A Comparison of PSO and Reinforcement Learning for Multi-Robot Obstacle Avoidance

Ezequiel Di Mario, Zeynab Talebpour, and Alcherio Martinoli
Distributed Intelligent Systems and Algorithms Laboratory
École Polytechnique Fédérale de Lausanne
{ezequiel.dimario, zeynab.talebpour, alcherio.martinoli}@epfl.ch

***Abstract*—The design of high-performing robotic controllers constitutes an example of expensive optimization in uncertain environments due to the often large parameter space and noisy performance metrics. There are several evaluative techniques that can be employed for on-line controller design. Adequate benchmarks help in the choice of the right algorithm in terms of final performance and evaluation time. In this paper, we use multi-robot obstacle avoidance as a benchmark to compare two different evaluative learning techniques: Particle Swarm Optimization and Q-learning. For Q-learning, we implement two different approaches: one with discrete states and discrete actions, and another one with discrete actions but a continuous state space. We show that continuous PSO has the highest fitness overall, and Q-learning with continuous states performs significantly better than Q-learning with discrete states. We also show that in the single robot case, PSO and Q-learning with discrete states require a similar amount of total learning time to converge, while the time required with Q-learning with continuous states is significantly larger. In the multi-robot case, both Q-learning approaches require a similar amount of time as in the single robot case, but the time required by PSO can be significantly reduced due to the distributed nature of the algorithm.**

I. 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 applying a designed controller to real robots there might be an unexpected drop in performance due to a number of factors such as imperfections in fabrication, changes in the environment, or modeling inaccuracies.

Machine-learning techniques provide an alternative to human-guided design that can address the previously mentioned challenges. In particular, evaluative methods can automatically synthesize robotic controllers in large search spaces, coping with discontinuities and nonlinearities, and find innovative solutions not foreseen by human designers. Furthermore, the learning process can be implemented fully on-board, enabling automatic adaptation to the underlying hardware and the environment.

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

However, the main drawback of working in an on-line, evaluative framework is the large amount of time needed to characterize the performance of candidate controller solutions. The noise in performance evaluation may arise from several sources of uncertainty, such as sensor noise, manufacturing tolerances, or lack of strict coordination in multi-robot settings.

In this paper we focus on two types of evaluative methods that have been employed for robotic controller design: Particle Swarm Optimization, a population-based metaheuristic, and Q-learning, a Reinforcement Learning-type algorithm.

Particle Swarm Optimization can be used to implement the adaptation process in multi-robot systems in a distributed fashion, which reduces the required evaluation time through parallelization, but requires the evaluation of multiple candidate solutions. Q-learning, on the other hand, can iteratively refine a single policy, which may reduce the required evaluation time.

In order to quantitatively compare the two evaluative learning techniques, we use multi-robot obstacle avoidance as a benchmark task. By carefully defining our testing scenario, we can quantify the differences between the two algorithms in terms of performance, total evaluation time, and their resulting behaviors.

II. Related Work

Optimization algorithms are typically evaluated on standardized numerical benchmark functions, such as DeJong's test suite [1]. The design of high-performing robotic controllers is an instance of an optimization problem under uncertainties, yet to our knowledge there is no agreed set of robotic benchmark tasks. Obstacle avoidance was used in one of the earliest works of evaluative adaptation with Genetic Algorithms applied to real robots [2], and it has also been employed to test other learning algorithms such as Particle Swarm Optimization [3] and Reinforcement Learning [4]. However, due to different environments and performance metrics, it is not possible to establish direct comparisons among algorithms based on these previous studies.

We choose to keep 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.

Particle Swarm Optimization (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/decentralized implementation due to its distinct individual and social components and its use of the neighborhood concept. Most of the work on distributed implementation has been focused on benchmark functions running on computational clusters [7]–[9]. Implementations with mobile robots are mostly applied to odor source localization [10], [11], and robotic search [12], 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 local minima, which does not represent a complex optimization problem.

Most of the research on optimization in noisy environments has focused on evolutionary algorithms [13]. The performance of PSO under noise has not been studied as 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 [14]. Pan et al. proposed a hybrid PSO-Optimal Computing Budget Allocation (OCBA) technique for function optimization in noisy environments [15]. 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 [3], [16].

