2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

The Role of Environmental and Controller Complexity in the Distributed Optimization of Multi-Robot Obstacle Avoidance

Ezequiel Di Mario Iñaki Navarro Alcherio Martinoli

***Abstract*— The ability to move in complex environments is a fundamental requirement for robots to be a part of our daily lives. Increasing the controller complexity may be a desirable choice in order to obtain an improved performance. However, these two aspects may pose a considerable challenge on the optimization of robotic controllers. In this paper, we study the trade-offs between the complexity of reactive controllers and the complexity of the environment in the optimization of multi-robot obstacle avoidance for resource-constrained platforms. The optimization is carried out in simulation using a distributed, noise-resistant implementation of Particle Swarm Optimization, and the resulting controllers are evaluated both in simulation and with real robots. We show that in a simple environment, linear controllers with only two parameters perform similarly to more complex non-linear controllers with up to twenty parameters, even though the latter ones require more evaluation time to be learned. In a more complicated environment, we show that there is an increase in performance when the controllers can differentiate between front and backwards sensors, but increasing further the number of sensors and adding non-linear activation functions provide no further benefit. In both environments, augmenting reactive control laws with simple memory capabilities causes the highest increase in performance. We also show that in the complex environment the performance measurements are noisier, the optimal parameter region is smaller, and more iterations are required for the optimization process to converge.**

I. Introduction

In simple environments, it is usually straightforward for human designers to foresee the different conditions a robot will be exposed to, thus robotic controllers can be designed manually by simplifying the number of parameters or inputs used. However, for more complex environments, the human design of high-performing controllers becomes a challenging task, especially when the design space is constrained by severe limitations of the on-board resources. Machine-learning techniques are an alternative to human design that can automatically synthesize robotic controllers in large search spaces, coping with discontinuities and nonlinearities, and find innovative solutions not foreseen by human designers, especially when a large plasticity is available for control design.

Increasing the complexity of the controller is one possible way to obtain a higher performance. In particular, for artificial neural networks this can be achieved by increasing the number of neurons and sensory inputs, adding memory in the form of recurrence, or adding non-linear operators.

Yet this increased complexity of the controller creates a harder problem for the learning algorithm. A harder problem may mean noisier performance measurements, narrower optimal parameter regions, or slower convergence.

Thus, the goal of this article is to quantify the trade-offs between the complexity of computer-synthesized controllers and their performance in different environments. For this purpose, we use multi-robot obstacle avoidance as a benchmark task.

Obstacle avoidance was used in one of the earliest works of evaluative adaptation with Genetic Algorithms applied to real robots [1], and it has also been employed to test other learning algorithms such as Particle Swarm Optimization (PSO) [2] and Reinforcement Learning [3]. We chose obstacle avoidance as a benchmark task because it can be implemented with different number of robots, requires basic sensors and actuators available in most mobile robots, and the performance metric can be defined to be fully evaluated with on-board resources. Thus, it can serve as a benchmark for testing learning algorithms with real robots in the same way that standard benchmark functions are used in numerical optimization, such as DeJong's test suite [4].

The adaptation technique used is PSO [5], which allows a distributed implementation in each robot, speeding up the optimization process and adding robustness to failure of individual robots.

The remainder of this article is organized as follows. Section II provides some background on PSO, and on the relationship between enviromental and controller complexity in robotic problems. In Section III, we describe the experimental methodology, comprising controllers, environments, and optimization algorithm. Section IV presents the experimental results obtained and discusses the validity of the proposed hypotheses. Finally, Section V concludes the paper.

II. Background

PSO is a relatively new metaheuristic originally introduced by Kennedy and Eberhart [5], which was inspired by the movement of flocks of birds and schools of fish. 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 [6].

PSO is well suited for distributed implementations due to its distinct individual and social components and the use of

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

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

the neighborhood concept. Most of the work on distributed implementation has been focused on benchmark functions running on computational clusters [7], [8]. Implementations with mobile robots are mostly applied to odor source localization [9], [10], and robotic search [11], where the particles' position is usually directly matched to the robots' position in the arena. Thus, the search is conducted in two dimensions and with few or even only one optima, which does not represent a complex optimization problem.

Most of the research on optimization in noisy environments has focused on evolutionary algorithms [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]. Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions and for certain scenarios of limited-time learning in presence of noise [2], [15].

