect of different

noise distributions on the performance of evolution strategies, concluding that significant differences occur when the noise distribution assumed to be Gaussian is skewed, biased, or presents unbounded variance.[12]

The performance of PSO under noise has not been studied so extensively. 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.[13] Pan et al. proposed a hybrid PSO-Optimal Computing Budget Allocation (OCBA) technique for function optimization in noisy environments.[14]However, these two studies were conducted on benchmark functions with an additive Gaussian noise model, and one of the goals of this paper is to test whether this assumption holds for the noisy performance evaluations found in multi-robot learning.

Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions and for certain scenarios of limited-time learning under the presence of noise,[15,16] which is why we chose PSO for this research. Pugh also showed that PSO can perform satisfactorily with low population sizes, a result that is of particular interest for our work because a smaller number of robots can be used while leaving the optimization process robust to connectivity issues between the robots.

In our recent work,[17] we have compared PSO with Q-Learning for the same multi-robot obstacle avoidance benchmark task used in this paper, showing that both algorithms could achieve similar performances if a continuous state representation was used for Q-Learning.

3 Materials and Methods

This paper discusses the role of the algorithmic parameters that determine total evaluation time in distributed PSO implementations. In order to perform our analysis, we use multi-robot obstacle avoidance as a benchmark task. We chose this task because

it can be implemented with different number of robots (though the performance depends on the density of robots in the arena), requires basic sensors and actuators that are available in most mobile robots, and the chosen performance metric can be fully evaluated with on-board resources. Thus, it can serve as a benchmark for testing distributed learning algorithms with real robots in the same way standard benchmark functions are used in numerical optimization.[1, 18]

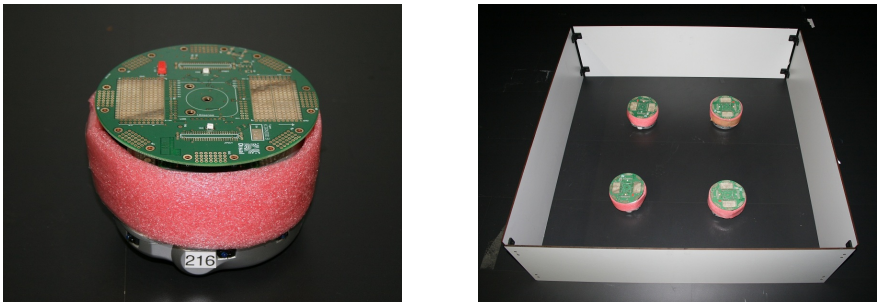
Robots navigate autonomously in a square arena in which walls and other robots are the only obstacles. We use the same metric of performance as Floreano and Mondada,[2] which consists of three factors, all normalized to the interval [0, 1] (Eq. 1).

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

V is the average wheel speed, Δv the wheel speed difference, and i the proximity sensor activation value of the most active sensor. 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. This function, when maximized, rewards robots that move quickly, turn as little as possible, and spend as little time as possible near obstacles.

Our experimental platform is the Khepera III mobile robot, a differential wheeled vehicle with a diameter of 12 cm. The Khepera III is equipped with nine infrared sensors as well as five ultrasound sensors for short and medium range obstacle detection. An extension board with two color LEDs is added for tracking purposes. We also added a soft plastic cover to protect the robots during collisions, especially during the early phases of the learning process (Figure 1a).

Simulations were performed in Webots,[19] a realistic physics-based submicroscopic simulator that models dynamical effects such as friction and inertia. In this context, by submicroscopic we mean that it provides a higher level of detail than usual microscopic models, faithfully reproducing intra-robot modules (e.g., individual sensors and actuators). Experiments with real robots take place in a square arena (Figure 1b), equipped with an overhead camera connected to a computer running



(a) (b)

Fig. 1 Experimental setup. (a) Khepera III robot with protective cover and tracking board. (b) Four robots in the square arena.

SwisTrack,[20] an open source tracking software that enables us to log the position and orientation of robots in the arena. The arena size is 1x1 m unless noted otherwise.

The controller architecture is an artificial neural network of two units with sigmoidal activation functions, and a single output per unit to control the two motors. 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.

The optimization problem to be solved by the adaptation algorithm is to choose the set of weights of the artificial neural network controller such that the fitness function F as defined in Equation 1 is maximized. The algorithm we employed is the noise-resistant variation of PSO introduced by Pugh et al.[15] This variant operates by re-evaluating personal best positions and aggregating them with the previous evaluations (in our case a regular average performed at each iteration of the algorithm), which results in a more accurate estimate of the performance metric in the presence of noise, but requires twice as many evaluations at each iteration. The pseudocode for the algorithm is shown in Figure 2, where N_i is the number of iterations for PSO, N_p is the total number of particles, and N_{rob} is the number of robots.

```
1: Intialize particles
2: for N_i iterations do
3:     for ⌈N_p/N_rob⌉ particles do
4:         Update particle position
5:         Evaluate particle
6:         Re-evaluate personal best
7:         Aggregate with previous best
8:         Share personal best
9:     end for
10: end for
```

Fig. 2 Noise-resistant PSO algorithm

The position of each particle represents a set of weights of the neural network controller. Each particle evaluation consists of a robot moving in the arena for a fixed time t_{eval} running the neural controller with the weights given by that particle's position. The fitness corresponding to the particle is equivalent to the performance of the robot measured with function F from Equation 1.

