Distributed Particle Swarm Optimization - Particle Allocation and Neighborhood Topologies for the Learning of Cooperative Robotic Behaviors

Iñaki Navarro Ezequiel Di Mario Alcherio Martinoli

***Abstract*— In this article we address the automatic synthesis of controllers for the coordinated movement of multiple mobile robots, as a canonical example of cooperative robotic behavior. We use five distributed noise-resistant variations of Particle Swarm Optimization (PSO) to learn in simulation a set of 50 weights of an artificial neural network. They differ on the way the particles are allocated and evaluated on the robots, and on how the PSO neighborhood is implemented. In addition, we use a centralized approach that allows for benchmarking with the distributed versions. Regardless of the learning approach, each robot measures locally and individually the performance of the group using exclusively on-board resources. Results show that four of the distributed variations obtain similar fitnesses as the centralized version, and are always able to learn. The other distributed variation fails to properly learn on some of the runs, and results in lower fitness when it succeeds. We test systematically the controllers learned in simulation in real robot experiments.**

I. Introduction

This article tackles the high-dimensional synthesis and optimization of controllers for cooperative behaviors carried out by resource-constrained robots. Evaluative machine-learning techniques are an alternative to human design that may allow to fully exploit the platforms' limited sensing capabilities, cope with discontinuities and nonlinearities, as well as deal with noise in the performance evaluations [1]–[5].

As in our previous works [6], [7], the cooperative task chosen is a loosely-coordinated collective movement or flocking [8]–[11], in which a group of robots move together. Some researchers have shown that it is possible to use learning techniques to generate cooperative behaviors [2], [3], [12], [13]. Mataric [12] and Parker [13] addressed the topic of learning in multi-robot teams using a small number of parameters per robot, as opposed to the large search space addressed in this paper.

The aim of this paper is to distribute the learning of a large parameter space by testing several candidate solutions in parallel on the available robotic resources. Such an approach allows the distributed robotic system to increase its robustness to failure of individual robots and speed up the overall learning process [14].

In our previous attempt in this direction [7], the distributed learning of cooperative tasks did not match the performance of the centralized approach in terms of both average and standard deviation between learning runs. In this article, we explore five different distributed variations of Particle Swarm Optimization (PSO) [15], in order to make the distributed version competitive in the learning of cooperative tasks. These variations explore different ways of allocating the candidate solutions among the robots and sharing information among particles.

Moreover, as in the previous attempt [7], we compare the performances of our distributed versions to a centralized implementation. In this context, it is worth noting that the distributed learning approaches result in heterogeneous controllers (robots run different controllers), while the centralized implementation synthesizes homogeneous controllers (every robot runs the same controller).

We have designed a local performance metric that can be evaluated by each robot while leading to the desired cooperative behavior. We augmented the local performance metric from [6], considering only collision avoidance and attraction to neighboring flockmates, to enforce the three Reynolds' flocking rules [16] by adding a factor that reflects the alignment with neighboring flockmates.

It should be mentioned that the task as implemented in this article is harder than those from other works in that the robots are not physically connected to each other [2], they are required not only to aggregate but also move together [3], and there is no environmental template or goal to guide their movement [1]. In the case of [2] and [3] learning has been done only in a centralized manner, using homogeneous controllers and a global performance metric.

Some researchers have used population-based optimization techniques to improve the performance of manually designed flocking controllers, using PSO [17], Genetic Algorithms [18], or Evolutionary Strategies [19]. In these works the controllers are homogeneous or only the controller of a single robot is optimized, so the allocation problem studied in our paper is not addressed by them. Lee and Myung [20] used a distributed version of PSO for online optimization of the individual trajectories of robots running model predictive control for flocking by using parallel instances of PSO for each robot, which is similar to one of the allocation variations explored in our paper. Our approach in this article differs with these previous works in that our behaviors are generated by a highly plastic artificial neural network and not by a specific flocking controller.

The rest of this article is organized as follows. In Section II we describe the robotic platform, the control architecture, the fitness metric, and the learning algorithms proposed. In Section III we present the different experiments performed

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.

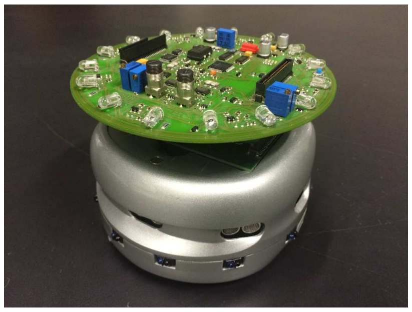

Fig. 1. A Khepera III robot expanded with the relative positioning extension board, used in the flocking experiments.

and discuss the results obtained both in simulation and with real robots. Lastly, Section IV draws the conclusions of this work and formulates some prospective investigation topics.

II. Methodology

Different variations of PSO are used in this article in order to learn flocking behaviors in a distributed manner. The learning problem for PSO is to choose a set of parameters of an underlying robotic controller such that a given fitness metric is maximized. The learning process is performed completely in simulation using four robots, while the learned solutions are tested both in simulation and using real robots.