In our previous work [17], 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 to reduce the total evaluation time so that it is feasible to implement the adaptation process within the limits of the robots' energy autonomy without renouncing the benefits of a population-based, evaluative learning algorithm.

Reinforcement Learning (RL) [18] is a learning method which tries to maximize the expected cumulative reward for an agent during its lifetime using the interaction with the environment. RL attempts to learn the optimal policy, which can be thought of as a mapping from the system's perceptual states to its actions, using the reward signal in each step. There have been numerous works on applying RL methods to the robotic domain. An extensive survey can be found in [19].

Mobile robots in particular have been the focus of study for a number of researchers in this area. [20] presents a framework for using RL on mobile robots with the ability to incorporate human knowledge about the task. In the initial phase, the RL system passively observes the states, actions and rewards encountered by the robot until the policy is good enough to control the robot. [21] introduces Bayesian-discrimination-function-based Reinforcement Learning (BRL) which adaptively segments the state and action spaces through the learning process, eliminating the need for the state and action spaces to be designed by a human. This method has proven to be effective at handling problems in multi-robot systems which operate in a dynamic environment. In [22] RL has been formulated to solve the multi-robot obstacle avoidance problem in a noisy and dynamic environment while reducing the space of states and actions through the use of conditions and behaviors.

Q-learning is a common RL method which learns the utility of performing actions in particular states. In [4] a neural network is used to store the Q-values for a continuous state and discrete action problem. This formulation is shown to enhance the learning ability of the agent for solving the obstacle avoidance problem in a complicated and unpredictable environment. In [23] a function approximation method based on radial basis functions and Gaussian functions is used for estimating the state value function in a biped robot control problem. The learning algorithm proposed by the authors is swarm reinforcement learning, which combines concepts from population-based methods with reinforcement learning.

The continuous nature of the obstacle-avoidance task both in terms of the states (sensory information) and actions (wheel speeds) complicates the use of the conventional Q-learning method. Therefore, we have chosen two different approaches to manage the complexity in the size of state and action spaces. In our first approach, state and action spaces are discretized using a fixed number of intervals. In the second approach, a neural network is used as a function approximator to store the Q-values with a continuous state space, as proposed by [4].

III. Materials and Methods

This paper discusses a case study of multi-robot obstacle avoidance, a basic behavior in robotics. Robots navigate autonomously in a square arena of 1 m^2 in which walls and other robots are the only obstacles. We use the same metric of performance that Floreano and Mondada defined for their homing experiment in an empty arena in [2]. The metric of performance consists of two factors, both normalized to the interval [0, 1] (Eq. 1)

$$f = f_v \cdot (1 - f_i) \tag{1}$$

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

$$f_i = \frac{1}{N_{eval}} \sum_{k=1}^{N_{eval}} i_{max,k} \tag{3}$$

$\{v_{l,k}, v_{r,k}\}$ 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 forwards or backwards 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 infra-red sensors as well as five ultrasound sensors for short and medium range obstacle detection. The experiments were carried out in simulation using Webots [24], a physics-based simulator that models dynamical effects such as friction and inertia, and individual sensors and actuators (Figure 1).

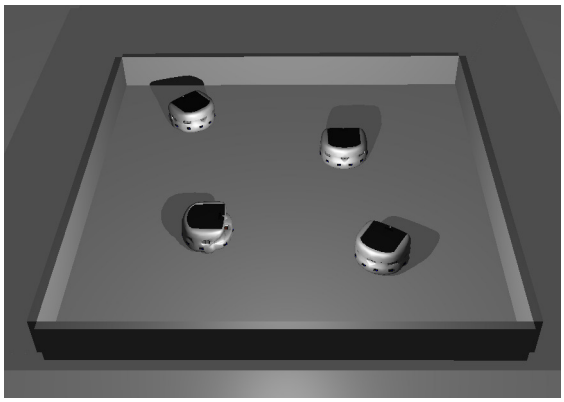

Fig. 1. Simulation of 4 Khepera III robots navigating in a square arena.

Since the response of the Khepera III proximity sensors is not a linear function of the distance to the obstacle, the proximity values were linearized and normalized using measurements of the real robot sensor's response as a function of distance. This linearization 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 or larger than 10 cm.

The multi-robot obstacle avoidance task as described presents four distinct sources of uncertainties in the performance evaluations: the proximity sensors' noise, the robots' wheel slip, the initial pose for each evaluation, and the behavior of other robots in the arena (which constitute moving obstacles).