In our previous work [16], we analyzed in simulation how different algorithmic parameters in a distributed implementation of PSO affect the total evaluation time and the resulting fitness. We proposed guidelines aiming at reducing the total evaluation time so that it is feasible to implement the adaptation process within the limits of the robots' energy autonomy. Further, we analyzed how the behavior and performance of the controllers differed based on the environment where learning takes place [17], and showed that no single learning environment was able to generate a behavior general and robust enough to succeed in all testing environments.

Regarding controller complexity, Al-Kazemi and Habib investigated the internal behavior of PSO when the complexity of a problem is increased by adding dimensions to the problem [18]. They used different metrics to conclude that swarm particles behave in a similar way independently of the dimension of the search space. Auerbach and Bongard studied the relationship between environmental and morphological complexity in evolved robots [19], showing that many complex environments lead to the evolution of more complex body forms than those of robots evolved in simple environments.

III. Methodology

In order to analyze the trade-offs between controller complexity and performance in different environments, we perform a set of twelve experiments, involving six controller architectures (described in Subsection III-C) and two environments (Subsection III-D). The fitness function, experimental platform, and optimization algorithm are the same for all experiments. The learning is conducted in simulation, and the best solutions are later tested both in simulation and with real robots.

A. Fitness Function

We use the same metric of performance as [1], which is present in several studies on learning obstacle avoidance (e.g., [2], [20], and our own previous work [16]). The fitness function consists of three factors, all normalized to the interval [0, 1]:

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

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

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

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

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k, $i_{max,k}$ is the normalized proximity sensor activation value of the most active sensor at time step k, and N_{eval} is the number of time steps in the evaluation period. This function rewards robots that move quickly (f_v), turn as little as possible (f_t), and stay away from obstacles (f_i). Each factor is calculated at each time step and then the product is averaged for the total number of time steps in the evaluation period. The averaging effect reduces the impact of the initial conditions, but it also filters abrupt variations, unless they last appreciably in time, e.g., the robot getting stuck is much more heavily penalized than a single instantaneous collision.

B. Experimental Platform

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

Since the response of the Khepera III proximity sensors is not a linear function of the distance to the obstacle, the proximity values are inverted and normalized using measurements of the real robot sensor's response as a function of distance. This inversion and normalization results in a proximity value of 1 when touching an obstacle, and a value of 0 when the distance to the obstacle is equal to or larger than 10 cm.

Simulations are performed in Webots [21], 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).

C. Controllers

Six controllers of incremental complexity are used to understand the effect of complexity in the adaptation process and its relationship with the environment. The incremental complexity is achieved by increasing the number of sensors used as inputs, adding non-linearities, and adding memory

TABLE I

SUMMARY OF CONTROLLER ARCHITECTURES

Controller	# Parameters	# Sensors	Linear	Memory
brait2a	2	2	Yes	No
brait2b	2	2	Yes	No
brait10	10	4	Yes	No
brait20	20	9	Yes	No
ann20	20	9	No	No
ann24	24	9	No	Yes

in the form of recurrent neural network connections. Table I presents a summary of the controller architectures.

The two simplest Braitenberg controllers use only two parameters. They both take as inputs two virtual sensors, left and right, obtained from averaging and normalizing the sensor values of the three front sensors situated at the left and right sides of the robot, disregarding the three sensors in the back part. Equation 5 specifies the normalized wheel speeds $\{v_l, v_r\}$ for controller *brait2a*, where w_0 and w_1 are the parameters to be optimized; and i_l and i_r the virtual left and right sensors.

$$\begin{aligned} v_l &= 1 + w_0 \cdot i_r \\ v_r &= 1 + w_1 \cdot i_l \end{aligned} \tag{5}$$

Equation 6 defines the *brait2b* controller:

$$\begin{aligned} v_l &= w_0 + w_1 \cdot i_r \\ v_r &= w_0 + w_1 \cdot i_l \end{aligned} \tag{6}$$

Controller *brait2a* uses one parameter for each virtual sensor and a fixed bias speed set at the maximum, while controller *brait2b* uses the same parameter for both virtual sensors and has the bias speed as another parameter.

The *brait10* controller uses ten parameters. It takes as inputs four virtual sensors (front-left, front-right, back-left and back-right) obtained from averaging and normalizing in pairs the sensor values of eight sensors of the robot and discarding the central sensor in the back part. Equation 7 defines the normalized wheel speeds $\{v_l, v_r\}$ for controller *brait10*, where $\{w_0, \cdots, w_{10}\}$ are the parameters to be optimized; and $\{iv_1, \cdots, iv_4\}$ represent the four virtual sensors.