A. Experimental Platform

The experimental platform used is the Khepera III mobile robot, a differential wheeled vehicle with a diameter of 12 cm. Its sensing capabilities are augmented with a relative positioning system [21], which calculates range and bearing to nearby robots based on the strength of infrared signals. The system also communicates the ID of the robot, allowing to estimate also the heading of neighboring robots by exchanging the bearings between a pair of robots. In our experiments, this communication is implemented using the IEEE 802.11 wireless standard and UDP messages. The Khepera III has two wheel encoders, which are used to estimate the trajectory followed by the robots for the fitness calculations. A Khepera III equipped with the relative positioning system can be seen in Fig. 1.

Simulations are performed in Webots [22], a high-fidelity 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 simulators, faithfully reproducing intra-robot modules (e.g., individual sensors and actuators). The simulator has a built-in relative positioning system that gives information about the distance and direction to neighboring robots within line-of-sight, mimicking the one used in the real robots.

B. Control Architecture

The controller used is an artificial neural network with nine inputs, a hidden layer of four units with sigmoidal activation functions, and two output units also with sigmoidal activation (see Fig. 2). The sigmoidal activation function

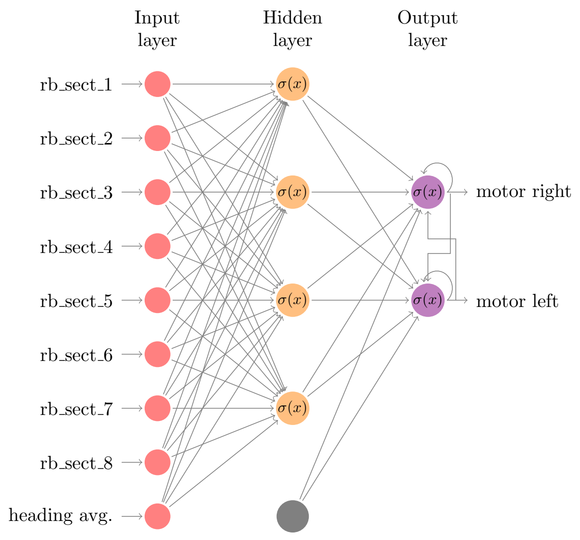

Fig. 2. Diagram of the neural network controller. In red are the inputs, yellow the hidden layer with sigmoidal outputs, in blue the sigmoidal outputs which control the motor speed, and in gray the bias input. $\sigma(x)$ indicates sigmoidal output.

used is $\sigma(x) = \frac{2}{1+e^{-x}} - 1$. The output neurons also have as input a connection from a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output. The controller uses only local, on-board measurements. Its inputs are the range and bearing measurements and the heading average among the robots, while the outputs determine the two wheel speeds. The total number of weights of the controller to be optimized by the PSO algorithm is 50 (36 from the inputs to the hidden layer, eight from the hidden layer to the output neurons, four from the recurrent connections, and two from the bias speeds).

The eight range and bearing inputs (rb_sect_k) are obtained by dividing the bearing into eight sectors of $\pi/4\,rad$, and calculating the activation of each sector by taking the minimum range value measured in that sector and dividing it by the maximum possible range, which is 3.3 meters. The first sector covers the bearing $[0, \pi/4)\,rad$. The ninth input corresponds to the average of the headings among all the neighboring robots, in the robot's own coordinate system and normalized to the interval [-1,1]. The use of a single averaged input instead of one input per robot allows the controller to generalize to any number of robots.

C. Fitness Function

The fitness function, calculated individually and locally by each robot using exclusively on-board resources, estimates the performance of the group of flocking robots. It is composed of three factors: movement, alignment, and compactness. The fitness function rewards robots that move as far as possible from their initial positions, align their headings, and stay close to each other without colliding. All the factors are normalized to the interval $[0, 1]$.

The movement factor (f_1) is the normalized distance between the initial and the final positions of the center of mass of the group of robots (see Eq. 1). The normalization constant (D_{max}) is the maximum distance that a robot can travel in one evaluation, i.e., the robot's maximum speed multiplied by the evaluation time. This factor is estimated using the odometry of the robot, by means of the wheel encoders, and the relative position to neighboring robots. If a neighboring robot position can not be estimated (due to occlusions or limited range of the relative positioning system), the last absolute position where the robot was seen is used as final position.

$$f_1 = \frac{|\vec{x_c}(t_f) - \vec{x_c}(t_0)|}{D_{max}} \tag{1}$$

The alignment factor (f_2) quantifies the heading difference between two robots (H_{diff}) averaged between the robot and each of its neighbors, and during the evaluation time. It has a maximum value of 1 when all the robots are aligned and tends to 0 when robots are not aligned. It is defined as:

$$f_2 = 1 - \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \left(\frac{1}{N_{neigh}} \sum_{j=1}^{N_{neigh}} abs(H_{diff_{j,k}})/\pi\right) \tag{2}$$

where N_{eval} is the number of time steps in the evaluation period, N_{neigh} is number of neighboring robots and $H_{diff_{j,k}}$ is the difference of heading to neighboring robot j at time step k. The heading difference is calculated using the relative positioning system and communication. These measurements might be affected by occlusions and range limitations. If for a time step no neighbor is seen then H_{diff} is set to π for that time instant, representing alignment worst case.