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. 2). $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$v_{i,j} = w \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \tag{2}$$

The neighborhood presents a ring topology with one neighbor on each side. Particles' positions and velocities are initialized randomly with a uniform distribution in the [-20, 20] interval, and their maximum velocity is also limited to that interval. At the beginning of each evaluation, a random speed is applied to the wheels for 3 s

to randomize the robot's pose and reduce the influence of the previous evaluations. At the end of each optimization run, the best solution is tested with 40 evaluation runs of 20 s, and the final performance is the average of these final evaluations.

The total evaluation time for PSO depends on four factors: population size (N_p), individual candidate evaluation time (t_e), number of iterations of the algorithm (N_i), and number of re-evaluations of the personal best position associated with each candidate solution within the same iteration (N_{re}), as shown in Eq. 3.

$$t_{tot} = t_e \cdot N_p \cdot N_i \cdot (N_{re} + 1) \tag{3}$$

If we assume that there is a fixed upper limit for the total evaluation time, an increase in any of the parameters would result in a proportional decrease in the rest.

In a parallelized or distributed implementation, fitness evaluations are distributed among N_{rob} robots, and the wall-clock time t_{wc} required to evaluate candidate solutions is reduced (Eq. 4).

$$t_{wc} = t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot N_i \cdot (N_{re} + 1) \tag{4}$$

It is worth noting that the number of robots is not necessarily the same as the population size. In fact, the choice of the population size depends on the dimension of the search space and the complexity of the task, while the choice of number of robots is based on more experimental considerations (e.g., availability of robots, targeted number of robots needed for a specific mission in a given environment). Thus, a robot may have several particles to evaluate within the same candidate solution pool as opposed to only one.

A full optimization of the algorithmic parameters to minimize the total evaluation time would be a computationally very expensive problem, due to the large number of candidate configurations, the combination of continuous and discrete parameters, the large variation between runs, and the possible existence of local optima. Thus,

our approach is to analyze each parameter individually, taking into account its impact on the final performance as compared to a full-time adaptation baseline.

Our baseline set of parameters, based on the work of Pugh et al,[16] is shown in Table I. This set of parameters amounts to a total evaluation time of approximately 417 hours if carried out on a single robot, what we refer to as full-time adaptation.

To complement our robotic case study and add more generality, we also perform runs on four traditional mono and multi-modal benchmark functions without noise: the sphere, Rosenbrock's, Rastrigin's, and Griewank's, defined in Eq. 5. The baseline parameters for the algorithm ran on benchmark functions are also shown in Table I.

$$
\begin{aligned}
f_1(\mathbf{x}) &= \sum_{i=1}^{D} x_i^2 \\
f_2(\mathbf{x}) &= \sum_{i=1}^{D-1} [(1-x_i^2) + 100(x_{i+1} - x_i^2)^2] \\
f_3(\mathbf{x}) &= 10D + \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i)] \\
f_4(\mathbf{x}) &= 1 + \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}})
\end{aligned}
\tag{5}
$$

Table I Algorithmic parameter values

Parameter	Obstacle Avoidance	Benchmark functions
Population size N_p	100	100
Iterations N_i	50	500
Evaluation span t_e	150 s	-
Re-evaluations N_{re}	1	0
Personal weight w_p	2.0	2.0
Neighborhood weight w_n	2.0	2.0
Dimension D	24	24
Inertia w	0.8	0.6
V_{max}	20	5.12

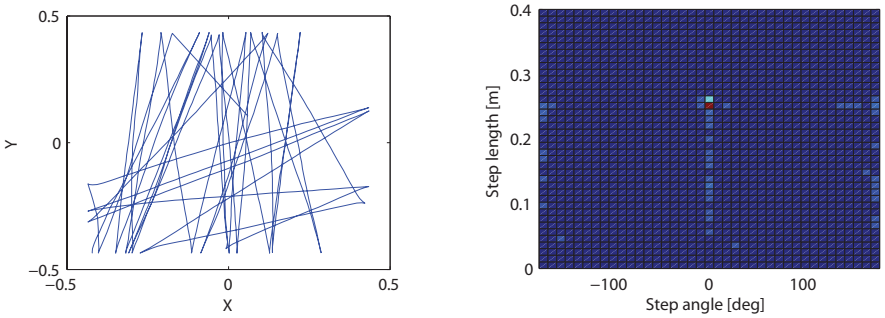
4 Results and Discussion

The results of this paper are presented as follows: Section 4.1 explores the best solution strategies in simulation for different number of robots and arena sizes. Section 4.2 introduces the analysis of the algorithmic parameters with a comparison of the fitness as a function of the total evaluation time. In Sections 4.3 to 4.6 we analyze each of the four previously mentioned algorithmic parameters and propose guidelines for setting their values. In Section 4.7 we apply the proposed guidelines to reduce the total evaluation time in simulation and compare the results with the full-time adaptation. Section 4.8 discusses how controllers optimized in simulation perform when transferred to real robots and analyzes the fitness distributions of the evaluation runs. In Section 4.9 we implement the proposed guidelines with real robots and we compare the optimization performed exclusively on real robots with a hybrid simulate-and-transfer approach. Finally, Section 5 concludes the paper with a summary of our findings and presents an outlook of our future work.