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{4} iv_k \cdot w_k \\ v_r &= w_5 + \sum_{k=1}^{4} iv_k \cdot w_{k+5} \end{aligned} \tag{7}$$

The *brait20* controller uses all the sensors as inputs (Equation 8). The wheel speeds $\{v_l, v_r\}$ depend on the normalized proximity sensor values $\{i_1, \cdots, i_9\}$, and the 20 weight parameters being optimized $\{w_0, \cdots, w_{19}\}$ (one weight per proximity sensor per wheel, and the two wheel speed biases).

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{9} i_k \cdot w_k \\ v_r &= w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10} \end{aligned} \tag{8}$$

The last two controllers are artificial neural networks. The *ann20* controller is a non-recurrent artificial neural network of two units with sigmoidal activation function $f(\cdot)$. The outputs of the units define the wheel speeds $\{v_l, v_r\}$, as shown in Equation 9. Each neuron has 10 input connections: the 9 infrared sensors and a connection to a constant bias speed.

$$\begin{aligned} v_l &= f(w_0 + \sum_{k=1}^{9} i_k \cdot w_k) \\ v_r &= f(w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10}) \end{aligned} \tag{9}$$

The *ann24* controller is a recurrent artificial neural network of two units with sigmoidal activation functions $f(\cdot)$. By recurrent we mean that the outputs of the network from the previous time step are stored in memory and used as inputs for the next time step. The outputs of the units determine the wheel speeds $\{v_{l,t}, v_{r,t}\}$, as shown in Equation 10. 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.

$$\begin{aligned} v_{l,t} &= f(w_0 + \sum_{k=1}^{9} i_k \cdot w_k + w_{10} \cdot v_{l,t-1} + w_{11} \cdot v_{r,t-1}) \\ v_{r,t} &= f(w_{12} + \sum_{k=1}^{9} i_k \cdot w_{k+12} + w_{22} \cdot v_{l,t-1} + w_{23} \cdot v_{r,t-1}) \end{aligned} \tag{10}$$

D. Environments

We conduct experiments in two different environments. The first one is an empty square arena of 2 m x 2 m, where the walls and the other robots are the only obstacles. The second environment is the same bounded arena with cylindrical obstacles added, shown in Figure 1. The obstacles have two different sizes: there are 5 large obstacles (diameter 25 cm), and 15 small obstacles (diameter 10 cm). In simulation, the obstacles are randomly repositioned before each fitness evaluation, which means that the obstacle configuration is different for each evaluation. In real-robot experiments, the obstacles are kept in fixed positions, the variation between runs is provided by the randomized initial pose of the robots.

All experiments are conducted with 4 robots. The method for initializing the robots' pose for each fitness evaluation is different between simulation and experiments with real robots. In simulation, the initial positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. For the