The compactness factor (f_3) estimates the desired robot to robot distance within the group of robots. It is calculated as the average over the evaluation time and over each neighboring robot of the inter-robot fitness:

$$f_3 = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} \left(\frac{1}{N_{neigh}} \sum_{j=1}^{N_{neigh}} fit_{inter_{j,k}}\right) \tag{3}$$

where N_{eval} is the number of time steps in the evaluation period, N_{neigh} is number of neighbors, and $fit_{inter_{j,k}}$ is the inter-robot fitness between the robot and its neighbor j at time step k. We define the inter-robot fitness between two robots as a function of the distance between them, as shown in Fig. 3. The fitness is maximum at the desired inter-robot distance of $0.4m$, and it is zero when the robots are closer than $0.2m$ (slightly larger than the robots' diameter) or further apart than $0.6m$. It rewards that robots stay close to each other without colliding, implementing two of the Reynolds' rules. The inter-robot distance is measured using the relative positioning system, so it is affected by occlusions. If for a time step a neighbor is not detected then the inter-robot fitness to that robot is set to 0 for that time instant.

The fitness function is obtained by aggregating the three factors using the generalized aggregation function described by Zhang et al. [23]:

$$F = \left(\frac{\omega_1 f_1^s + \omega_2 f_2^s + \omega_3 f_3^s}{\omega_1 + \omega_2 + \omega_3}\right)^{\frac{1}{s}} \tag{4}$$

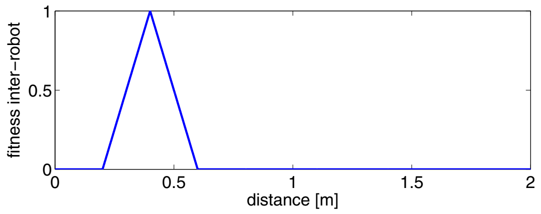

Fig. 3. Inter-robot fitness as a function of the distance between two robots.

where f_i are the fitness factors, ω_i their corresponding aggregation weights, and s is the degree of compensation or trade-off strategy (higher s means that a high value for a certain factor can compensate for lower values in the others). For all experiments in this article we set the degree of compensation s tending to zero, simplifying Eq. 4 to:

$$F = \lim_{s \to 0} \left(\frac{\omega_1 f_1^s + \omega_2 f_2^s + \omega_3 f_3^s}{\omega_1 + \omega_2 + \omega_3}\right)^{\frac{1}{s}} = \left(f_1^{\omega_1} f_2^{\omega_2} f_3^{\omega_3}\right)^{\frac{1}{\omega_1 + \omega_2 + \omega_3}} \tag{5}$$

Since the three factors (f_i) are in the interval $[0,1]$, the fitness function F will also be in the same range. The aggregation weights used in this article are: $\omega_1 = 0.4$, $\omega_2 = 0.5$, and $\omega_3 = 0.1$.

In our previous work [24], we showed that the fitness evaluations for learning a simpler robotic task had a large standard deviation, and that performing re-evaluations was an effective way of dealing with this challenge in the learning. Given the more complex behavior to be learned in this article and the difficulties encountered while doing so, we decided to perform multiple internal evaluations of the fitness and average them in order to make the learning more robust. Concretely, each candidate solution is evaluated $N_{eval} = 4$ times during $45s$ and its performance averaged before consideration by the noise-resistant PSO algorithms:

$$F' = \frac{1}{N_{eval}} \sum_{i=1}^{N_{eval}} F_i \tag{6}$$

D. Learning Algorithm

The different PSO algorithms studied in this paper are all based on the noise-resistant version introduced by Pugh et al. [25]. This baseline algorithm works by re-evaluating personal best positions and aggregating them with the previous evaluations, by performing a regular average at each iteration of the algorithm. The pseudocode for the algorithm is presented in Fig. 4.