4.1 Best Solution Strategies

Consider a single robot in an empty arena. A priori, we can think of two different obstacle avoidance strategies: going back and forth between walls, and circling around the center of the arena. Determining the dominant solution in a multi-robot setting is less straightforward. In simulation, we explored the best solution strategies by analyzing the trajectories described by the best performing controllers, focusing on the step length and angle distributions as described in our previous work.[21]

The best solution strategy observed both in the single and multi-robot case consisted in going back and forth between walls in a straight line, reversing the direction of motion every time an obstacle was detected (Figure 3a.) This strategy is not surprising when we consider that the fitness function does not penalize going back-



(a) (b)

Fig. 3 (a) A sample trajectory described by the best solution strategy. (b) Its step length and angle distribution.

wards but penalizes turning. Therefore, by exploiting the recurrence of the artificial neural network architecture the robot can perform a quick reversal of direction with little penalty.

Figure 3b shows the bidimensional step length and angle distribution of the best trajectory. Most steps consists of the robot moving straight at maximum speed (step length=25cm, step angle=0°). The remaining steps represent the robot decelerating when reversing direction (0<step length<25cm, step angle=0°), and sharp 180° turns.

We looked for circular solutions by searching for high absolute values of the median step angle. We chose this criterion as opposed to the mean step angle as it is less sensitive to the sharp turns that occur when encountering obstacles. A negative median step angle implies clockwise turning, a positive one counter clockwise turning, and a value of zero moving in straight lines. We wanted to determine if the circular behavior could arise automatically under different environmental conditions. We varied the length of the square arena from 0.4 m to 20 m and the number of robots from 1 to 8, but the best solution found for all cases was moving in straight lines. Circular behaviors became the best strategy only when going backwards was explicitly penalized in the fitness function.

4.2 Initial Parameter Comparison

As a first step in the analysis of the algorithmic parameters, we started from the total evaluation time baseline of 417 h and reduced N_p, N_i, and t_e individually to 5, 5, and 5 s respectively, while keeping the other two parameters at their baseline values, plotting the three curves in the same graph for better comparisons. We performed 100 independent runs for each set of parameter values and with 1, 2, 4, and 8 robots. When multiple robots were considered, all of them were learning in parallel. Results are shown in Figure 4. The fitness presented in Figure 4 and in all fitness Figures in Sections 4.2 to 4.7 are the final performances (mean performance of the best solution over the 40 final evaluations) averaged for 100 optimization runs. Error bars represent the standard deviation among the 100 optimization runs.

In all cases, reducing the evaluation span t_e has the least impact on the resulting fitness, followed by N_p and N_i. When comparing the same total evaluation time across different numbers of robots, it can be noted that as the number of robots increases, the arena becomes more crowded and obstacle avoidance becomes harder, thus causing the average fitness to decrease. Also, performances are noisier (see larger error bars in lower right corner) and therefore there is less impact of a decreased N_p or N_i (flatter profile than with 1-2 robots). The following sections present a more detailed analysis of each individual parameter.

4.3 Evaluation Span

To analyze the effect of the evaluation span, we reduced t_e from 150 s to 5 s, with 20 s steps in the higher range and 5 s steps in the low range. We did 100 runs for each value, and plotted the mean and standard deviation in Figure 5, left.

The mean fitness remains fairly constant for $30\,\text{s} < t_e < 150\,\text{s}$ for all number of robots. In fact, the difference in fitness between 150 s and 30 s is not statistically

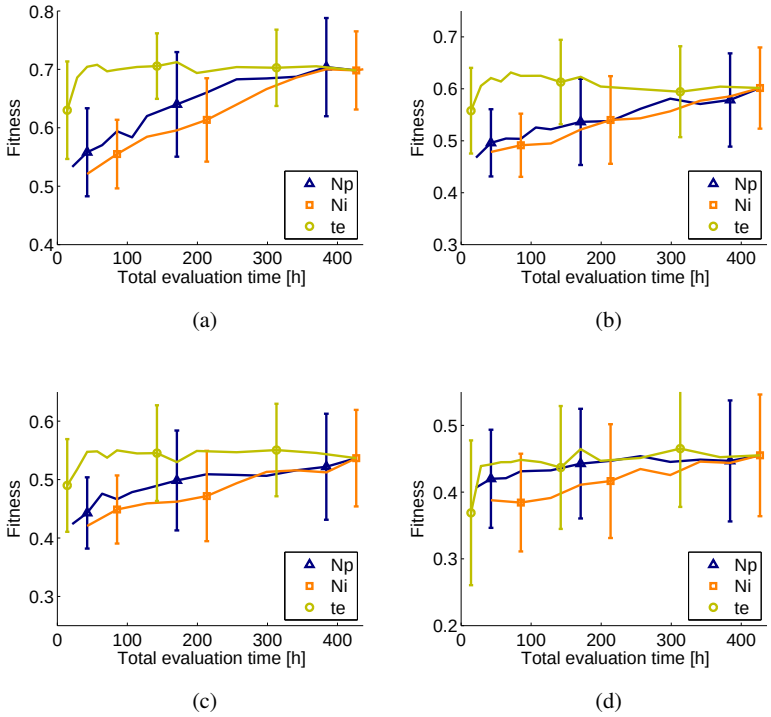
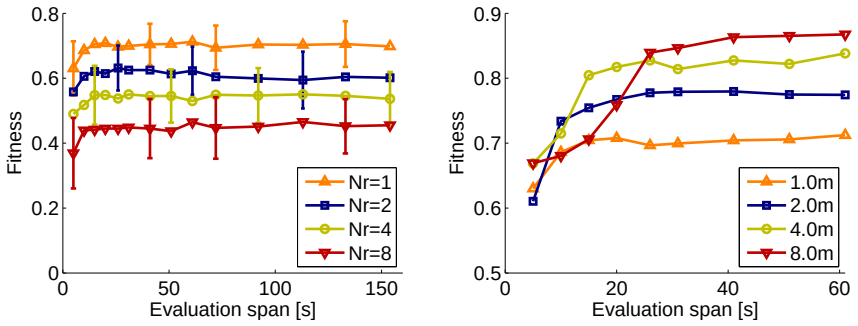