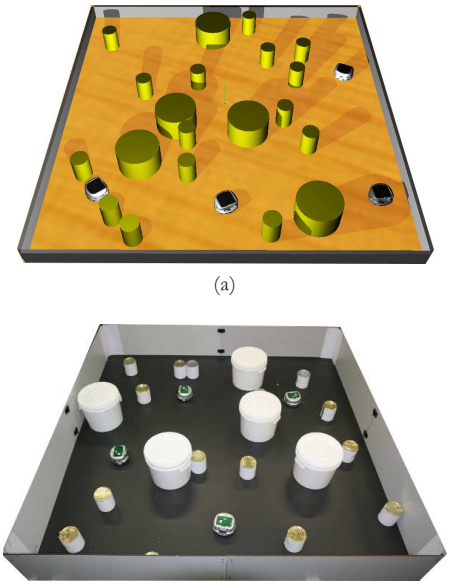

Fig. 1. Complex environment formed by outer walls and twenty cylindrical obstacles of different sizes. (a) Simulation arena (b) Real arena.

```
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.

experiments with real robots, in the empty arena a random speed is applied to each wheel for three seconds to randomize the robots' pose. In the arena with obstacles, the robots are manually repositioned to avoid disturbing the location of the obstacles, and then the robots turn in place with a random speed for two seconds to randomize the orientation.

E. Optimization Algorithm

The optimization algorithm is the distributed, noise-resistant variation of PSO introduced by Pugh et al. [15], which 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). The pseudocode for the algorithm is shown in Figure 2.

The position of each particle represents a set of weights of a controller. Each particle evaluation consists of a robot moving in the arena for a fixed time (t_e =40 s) running the controller with the weights given by that particles position. The tness corresponding to the particle is equivalent to the performance of the robot measured with function F from Eq 1.

TABLE II

PSO PARAMETER VALUES

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	24
Iterations N_i	200
Evaluation span t_e	40 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

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

$$v_{i,j} = w \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \quad (11)$$

The algorithm is implemented in a distributed fashion, which reduces the total evaluation time required by a factor equal to the number of robots. Each robot evaluates in parallel a different candidate solution and shares the solution with its neighbors in order to create the next pool of candidate solutions. Communication is used only to share the solutions, and the communication delay is negligible in comparison to the evaluation time of the controllers, which is 40 s. There is no explicit communication used to coordinate the motion of the robots.

The PSO 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.

The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented in our previous work [16] and are shown in Table II. The parameters were chosen for the more complex controller (*ann24*) and kept the same for the simpler ones in order to keep the total learning time constant for all experiments, although simpler controllers could have been optimized with less iterations and/or less particles.

IV. Results and Discussion

We begin by presenting the performance obtained in simulation for the twelve experimental conditions described in Section III. Figure 3 shows the fitness of the best set of weights found with PSO in 20 evaluation runs performed with four robots, leading to 80 fitness measurements per case. $\{A, B, C, D, E, F\}$ represent the six controllers {*brait2a, brait2b, brait10, brait20, ann20, ann24*}, e stands for empty arena, and o stands for arena with obstacles.

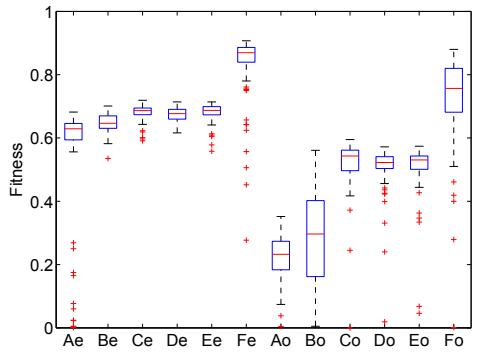

Fig. 3. Best weights evaluated in simulation. {*A, B, C, D, E, F*} represent the six controllers {*brait2a, brait2b, brait10, brait20, ann20, ann24*}, *e* stands for empty arena, and *o* stands for arena with obstacles. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

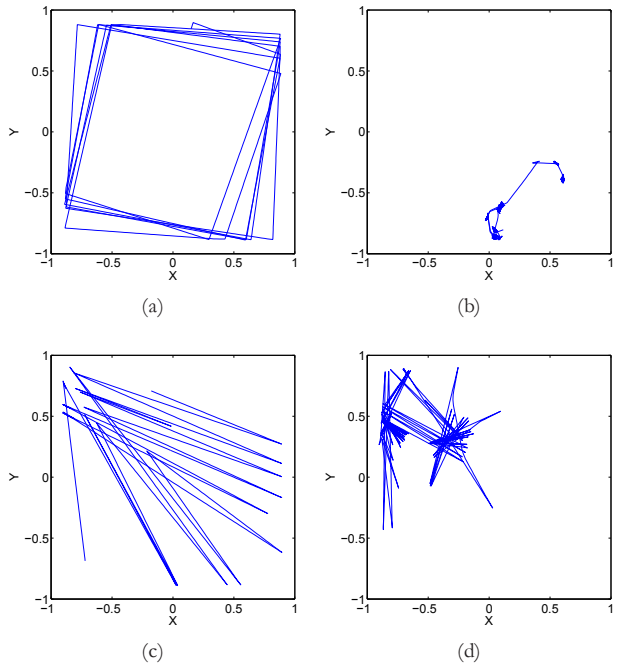

Fig. 4. Trajectories for the best weights for recurrent and non-recurrent controllers in the two environments. (a) Controller *brait20* in empty arena (b) Controller *brait20* in obstacles arena (c) Controller *ann24* in empty arena (d) Controller *ann24* in obstacles arena.