Each particle position represents a set of weights of the neural controller, with each weight corresponding to one dimension. The optimization process for PSO becomes choosing the set of weights from the controller such that the fitness function F as defined in Eq. 5 is maximized. Particle evaluations consist of a group of robots moving for a fixed time. As defined in Eq. 7, 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

```
1:  Initialize particles
2:  for $N_i$ iterations do
3:      for $N_p$ 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. 4. Common noise-resistant PSO algorithm.

TABLE I

PSO PARAMETER VALUES

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	52 / 13
Iterations N_i	400
Evaluation span t_e	4x45 s
Re-evaluations N_{re}	1
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Dimension D	50
Inertia w	0.8
V_{max}	20

w_p, and a randomized attraction to the neighborhood's best $x^*_{i',j}$ weighted by w_n. $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$\begin{aligned} v_{i,j} &:= w_I \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) \\ &\quad + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \qquad (7) \\ x_{i,j} &:= x_{i,j} + v_{i,j} \qquad (8) \end{aligned}$$

At the end of each iteration each particle shares its personal best position with its neighborhood. By default, this neighborhood is implemented as a ring topology with one neighbor on each side. Here, neighborhood refers to the PSO algorithm, not to the physical location of robots typically influencing relative positioning and communication. We will explore different variations of this topology, and describe them in Section II-E. 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. The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented in our previous work [26] and are shown in Table I. Since the dimension of the search space is 50 and four robots are used, we round up to 52 particles in order to have a multiple of the number of robots used which is four.

E. Algorithmic Variations

In this article we explore six variations of the noise-resistant PSO algorithm. They differ on the way the particles are allocated and evaluated on the robots, and on how the neighborhood is implemented. In distributed PSO we need to evaluate N_p particles ($\{P_1, P_2, \ldots, P_{N_p}\}$) representing N_p candidate solutions, by testing them on N_{rob} robots ($\{R_1, R_2, \ldots, R_{N_{rob}}\}$). At each iteration we need to evaluate both the current position ($\vec{x}_i$) of each particle P_i, as well as the historical personal best position ($\vec{x}^*_i$). In our case the performance of each position is obtained by carrying out N_{eval} single evaluations (F_i). Our problem here is how to allocate in an appropriate manner the N_p particles into the N_{rob} robots for every single evaluation, so that the distributed PSO obtains good solutions.

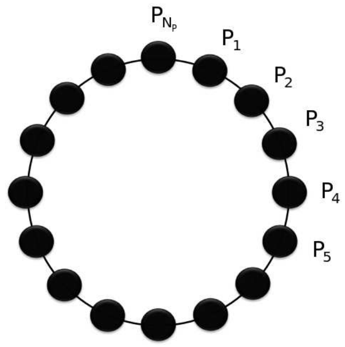

Fig. 5. PSO ring topology indicating the particle number. The neighborhood of P_i is $\{P_{i-1}, P_i, P_{i+1}\}$.

At the end of each iteration the personal best solution ($\vec{x}^*_i$) of each particle P_i is shared with other particles. The sharing topology studied in this article is an undirected ring graph with a neighborhood size of 3, so each particle P_i shares information with itself and the two other particles. For instance, in the most simple case, this ring is static (see Fig. 5), so on every iteration the neighborhood of P_i is the own particle and the two adjacent particles (P_{i-1}, P_i, and P_{i+1}).

The first studied variation of the particle allocation and neighborhood topology is a centralized version, which is used as a reference, while the remaining five distributed implementations are the focus of this paper. In the centralized version each particle is evaluated at the same time in all the robots. The distributed versions allocate a different candidate solution to each robot, and evaluate the fitness independently and individually on each robot. They allow to speed up the learning by a factor equal to the number of robots available in the system. These variations of the PSO algorithm are described in the following paragraphs.

- *central*. Each particle is evaluated concurrently on every robot. Since each robot has a different local evaluation of the group performance, the fitness reported to the algorithm is calculated as the average of the individual fitnesses. The neighborhood ring topology is static, i.e., each particle has always the same two neighbors. The number of particles used is always 13, one fourth of the 52 particles in the distributed variations. It allows for fair comparison between the algorithms, equalizing the total evaluation time between centralized and distributed versions.