Fig. 4 Fitness as a function of total evaluation time for 1, 2, 4, and 8 robots corresponding to labels (a) to (d) respectively. Each curve represents the reduction of one individual parameter (population size, number of iterations, and evaluation span) with the others held constant.

significant in all cases (Mann-Whitney U test, 5% significance level). For $t_e < 30\,\mathrm{s}$ the fitness starts to decrease, although at different rates for different numbers of robots. In particular, for 8 robots t_e can be reduced to 10 s without a major change in fitness, which suggests that a crowded arena may allow for shorter evaluation spans due to more frequent collisions, and thus more opportunities to learn to avoid them. It is interesting to note that reducing the evaluation span does not seem to increase the fitness variation between runs.

The evaluation span parameter depends on the task and the environment. With the goal of trying to explain the lower limits for our task, we varied the evaluation span for several arena sizes using one robot (Figure 5, right). In this case, the point



(a) (b)

Fig. 5 (a) Mean fitness for different evaluation span values and number of robots in a 1x1 m arena. (b) Mean fitness for different evaluation span values and arena sizes with a single robot (y axis zoom in the 0.5 to 0.9 range).

where performance starts to drop occurs at longer evaluation spans for larger arena sizes (15 and 25 seconds for 4 and 8 meters respectively). We suspect this point is related to the minimum time it takes to have at least one collision. In fact, if the robot moves in a straight line at a maximum speed of 0.25 m/s, it takes 16 and 32 seconds to cross one side of an arena of 4 and 8 meters respectively. Therefore, the robot speed and environment size can be used as guidelines to choose an evaluation span. We suggest that, in general, the minimum evaluation span should guarantee at least one interaction of the robot with other components of the environment relevant to the task at hand.

4.4 Re-evaluations

We then compared the performance of noise-resistant PSO with standard PSO to determine if re-evaluations improve performance in limited time scenarios. For any given set of parameters, noise-resistant PSO takes twice as much evaluation time as standard PSO due to the personal best re-evaluations. In order to perform a fair

comparison, if we remove re-evaluations we need to double one of the other parameters to keep total evaluation time constant. We thus compared four alternatives: re-evaluations, doubled iterations, doubled evaluation span, and doubled population size. We performed 100 runs for each algorithmic variant and plotted the final fitness in Figure 6.

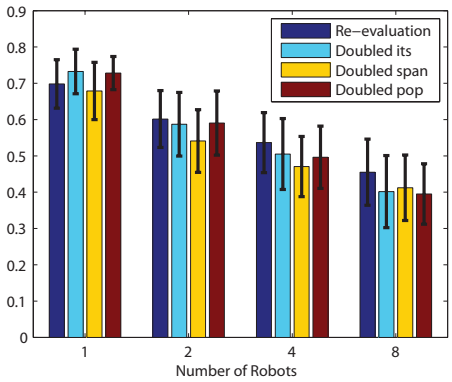


Fig. 6 Average fitness and standard deviation for noise-resistant PSO, standard PSO with doubled number of iterations, standard PSO with doubled evaluation span, and standard PSO with doubled population size.

In the single robot case, noise-resistant PSO performed significantly worse than standard PSO with doubled iterations and doubled population size. However, as the number of robots is increased, the relative performance of noise-resistant PSO improves: for 2 robots there is no significant difference, and for 4 and 8 robots noise-resistant PSO significantly outperforms standard PSO with doubled iterations and doubled population size. It is worth noting that there is no significant difference between doubling population size and number of iterations for all number of robots, and that doubling the evaluation span performs significantly worse in all cases except for 8 robots.

These results suggest that the decision to invest time in re-evaluations depends on the amount of uncertainty in fitness evaluations. In fact, as the arena becomes

more crowded, there is more uncertainty in fitness evaluations, which depend both on the performance of other robots (avoiding other robots is easier if other robots are also trying to avoid you) and on initial conditions such as position in the arena (a robot is more likely to be trapped against a corner in a crowded arena).

Re-evaluations may also reduce the effect of heterogeneities on solution sharing in multi-robot evaluations, as shared solutions are re-evaluated at each iteration and thus can be dropped if they do not perform as well as they had done on other robots. The final advantage of re-evaluations can be seen in the case of dynamic environments, where a previously found good solution may no longer be valid after a certain amount of time. Thus, re-evaluations seem to be a good recommendation in general for multi-robot learning scenarios.

4.5 Population Size

Both for our robotic case study and the benchmark functions, the population size was reduced from 100 to 30 in steps of 10, and from 30 to 5 in steps of 5, in order to obtain more data points in what we expected to be an interesting region, while keeping all the other parameters the same as in the baseline. We did 100 independent runs for each value, results are shown in Figure 7, left and Figure 8.