As expected, the fitness in the environment with obstacles is generally lower than in the empty environment. In both environments, the greatest gain in performance is observed when adding recurrence (e.g., 27% improvement between *ann20* and *ann24* in the empty environment, 47% in the environment with obstacles). Recurrence allows the robots to switch the direction of movement between forwards and backwards, while non-recurrent controllers move always forwards. This difference in behavior can be seen in the trajectories described by recurrent and non-recurrent controllers in each environment, shown in Figure 4. The trajectories for all the non-recurrent controllers, although not shown here, are very similar.

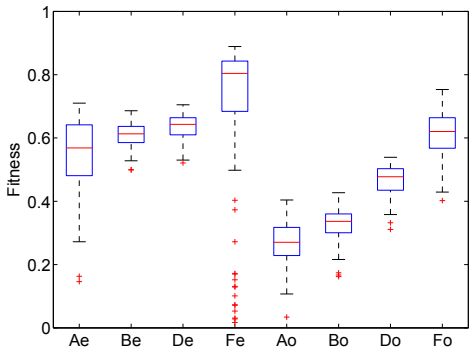

Fig. 5. Best weights evaluated with real robots.{*A,B,D,F*} represent the 4 controllers {*brait2a, brait2b, brait20, ann24*}, *e* stands for empty arena, and *o* stands for arena with obstacles.

Interestingly, increasing the number of sensors from four to nine and adding non-linearities do not bring any significant improvement in neither environment. For example, when comparing the fitness of controller *brait10* with *ann20*, there is no statistically significant difference in the empty environment (Mann-Whitney U test, $p = 0.38$), and *ann20* is slightly worse in the environment with obstacles ($p = 0.03$). However, in the environment with obstacles, going from two to four sensors (*brait2a*/*brait2b* to *brait10*) has a much larger impact on performance than in the empty environment. This suggests that the more complex environment requires the robot to be able to differentiate obstacles in the front and in the back, which is enabled by the two additional sensors.

In general, the performance difference between controllers is higher in the complex arena than in the empty one, meaning that fewer parameters may be used in simpler environments without a significant performance loss, but more complex environments require more complex controllers.

The best weights obtained in simulation for controllers *brait2a, brait2b, brait20, ann24* were also evaluated for 20 runs of 40 s with real robots in the two environments, and the results are shown in Figure 5. The performances are slightly lower, but the same trends mentioned for Figure 3 are observed: the controllers with more parameters perform better than the simpler ones, and this difference is larger in the environment with obstacles than in the empty one.

Since PSO is a stochastic optimization method, each PSO optimization run may converge to a different solution. Therefore, for statistical significance, we performed in simulation 100 PSO optimization runs for each experimental condition. Figure 6 shows the progress of the PSO optimization for the six controllers in the two environments. Note that in this case the error bars represent the variation in the best performance found at each iteration among the 100 optimization runs, which is different from the variation in the 20 evaluation runs performed with the best solution shown in Figures 3 and 5.

The optimization takes more time to converge for the more complex controllers. For example, the fitness for the recurrent ANN keeps improving during the 200 iterations, while it quickly flattens before 100 iterations for the Braitenberg

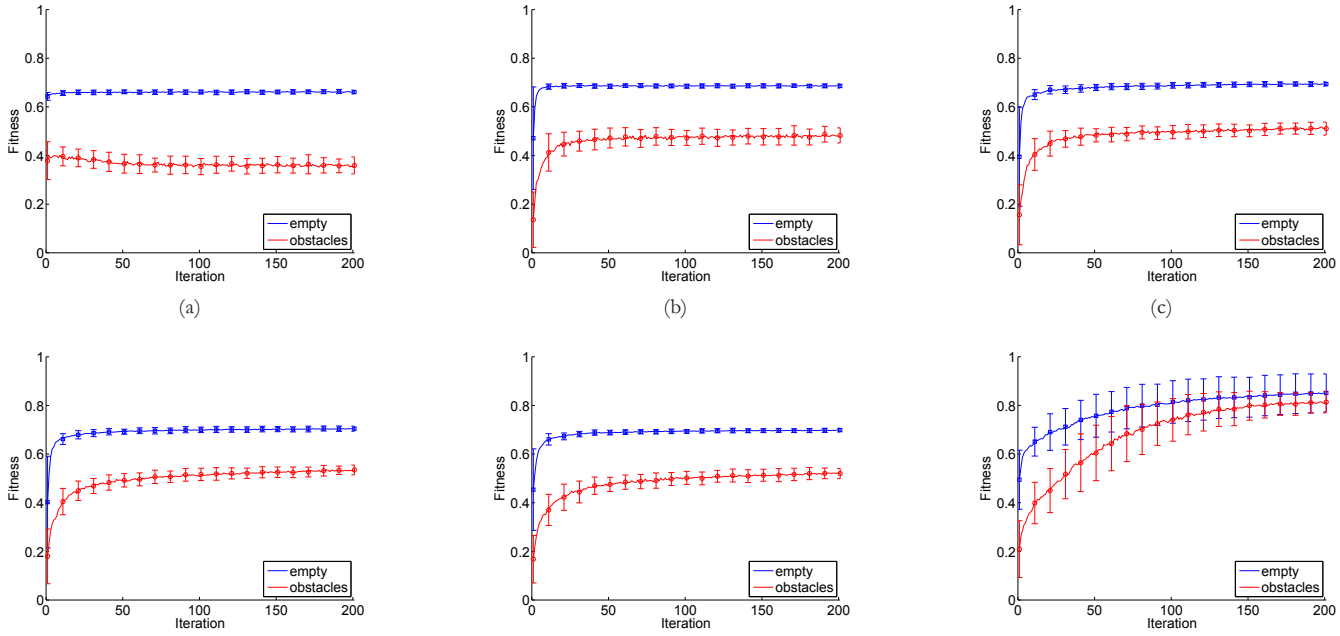

Fig. 6. Best fitness found at each iteration for 100 PSO optimization runs. Bars represent the standard deviation between runs. (a) Controller *brait2a* (b) Controller *brait2b* (c) Controller *brait10* (d) Controller *brait20* (e) Controller *ann20* (f) Controller *ann24*