- *alloc-seq*. It distributes sequentially a different candidate

solution to each robot: P_1 assigned to R_1, P_2 to R_2, P_3 to R_3, P_4 to R_4, P_5 to R_1, P_6 to R_2, ..., and P_{N_p} to R_4. This implies that each particle is evaluated always on the same robot. In addition, each particle is always evaluated together with the same other three particles, in the same group. Particles $\{P_1,\ldots,P_4\}$ are evaluated together, and so are $\{P_5,\ldots,P_8\}$, ..., and $\{P_{N_p-3},\ldots,P_{N_p}\}$. In the static ring topology used, the neighborhood of P_i is P_{i+1} and P_{i-1}. As a result, in each group of four particles that are evaluated concurrently on the four robots, two of them have a neighborhood entirely within the same evaluation group, while each of the other two has one neighbor from another evaluation group. It corresponds to the algorithmic variation that we used in our previous work [7].

- *alloc-robot*. It allocates the particles assigning them in blocks to each robot: $\{P_1,\ldots,P_{N_p/4}\}$ to R_1, $\{P_{N_p/4+1},\ldots,P_{2N_p/4}\}$ to R_2, ..., and $\{P_{3N_p/4+1},\ldots,P_{N_p}\}$ to R_4. The concurrent evaluation has to take place on each robot at the same time, so P_i is evaluated always together with $P_{i+N_p/4}$, $P_{i+2N_p/4}$, and $P_{i+3N_p/4}$. The neighborhood used is the same static ring topology as in previous cases. Given the particle allocation, all the particles that are evaluated on a robot R_i except two have as neighborhood particles evaluated also in robot R_i. The two remaining particles have one neighbor evaluated on the same robot and a second in a different one. This promotes better specialization and adaptation in specific robots, since there is more flow of intra-robot information within the PSO algorithm. As opposed to *alloc-seq*, there is no neighborhood with particles evaluated at the same time.
- *alloc-rand*. Particles are allocated in a random order and on random robots at each single evaluation (F_i). This allocation generates N_p/N_{rob} parallel evaluations of different combinations of N_{rob} particles, obtaining one evaluation for each particle in each single evaluation. At each single evaluation (F_i) a new random allocation is generated. Therefore, each particle is randomly tested together with any other three particles, and in any of the robots. This allows to discriminate good from bad candidate solutions at the same time they are tested in parallel, since each particle is eventually evaluated with many others, and its fitness averaged. The neighborhood used is also the static ring topology, so besides the random allocation each particle always shares it personal best with the same two particles.
- *neigh-rand*. The particles are allocated sequentially in the same manner as in *alloc-seq*. The difference here is the neighborhood, which has a dynamic randomly generated ring topology. At each iteration a new random ring is defined, and each particle shares its personal best with two other particles. As in *alloc-seq*, each particle is always evaluated with the same three particles and on the same robot (allowing potential adaptation to the hardware), but it shares information with any particle.
- *parallel*. It consists of N_{rob} parallel PSO implementations, each of them allocating particles on a different robot. Each evaluation is performed in parallel on the four robots working together. It is an extreme case of *alloc-robot*, in which all the particles evaluated on a robot share information only with other particles evaluated on the same robot. As in *alloc-robot*, each particle is always evaluated together with the same three particles and on the same robot. The total number of particles used by the N_{rob} parallel PSO implementations is N_p, N_p/N_{rob} per parallel PSO.

In *central* there is homogeneity in the controllers, all robots run the same candidate solution when they are evaluated. A posteriori tests are then performed running the best solution on every robot. On the other hand, in the distributed versions, during the learning process the controllers are heterogeneous, each robot runs a different set of weights. In *alloc-seq*, *alloc-robot*, *neigh-rand*, and *parallel* each particle is always evaluated on the same robot and together with the same group of particles. Therefore, when testing the best controller resulting from these learning approaches, we find the particle with the best fitness and test it together with the other three particles used during the learning evaluations. However, this is not possible for *alloc-rand*, since particles are always evaluated in different combinations. Thus, we propose two testing approaches: homogeneous and heterogeneous. In the first, the best solution is replicated on the four robots. In the second, the four best ranked solutions are assigned to the four robots.