It is clear from Figures 4 and 8 that, at least with our baseline parameters, reducing the population size is better in terms of mean fitness than reducing the number of iterations, both for obstacle avoidance and for all benchmark functions (note that in benchmark functions lower fitness values mean better performance). Now the question that arises is how low should we set the population size? While there is no clear consensus in PSO literature,[22] there are a few guidelines based on the dimension D of the search space such as $N_p = D$ or $N_p = 10 + 2\sqrt{D}$ (this last formula is used in

Standard PSO 2006, an effort to define a PSO standard published online in Particle Swarm Central[1]).

Another approach is to start with a fixed value such as $N_p = 40$ and restart the algorithm with a larger N_p if early convergence is noticed. However, it is hard to determine if a restart is needed, especially when the maximum feasible fitness is not clear beforehand, which is often the case when learning robotic behaviors.

In Figure 7, left we note a slight change in the fitness slope at around $N_p = 25$, but this effect is much more clear in the case of the benchmark functions f_2, f_3, and f_4 (Figure 8, $N_p = 25$ and 500 default iterations, as mentioned in Table I, corresponding to 12500 function evaluations).

Also, when population size becomes small, more outliers appear due to runs that fail to converge to a satisfactory solution. This can be noted in the higher standard deviation seen in reduced N_p as compared to reduced N_i with the same total evaluation time (see Figure 4).

Thus, because of higher fitness, lower variance, the possibility to distribute particles among robots, and the impracticality of the restart process, we prefer to err on the side of larger population sizes, and we suggest the following guideline:

$$N_p = \max(D, N_{rob}) \tag{6}$$

4.6 PSO iterations

For our robotic case study, the number of iterations was reduced from 50 to 5 in steps of 5, again keeping all the other parameters the same as in the baseline. We did 100 independent runs for each parameter value, results are shown in Figure 7, right. The chosen benchmark functions traditionally use larger values of N_i, so we chose 500 as a baseline and reduced it to 50 in steps of 50 (see Figure 8).

[1] `http://www.particleswarm.info`

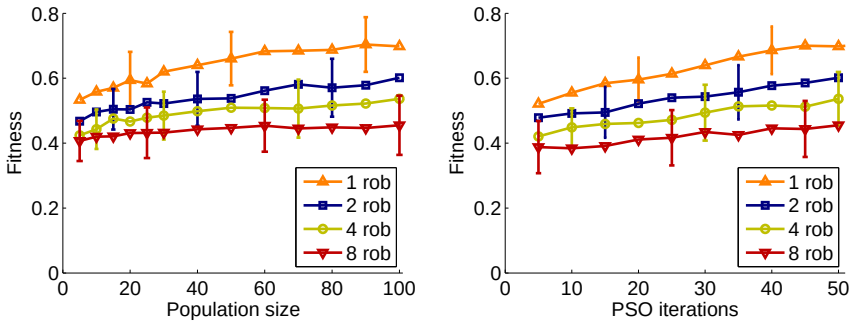


Fig. 7 (a) Mean fitness for different population size values. (b) Mean fitness for different number of iterations.

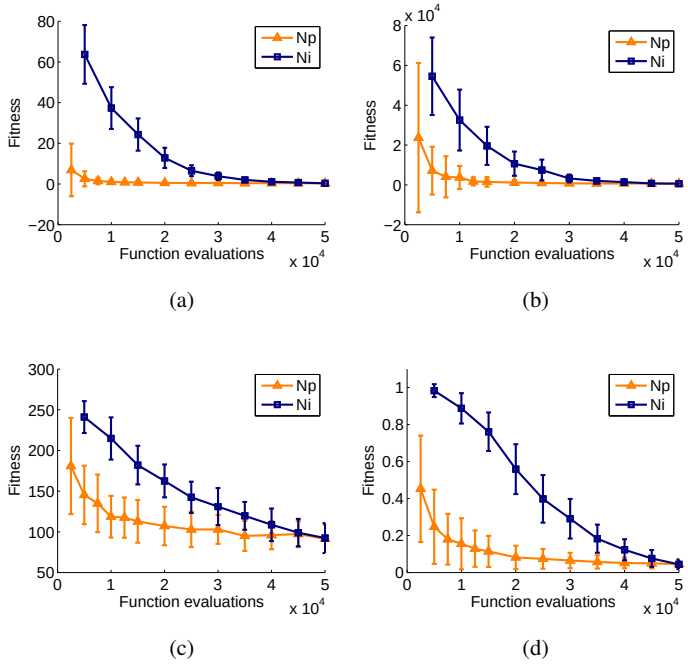


Fig. 8 Fitness as a function of number of evaluations for benchmark functions f1 (a), f2 (b), f3 (c), and f4 (d). Lower fitness is better. Each curve represents the reduction of one individual parameter (population size and number of iterations) with the other held constant.

We observed a nearly linear performance drop for obstacle avoidance and on benchmark functions f_3 and f_4. For f_1 and f_2, the behavior of N_i was similar to that of N_p, but with a worse fitness overall.