controllers. This is what we expected as the search for the more complex controllers takes place in a space of higher dimension. In addition, for each controller, the optimization in the complex environment is generally noisier and takes more time to converge than in the empty environment. A notable exception is controller *brait2a*, where the average fitness does not increase with the iterations, even though the standard deviation is reduced. We suspect this is due to the fact that it is more likely to find a good solution by chance in the random initialization, as we are using 24 particles in a two-dimensional search space.

In the case of controllers *brait2a* and *brait2b* the fitness landscape can be systematically explored, which is not the case with the more complex parameters due to the high-dimensional search spaces. Figure 7 shows the fitness of controllers *brait2a* and *brait2b* in the two environments. Each point $(param0, param1)$ represents the average of 40 fitness measurements performed with the corresponding parameter values.

It should be noted that there is no distinct maximum in any of the settings. Instead, there are regions of high performance where a range of parameter values achieve similar fitness.

As expected, the performance of controller *brait2a* is symmetric with respect to the $w_0 = w_1$ line, given the symmetric disposition of the proximity sensors and the sensor clustering. In the arena with obstacles, solutions in the first quadrant (upper right) perform poorly, and the best performing regions in the second and fourth quadrants (upper left and lower right) are shifted towards higher parameter values with respect to the empty arena, implying a more aggressive turning behavior.

Regarding controller *brait2b*, the best solutions are located in a triangular region in the fourth quadrant (lower right). This region seems to be unbounded, meaning that the bias speed parameter can increase as long as the sensor weight increases proportionally to make the robots turn around obstacles, and the actual robot speed will saturate at V_{max} when there are no obstacles around.

For both controllers, the fitness in the complex environment with obstacles is lower than in the empty environment, but also the best solution regions are smaller, and thus, the optimization process is harder.

The white circles in Figure 7 mark the best points found with the 100 PSO optimization runs. As there is no distinct maximum in the fitness landscape, the solutions are spread among the high fitness areas previously described.

V. Conclusion

Our goal in this paper was to study the trade-offs between the complexity of controllers and the complexity of the environment in the distributed optimization of robotic controllers. For this purpose, we employed a multi-robot obstacle avoidance case study in which the complexity of controllers was varied by changing the number of sensors used as input, adding non-linear functions, and adding memory by using the output of the previous time step as an additional input. Experiments were conducted in two environments of different complexity, given by the number of obstacles in the environment. The optimization algorithm was a distributed, noise-resistant variation of PSO.

In the simple environment, linear controllers with only two parameters performed similarly to more complex non-linear controllers with up to twenty parameters, even though

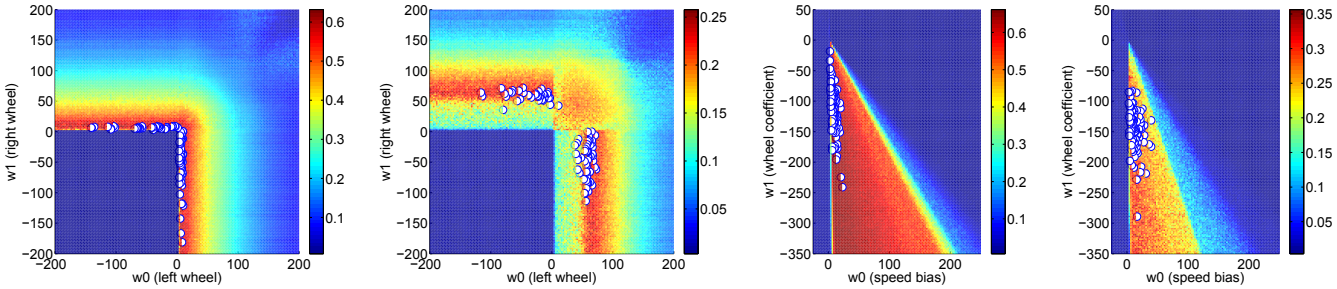

Fig. 7. Fitness landscape for the two-parameter controllers. The solutions found by PSO are marked with white circles. (a) *brait2a* in empty arena (b) *brait2a* with obstacles (c) *brait2b* in empty arena empty (d) *brait2b* with obstacles.

the latter ones required more iterations to be learned while the simpler ones could have been designed by hand or with a systematic search of the parameter space. Only the addition of memory resulted in a significant improvement in performance.