In *alloc-seq*, *alloc-robot*, *neigh-rand* and *parallel*, the fitness tends to evaluate all the particles in the group of robots with a very similar performance, although the controllers could be very different. This problem is partially solved in the *alloc-rand* variation.

III. Experiments and Results

The learning process is performed completely in simulation with the six algorithmic variants described in Section II-E. Since PSO is a stochastic optimization method, we perform 20 optimization runs for each of these learning schemes.

Each evaluation during the learning process has a duration of $45\,s$ and takes place in an unbounded arena. Four robots are placed forming a square of side length equal to two robot diameters with random orientations. The performance metric is calculated by the robots using only their internal measurements (simulated range and bearing and wheel encoders, both with added noise).

Figure 6 shows the progress of the learning for the six configurations discussed in this article. The centralized approach is the fastest to learn, in the sense that it achieves a high fitness in fewer iterations than the other configurations. The distributed approaches could benefit from more iterations to further improve their performance since the slope of the curve is not completely flat at the 400th iteration.

The first four configurations, namely *central*, *alloc-seq*, *alloc-robot*, and *alloc-rand*, show a low standard deviation between learning runs. *neigh-random* has a larger standard

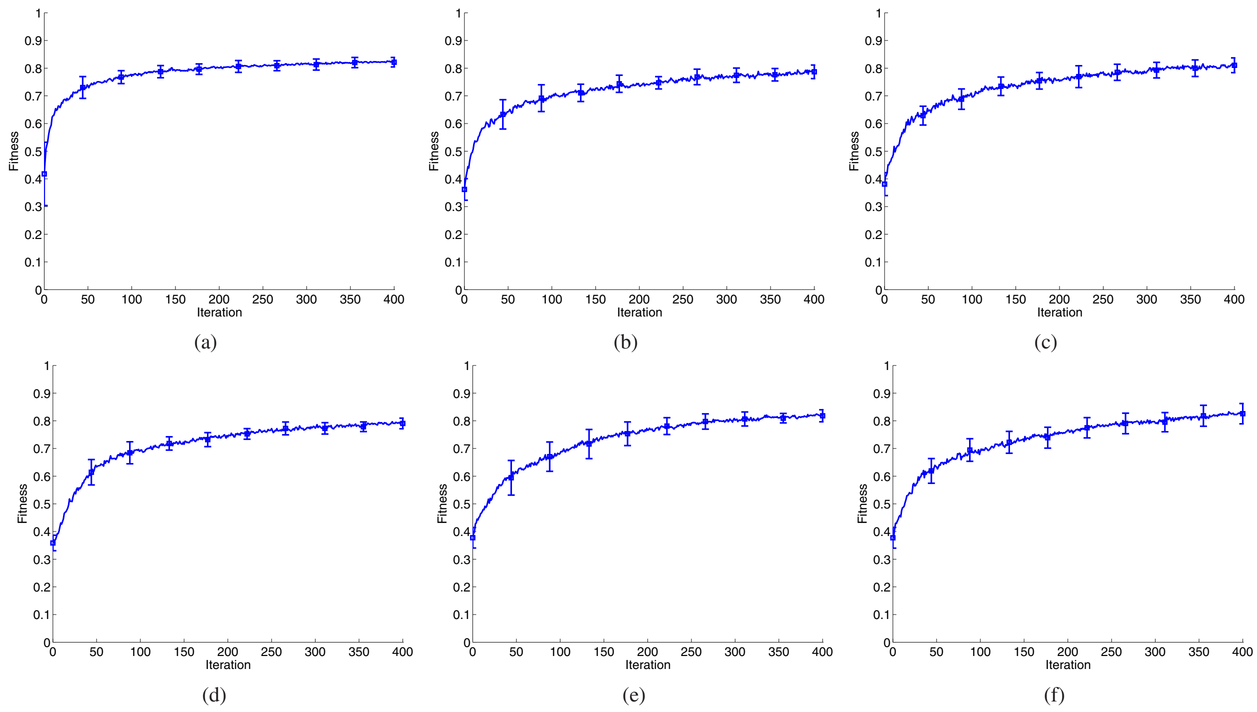

Fig. 6. Progress of the learning in simulation for the six configurations: (a) *central*, (b) *alloc-seq*, (c) *alloc-robot*, (d) *alloc-rand*, (e) *neigh-rand*, (f) *parallel*. The blue curve shows the average of the 20 runs, and the error bars represent the standard deviation.

deviation in the intial iterations but it decreases as the learning progresses. On the other hand, *parallel* has a fairly constant standard deviation during the whole process.

After the learning process is finished, the fitness of the best solution at the last iteration from each of the 20 learning runs is evaluated systematically in simulation, running 100 experiments of $60s$ for each solution. Figure 7 presents box-plots of the performances of these evaluations in simulation. Figures 7b to 7h show the result for each of the 20 individual runs, while Fig. 7a aggregates the results of the 20 runs for each configuration.

It is clear from Fig. 7a and 7b that the centralized approach achieves the highest performance and the lowest variance. Therefore, it serves as a baseline to compare all the distributed approaches.

The sequential approach (Fig. 7c) performs significantly worse than all the other distributed approaches on average, has a higher variance, and presents a small number of runs that perform very poorly. In this approach, the solution sharing is done mostly among particles that are evaluated at the same time in the same group. We hypothesise that this may cause an early convergence within each evaluation group to a solution that might not necessarily be the best one. In other words, because of the limited inter-evaluation group sharing there might be less exploration which may cause stagnation. In the remaining four distributed approaches, all particles share solutions with other particles that they are not evaluated with, and they achieve a higher fitness and smaller variance than the sequential approach.

In particular, the parallel learning approach (Fig. 7h) represents the extreme case where there is absolutely no sharing of solutions within the same evaluation group, i.e., each robot is running a completely independent instance of the learning algorithm, and it achieves the highest median performance of all the distributed configurations. The lack of sharing suggests that this approach could be used to do learning on a subset of robots while the others run pre-learned or manual controllers, or even employ different learning algorithms on each robot.