Given that N_i is the easiest parameter to adjust on the fly, this parameter seems suitable for trade-offs between performance and available learning time. That is, for a fixed available time t_{wc}, we suggest using the previous guidelines to determine the 3 other parameters and allocate all remaining time to N_i using Eq. 7, which is derived from Eq. 4.

$$N_i = \frac{t_{wc}}{t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot (N_{re}+1)} \tag{7}$$

4.7 Limited Time Adaptation

The Khepera III with tracking board and wireless card has a battery autonomy of about 2 hours. Following our proposed guidelines, we determine the algorithmic parameters to fit a full optimization run with 4 robots in that time frame (see Table II). We set $N_p = 24$, $t_e = 20$ s, $N_{re} = 1$, and $N_i = 30$, and run the adaptation process in simulation for $N_{rob} = \{1,2,4,8\}$. The final fitness (mean performance of the best solution over the 40 final evaluations) averaged for 100 optimization runs is shown in Figure 9.

The mean fitness difference between full-time and limited time adaptation is 17%, 17%, 14%, and 9% for 1, 2, 4, and 8 robots respectively. These values are relatively low considering the evaluation time was reduced more than 52 times. More importantly, both limited and full-time adaptation converged to the same obstacle avoidance strategy, regardless of the number of robots: going back and forth between walls in a straight line, reversing the direction of motion every time an obstacle was detected. We verified the sameness of the solution strategies by ana-

Table II Limited-time adaptation parameter values

Parameter	Value
Population size N_p	24
Iterations N_i	30
Evaluation span t_e	20 s
Re-evaluations N_{re}	1
Personal weight pw	2.0
Neighborhood weight nw	2.0
Dimension D	24
Inertia w	0.8
V_{max}	20

lyzing the trajectories described by the robots, focusing on the step length and angle distributions as described in Section 4.1.

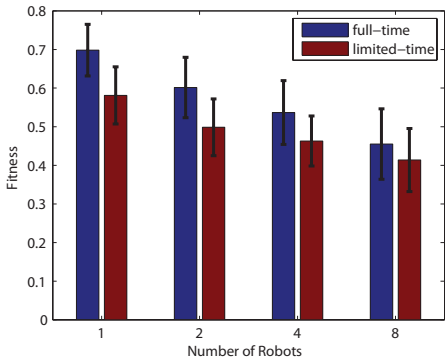


Fig. 9 Average fitness and standard deviation for full-time and limited-time adaptation.

4.8 Evaluation Runs with Real Robots

In order to analyze how well simulated controllers transfer to real robots, we took 5 different best set of weights (resulting from 5 independent optimization runs in sim-

ulation) and tested their performance for 40 evaluation runs each with real robots. Figure 10 shows the performance of the 5 different controllers under 4 evaluation conditions: 1 robot in simulation, 1 robot in reality, 4 robots in simulation, and 4 robots in reality.

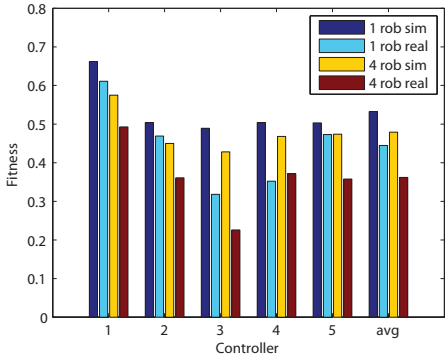


Fig. 10 Mean fitness over 40 evaluation runs for five controllers optimized in simulation and evaluated in simulation and reality with one and four robots.

In general, it can be noted that single robot evaluations perform better than multi-robot ones. This is to be expected, since as the number of robots increases, the arena becomes more crowded and obstacle avoidance becomes harder, especially when there is no explicit coordination scheme among robots.

Also, the performance in simulation is higher than with real robots. In fact, it is interesting to note that the difference in performance between simulation and reality is significantly larger with 4 robots than with 1 robot in all cases except for controller 4 (average performance difference: 24% for 4 robots and 16% for 1 robot).

Further insight on the difference between simulation and reality can be gained by analyzing the fitness evaluation distributions. Figure 11 shows the distributions for controller 3, the one with highest difference between simulation and reality.

For a single robot simulation, the fitness distribution is bell-shaped and has a relatively low variance. There is no statistically significant difference with a Gaussian

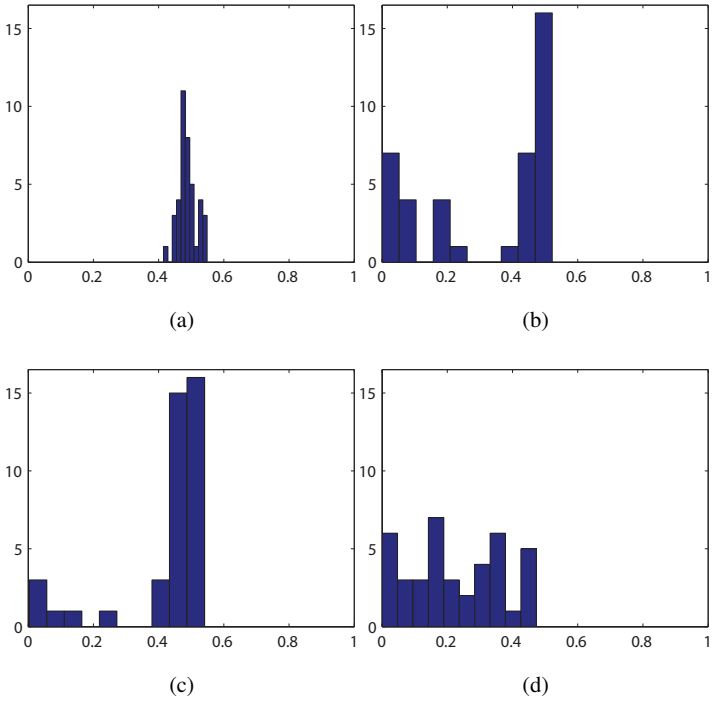


Fig. 11 Fitness distributions for controller 3 under different evaluation conditions. The horizontal axis represents the fitness, the vertical axis the number of evaluations. The cumulative value of all bins is 40, the total number of evaluations. (a) Evaluated in simulation with 1 robot. (b) Evaluated in reality with 1 robot. (c) Evaluated in simulation with 4 robots. (d) Evaluated in reality with 4 robots.