However, in the more complex environment, the difference in performance between controllers was more noticeable. The first significant performance improvement was seen when the number of sensors was increased so robots were able to differentiate between obstacles in the front and in the back, and the second improvement was due to the addition of memory by using the output of the previous time step as an additional input. These differences in performance justifies the use of more complex controllers with a larger number of parameters in more complex environments.

Regarding the effects of the environment on the optimization process, we showed that in complex environments the optimization problem was harder in three aspects: the performance measurements were noisier, the optimal parameter region was smaller, and more iterations were required for the optimization process to converge.

These challenges motivate our ongoing effort to study distributed, noise-resistant adaptation techniques that can optimize high-performing robotic controllers quickly and robustly. As future work, we intend to explore different controller architectures that can be parametrized and optimized, as well as other performance metrics that results in different avoidance 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] J. Pugh and A. Martinoli, "Distributed scalable multi-robot learning using particle swarm optimization," *Swarm Intelligence*, vol. 3, no. 3, pp. 203–222, May 2009.

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

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

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

[6] 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.

[7] S. B. Akat and V. Gazi, "Decentralized asynchronous particle swarm optimization," in *IEEE Swarm Intelligence Symposium*. IEEE, Sept. 2008.

[8] 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.

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

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

[11] J. Hereford and M. Siebold, "Using the particle swarm optimization algorithm for robotic search applications," in *IEEE Swarm Intelligence Symposium*, 2007, pp. 53–59.

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

[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. IASTED/ACTA Press, 2001, pp. 289–294.

[14] H. Pan, L. Wang, and B. Liu, "Particle swarm optimization for function optimization in noisy environment," *Applied Mathematics and Computation*, vol. 181, no. 2, pp. 908–919, Oct. 2006.

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

[16] E. Di Mario and A. Martinoli, "Distributed Particle Swarm Optimization for Limited Time Adaptation in Autonomous Robots," in *International Symposium on Distributed Autonomous Robotic Systems 2012*, Springer Tracts in Advanced Robotics 2014 (to appear).

[17] E. Di Mario, I. Navarro, and A. Martinoli, "The effect of the environment in the synthesis of robotic controllers: A case study in multi-robot obstacle avoidance using distributed particle swarm optimization," in *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, Advances in Artificial Life, ECAL 2013*, 2013, pp. 561–568.

[18] B. Al-Kazemi and S. Habib, "Complexity analysis of problem-dimension using PSO," in *WSEAS International Conference on Evolutionary Computing*, 2006, pp. 45–52.

[19] J. E. Auerbach and J. C. Bongard, "On the relationship between environmental and morphological complexity in evolved robots," in *Genetic and Evolutionary Computation Conference*. ACM Press, 2012, pp. 521–528.

[20] R. E. Palacios-Leyva, R. Cruz-Alvarez, F. Montes-Gonzalez, and L. Rascon-Perez, "Combination of reinforcement learning with evolution for automatically obtaining robot neural controllers," in *IEEE International Conference on Evolutionary Computation*, 2013, pp. 119–126.

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