As mentioned in Section II-E, in the case of random allocation there is no fixed evaluation group, so during testing we used two ways to select the best group: homogeneous and heterogeneous. It is interesting to note that the homogeneous approach *alloc-rand-hom* where the best solution is replicated in the four robots (Fig. 7f) performs significantly worse than the heterogeneous *alloc-rand-het* which uses the top four best-ranked solutions (Fig. 7e). This effect may be caused by specialization: the best solution could, for example, learn to move straight while the others follow. Therefore, it would obtain a higher fitness when tested with the three followers, but a poor one when the leader is replicated on the four robots.

In order to validate the results obtained in simulation, we selected the controller with the highest median for each learning approach and tested it on real robots. We discarded *alloc-rand-hom* due to its low performance compared to *alloc-rand-het*. We run 20 experiments for each solution. The initial positions and number of robots were the same as used for learning in simulation, but the evaluation time was reduced to $15s$ due to space constraints in the laboratory. The fitness function was computed on each robot using only its on-board resources. In order to do fair comparisons due

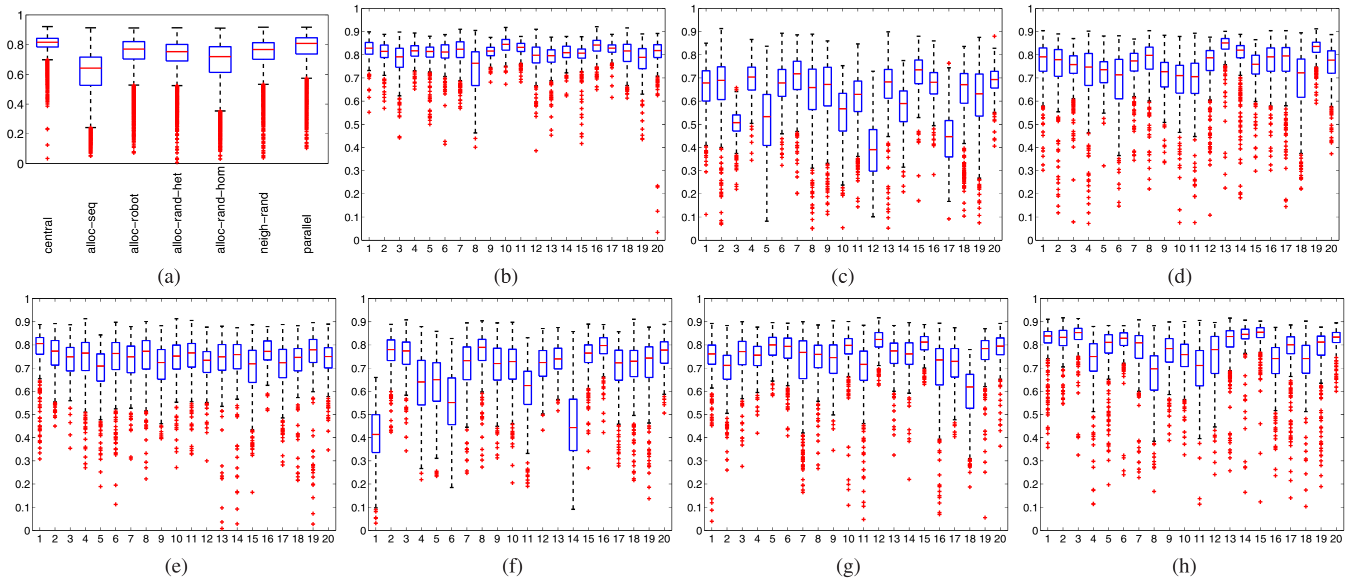

Fig. 7. (a) Performance in simulation for the different configurations aggregating results from 20 runs. (b) - (h) Performance in simulation for the 20 individual runs of (b) *central*, (c) *alloc-seq*, (d) *alloc-robot*, (e) *alloc-rand-het*, (f) *alloc-rand-hom*,(g) *neigh-rand*, (h) *parallel*. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers for 400 evaluations (100 experiments evaluated by four robots) of each controller.

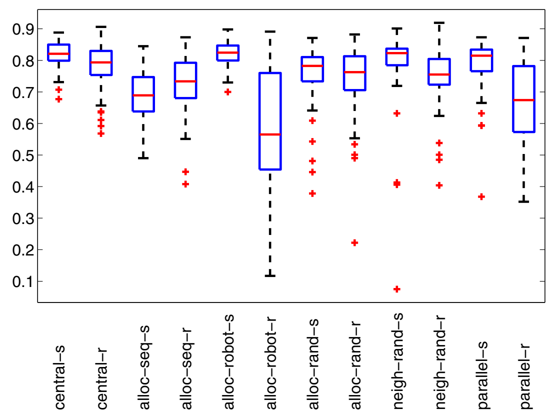

Fig. 8. Evaluation in simulation and with real robots for experiments of 15*s* for the selected controllers. The *-s* suffix refers to experiments conducted in simulation, while *-r* refers to those with real robots.

to the difference in the evaluation time, we also tested the selected controllers in simulation for 15*s*. The results are shown in Fig. 8.

The performances for 15*s* runs are generally lower than for 60*s* runs since the initial stage of aggregation and alignment of the robots represents a larger fraction of the total time. Four controllers, namely *central*, *alloc-seq*, *alloc-rand-het*, and *alloc-rand*, perform very similarly in simulation and in reality. There is a significant difference in performance for *alloc-robot* and *parallel* due to inaccuracies in the model of the robot in simulation. In particular, we observed that during real robot experiments with these two controllers the robots' wifi cards got stuck with one another, while in simulation this card is not modeled accurately enough, and the robots do not get stuck.

An example of a relevant trajectory for each learning variation is shown in Fig. 9, taken from the controllers with highest median. The trajectories observed in the selected controllers learned with *central*, *alloc-robot*, *neigh-rand* and *parallel* are in general very straight and robots maintain the group cohesiveness. The selected controller from *alloc-seq* fails to maintain cohesiveness in some evaluations, resulting sometimes in a split into two subgroups, which might later regroup. The trajectories from the selected controller using *alloc-rand-het* are not always completely straight, resulting sometimes in curvilinear paths. These trajectories reflect that the presented local fitness measured individually on each robot allows generating the desired cooperative behavior.

IV. Conclusion and Future Work

In this paper we have explored five distributed variations of noise resistant PSO with the aim of improving the learning of cooperative behaviors in a distributed manner. The most intuitive approach, which consists of sequentially assigning particles to each robot and evaluating them, cannot always learn correctly, and has the lowest median fitness across all the variations. The remaining four variations are always able to learn and have similar fitness to the centralized baseline approach. In these four variations, particles share solutions with particles they are not evaluated with, while the sequential variation mostly shares solutions with particles evaluated together. In all variations, we employed a local metric evaluated individually on each robot using on-board resources.

As continuation of this work, we would like to study further variations in the neighborhood topology, for instance by changing the number of neighbors. We also want to exploit and study the specialization capabilities by modifying the role of certain robots or changing their sensing capabilities.

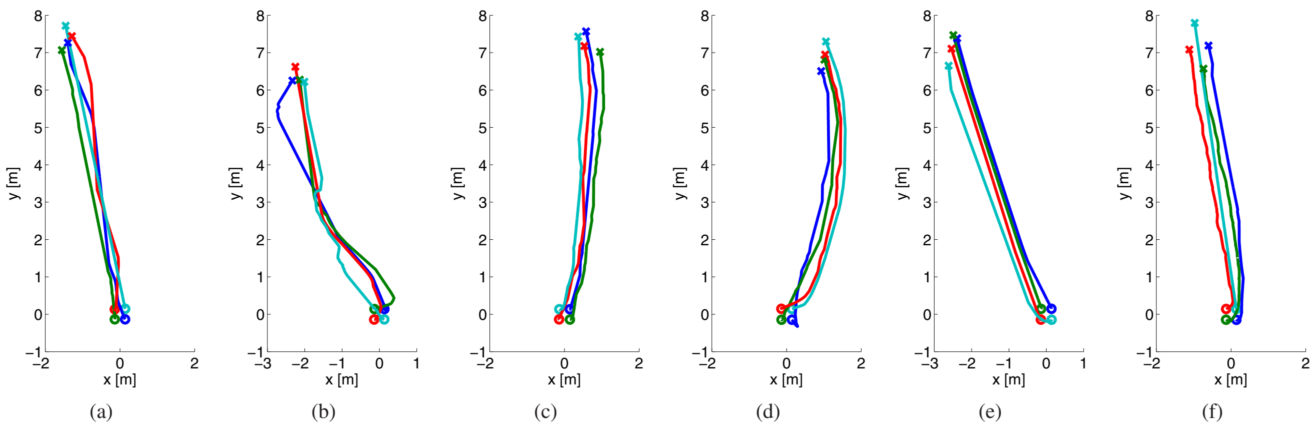