distribution (one-sample Kolmogorov-Smirnov test, $p = 0.531$). This distribution is the result of the added effects of the sensor noise, the actuator noise, and the random initial position in the arena. These are the only sources of uncertainty in the single robot simulation, and they all present Gaussian or uniform distributions.

In the real single robot case, several evaluations have performances similar to the ones in simulation (16 evaluations in the 0.5 fitness bin), but there are a few cases in which the robot gets stuck, resulting in a very poor performance. This effect is probably due to two physical details that were not modelled in the simulation: the plastic cover added to the robots to protect them against collisions, which has a

higher friction against walls and therefore facilitates getting stuck, and the wireless card, which protrudes from the body of the robot causing the back proximity sensor to be less exposed. The difference between the simulated and real distributions is statistically significant (two-sample Kolmogorov-Smirnov test, $p = 4.7e-5$).

In the multi-robot simulation (see Figure 11c), most evaluations have fitness values that are slightly lower than the single robot simulation, but we can notice failed runs that were not seen in the single robot simulation, which decrease the average performance considerably. These failed runs occur when a robot gets blocked in a corner or against a wall by other robots, a new source of uncertainty that does not seem to follow a Gaussian distribution (one-sample Kolmogorov-Smirnov test, $p = 1.2e-4$).

Finally, the multi-robot real evaluations have the lowest performance of all cases, due to the combination of unmodelled effects and the interference between robots. The difference between the simulated and real distributions for 4 robots is statistically significant (two-sample Kolmogorov-Smirnov test, $p = 3.7e-10$) and larger than in the case of 1 robot (Kolmogorov-Smirnov distance of 0.725 and 0.5 respectively).

4.9 Optimization Runs with Real Robots

In Section 4.8 we have shown a noticeable performance drop when evaluating controllers optimized in simulation with real robots. The goal of this section is to determine whether simulations can still be used to reduce the required real robot evaluation time when optimizing controllers for real robots.

Using the parameters from Table II and $N_{rob} = 4$, we did 30 optimization iterations in simulation, transferred the pool of candidate solutions to real robots, and did another 30 iterations. We compared this hybrid approach to 60 iterations ran exclusively on real robots (Figure 12).

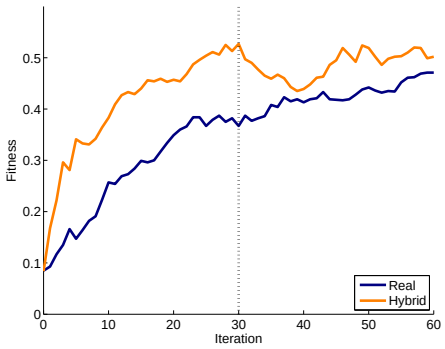


Fig. 12 Average swarm fitness per iteration for PSO with four real robots and hybrid PSO. For hybrid PSO the pool of candidate solutions was transferred from simulation to real robots at iteration 30 (marked with a vertical dashed line)

.

In the hybrid approach, for around 10 iterations after the switch from simulation to reality at iteration 30, the swarm fitness decreases as previous bests are re-evaluated and lower real performances are averaged with the previous simulated ones. After this initial decrease, the performance starts increasing again as new solutions that perform better in reality are found and shared through the swarm.

In the optimization run using real robots exclusively, the swarm fitness increases constantly but at a much lower rate. The hybrid approach reaches a slightly higher final performance while requiring half the real robot evaluation time (2h18min vs 4h36min). The simulation time is negligible when compared to the real-robot time (the simulation speed-up factor is approximately 40x). Therefore, even though simulated controllers do not perform as expected when transferred to real robots, a hybrid simulate-and-transfer approach coupled with a noise-resistant algorithm that re-evaluates performances at each iteration may help to reduce the required experimental time, as well as the wear and tear of the real platforms. Furthermore, both approaches converged to the same straight line strategy discussed in section 4.1.

5 Conclusion

We analyzed the effect of the four PSO algorithmic parameters that determine total evaluation time for a case study of multi-robot limited time learning of an obstacle avoidance behavior. Each parameter was varied independently, and based on the resulting fitness we proposed guidelines to choose these parameters for the case of multi-robot distributed learning. To add more generality to our guidelines, we ran analogous tests on benchmark functions traditionally used for numerical optimization problems.

For the population size parameter, we suggested to use at least the dimension of the search space D, and to use the number of robots if it is greater than D to take advantage of parallel evaluations and increased robustness. We proposed using the robot speed and environment size as guidelines to choose an evaluation span that guarantees at least one interaction of the robot with other components of the environment relevant to the task at hand. Due to the inherent uncertainty in controller evaluations when using more than one robot, we showed that a re-evaluation scheme based on actual re-evaluation and aggregation with former performances has a positive impact in multi-robot learning scenarios. The last parameter, the number of iterations, can be adjusted to fit the total evaluation time available.