Fig. 9. Example of trajectories of robot flocking in simulation during 60s for selected controllers from: (a) *central*, (b) *alloc-seq*, (c) *alloc-robot*, (d) *alloc-rand-het*, (e) *neigh-rand*, (f) *parallel*. The initial positions are marked with a circle, while the final positions are marked with a cross.

In addition, we would like to test the learned controllers on a larger set of robots in order to study experimentally the scalability, and on a reduced set of robots (two or three) to test the robustness of the controllers.

Finally, we want to implement more advanced ways of dealing with noisy fitness functions, such as PSO based on Optimal Computing Budget Allocation (OCBA) [27], adapting them to the distributed learning of cooperative behaviors.

References

[1] 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.

[2] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi, "Self-organized coordinated motion in groups of physically connected robots." *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, vol. 37, no. 1, pp. 224–39, 2007.

[3] M. Gauci, J. Chen, T. Dodd, and R. Groß, "Evolving aggregation behaviors in multi-robot systems with binary sensors," in *Eleventh Int. Symp. on Distributed Autonomous Robotic Systems, Springer Tracts in Advanced Robotics*, 2014, vol. 104, pp. 355–367.

[4] 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.

[5] J. Pugh and A. Martinoli, "Distributed scalable multi-robot learning using particle swarm optimization," *Swarm Intelligence*, vol. 3, no. 3, pp. 203–222, 2009.

[6] E. Di Mario, I. Navarro, and A. Martinoli, "Distributed learning of cooperative robotic behaviors using particle swarm optimization," in *14th International Symposium on Experimental Robotics*, June 2014, to appear in Springer Tracts in Advanced Robotics.

[7] 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.

[8] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[9] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, pp. 401–420, 2006.

[10] G. Antonelli, F. Arrichiello, and S. Chiaverini, "Flocking for multi-robot systems via the null-space-based behavioral control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1409–1414.

[11] I. Navarro and F. Matía, "A framework for collective movement of mobile robots based on distributed decisions," *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 685–697, 2011.

[12] M. Matarić, "Learning in behavior-based multi-robot systems: Policies, models, and other agents," *Cognitive Systems Research*, vol. 2, pp. 81–93, 2001.

[13] L. E. Parker, "L-ALLIANCE : Task-oriented multi-robot learning in behavior-based systems," in *Advanced Robotics, Special Issue on Selected Papers from IROS'96*, 1997, pp. 305–322.

[14] 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.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, 1995, pp. 1942 – 1948.

[16] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[17] S. Etemadi, R. Vatankhah, A. Alasty, G. Vossoughi, and M. Boroushaki, "Leader connectivity management and flocking velocity optimization using the particle swarm optimization method," *Scientia Iranica*, vol. 19, no. 5, pp. 1251 – 1257, 2012.

[18] T. I. Zohdi, "Computational design of swarms," *Int. J. Numer. Meth. Engng.*, vol. 57, pp. 2205–2219, 2003.

[19] H. Celikkanat, "Optimization of self-organized flocking of a robot swarm via evolutionary strategies," in *International Symposium on Computer and Information Sciences*, 2008.

[20] S.-M. Lee and H. Myung, "Particle swarm optimization-based distributed control scheme for flocking robots," in *Robot Intelligence Technology and Applications*, 2013, vol. 208, pp. 517–524.

[21] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast on-board relative positioning module for multi-robot systems," *Special issue on Mechatronics in Multi-Robot Systems, IEEE Trans. on Mechatronics*, vol. 14, no. 2, pp. 151–162, 2009.

[22] O. Michel, "Webots: Professional mobile robot simulation," *Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[23] Y. Zhang, E. Antonsson, and A. Martinoli, "Evolutionary engineering design synthesis of on-board traffic monitoring sensors," *Research in Engineering Design*, vol. 19, no. 2, pp. 113–125, 2008.

[24] 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.

[25] J. Pugh, Y. Zhang, and A. Martinoli, "Particle swarm optimization for unsupervised robotic learning," in *IEEE Swarm Intelligence Symposium*, 2005, pp. 92–99.

[26] E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited time adaptation in autonomous robots," in *Eleventh Int. Symp. on Distributed Autonomous Robotic Systems, Springer Tracts in Advanced Robotics*, 2014, vol. 104, pp. 383–396.

[27] 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. 5911–5917.