By applying our guidelines, we were able to reduce the total adaptation time to an amount which can be easily completed without fully depleting the batteries of the individual robots. This resulted in a maximum quantitative performance drop of 17% but with no observable difference in the qualitative behaviors of the solutions.

We then compared the performance of simulated and real robots. Using trajectory analysis, we studied the best solution strategies and showed that both simulation and real-robot optimization converged to the same strategy of going back and forth between walls in both the single and multi-robot case. We also found that the differences in performance between simulation and reality are more pronounced in a

multi-robot setting than in a single-robot one. In addition, except for the simpler single robot simulation, the fitness distributions were non-Gaussian and multi-modal. Therefore, the additive Gaussian noise model usually employed in studies of optimization algorithms under the presence of noise may not be well suited for real multi-robot implementations.

Finally, we showed that in spite of the differences in performance between simulation and reality, simulation can still be employed to further reduce the required real-robot experimental time by applying a hybrid approach in which the pool of candidate solutions is transferred to real robots for final refinement after the initial simulation stages. We compared the hybrid approach with a pure real-robot optimization and showed that it achieved a slightly higher performance while requiring half the real-robot evaluation time.

Our next step is to modify the algorithm to be able to add and remove robots dynamically and in real time. For this purpose, we intend to study the effect of asynchronous updates and dynamic neighborhood topologies on multi-robot learning. We also plan to apply our distributed algorithms to other controller architectures and robotic tasks. Lastly, we are interested in exploring PSO variations and other population-based algorithms that can be applied to limited-time distributed learning. Even though in this paper we employed the PSO algorithm, we believe the evaluation time and simulation transfer issues discussed are relevant to population-based multi-robot learning in general and not limited to PSO. Our final goal is to devise a set of guidelines for fast, robust adaptation of high-performing robotic controllers for a given mission.

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

References

1. J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz, "Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization," 2013.
2. 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.
3. R. a. Watson, S. G. Ficici, and J. B. Pollack, "Embodied Evolution: Distributing an evolutionary algorithm in a population of robots," *Robotics and Autonomous Systems*, vol. 39, pp. 1–18, Apr. 2002.
4. J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, pp. 1942 – 1948 vol.4, 1995.
5. R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *Journal of Artificial Evolution and Applications*, vol. 2008, no. 2, pp. 1–10, 2008.
6. S. B. Akat and V. Gazi, "Decentralized asynchronous particle swarm optimization," in *IEEE Swarm Intelligence Symposium*, DOI: 10.1109/SIS.2008.4668304, Sept. 2008.
7. J. Rada-Vilela, M. Zhang, and W. Seah, "Random Asynchronous PSO," *The 5th International Conference on Automation, Robotics and Applications*, pp. 220–225, Dec. 2011.
8. 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*, pp. 4175–4180, 2010.
9. L. Marques, U. Nunes, and A. T. Almeida, "Particle swarm-based olfactory guided search," *Autonomous Robots*, vol. 20, pp. 277–287, May 2006.
10. J. Hereford and M. Siebold, "Using the particle swarm optimization algorithm for robotic search applications," in *IEEE Swarm Intelligence Symposium*, pp. 53–59, 2007.
11. Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain EnvironmentsA Survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 303–317, June 2005.
12. D. Arnold and H.-G. Beyer, "A general noise model and its effects on evolution strategy performance," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 380–391, Aug. 2006.
13. K. E. Parsopoulos and M. N. Vrahatis, "Particle Swarm Optimizer in Noisy and Continuously Changing Environments," in *Artificial Intelligence and Soft Computing* (M. H. Hamza, ed.), pp. 289–294, IASTED/ACTA Press, 2001.
14. H. Pan, L. Wang, and B. Liu, "Particle swarm optimization for function optimization in noisy environment," *Applied Mathematics and Computation*, vol. 181, pp. 908–919, Oct. 2006.

15. J. Pugh, Y. Zhang, and A. Martinoli, "Particle swarm optimization for unsupervised robotic learning," in *IEEE Swarm Intelligence Symposium*, pp. 92–99, 2005.
16. J. Pugh and A. Martinoli, "Distributed scalable multi-robot learning using particle swarm optimization," *Swarm Intelligence*, vol. 3, pp. 203–222, May 2009.
17. E. Di Mario, Z. Talebpour, and A. Martinoli, "A Comparison of PSO and Reinforcement Learning for Multi-Robot Obstacle Avoidance," in *IEEE Conference on Evolutionary Computation*, vol. 1, pp. 149–156, June 2013.
18. K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.
19. O. Michel, "Webots: Professional Mobile Robot Simulation," *Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.
20. T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "Swistrack-a flexible open source tracking software for multi-agent systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4004–4010, 2008.
21. E. Di Mario, G. Mermoud, M. Mastrangeli, and A. Martinoli, "A trajectory-based calibration method for stochastic motion models," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4341–4347, Sept. 2011.
22. D. Bratton and J. Kennedy, "Defining a Standard for Particle Swarm Optimization," in *IEEE Swarm Intelligence Symposium*, pp. 120–127, Apr. 2007.