The controller architecture for PSO is a linear Braitenberg controller (Equation 4). The wheel speeds $\{v_l, v_r\}$ depend on the normalized proximity sensor values $\{i_1, \cdots, i_9\}$, and the 20 weight parameters $\{w_0, \cdots, w_{19}\}$ (one weight per proximity sensor per wheel, and the two wheel speed biases).

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{9} i_k \cdot w_k \\ v_r &= w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10} \end{aligned} \tag{4}$$

The optimization problem for PSO then becomes choosing the set of weights $\{w_0, \cdots, w_{19}\}$ such that the fitness function f as defined in Equation 1 is maximized. It should be noted that even thought the wheel speed is a linear function of the proximity sensor values, there is no explicit mathematical expression of the fitness as a function of the weight parameters of the controller. This mapping could result in a more or less nonlinear, discontinuous landscape, which depends on the direct interactions between the robot and the environment, which are not known in advance. This justifies the use of a black-box optimization metaheuristic such as PSO.

In addition to the continuous Braitenberg controller, two discrete controller versions were implemented to analyze the impact of discretization on the final performance and to compare with the two Q-learning approaches. In the first case, the input proximity sensor values are discretized to binary values using a threshold of 0.5, which corresponds to half of the proximity senors' range (10 cm), and the output speeds are discretized to the closest of the 5 values $\{\pm v_{max}, \pm v_{max}/2, 0\}$. In the second case, the proximity sensor values remain continuous and the output speeds are discretized to the closest of the 3 values $\{\pm v_{max}, 0\}$.

```
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:     Share personal best
8:   end for
9: end for
```

Fig. 2. Noise-resistant PSO algorithm

The PSO algorithm is the noise-resistant variation introduced by Pugh et al. [16], which consists in 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 movement of particle i in dimension j depends on three components: the velocity at the previous step weighted by an inertia coefficient w_I, a randomized attraction to its personal best $x^*_{i,j}$ weighted by w_p, and a randomized attraction to the neighborhood's best $x^*_{i',j}$ weighted by w_n (Eq. 5). $rand()$ is a random number drawn from a uniform distribution between 0 and 1.

$$\begin{aligned} V_{i,j} &:= w_I \cdot V_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) \\ &\quad + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \qquad (5) \\ x_{i,j} &:= x_{i,j} + V_{i,j} \qquad (6) \end{aligned}$$

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, the robots' pose is randomized to reduce the influence of the previous evaluations. At the end of each optimization run, the best solution is tested with 40 evaluations 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 (N_{re}), as shown in Eq. 7. This equation does not take into account the time required for the final performance evaluations.

$$t_{tot} = t_e \cdot N_p \cdot N_i \cdot (N_{re} + 1) \tag{7}$$

In a parallelized or distributed implementation, fitness evaluations are distributed among N_{rob} robots, and the wall-

TABLE I. PSO PARAMETER VALUES

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

clock time t_{wc} required to evaluate candidate solutions is reduced (Eq. 8).

$$t_{wc} = t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot N_i \cdot (N_{re} + 1) \tag{8}$$

The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented in our previous work [17] and are shown in Table I.

To solve the problem of obstacle avoidance with Reinforcement Learning we have used the Q-learning method. Q-learning attempts to learn the quality of a state-action combination $Q(s_t, a_t)$ using the update formula shown in Equation 9. α is the learning rate, r is the reward at each time step, and γ is the discount factor. The states are given by the proximity sensor values at each time step, and the actions are the possible wheel speeds. s_t is the current state of the robot, a_t is the current action of the robot in s_t, s_{t+1} is the next state that the robot will encounter after performing a_t in s_t and a_{t+1} stands for all possible actions that the robot can perform in its next state.

$$Q(s_t, a_t) := (1 - \alpha)Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] \tag{9}$$

The problem of perceptual aliasing occurs when different states in the world appear to be similar from the perception of the robot but require different responses. This aliasing is due to the partial observation of the world in our problem. Since the robot is not aware of its absolute position in the arena and all of its surroundings including other obstacles out of its range, it can receive identical sensory information in different parts of the arena. Imagine a position A in the arena where there is no obstacle sensed by the robot but if the robot moves one step forward it would sense an obstacle also imagine another position B where there are no obstacles sensed but if the robot moves one step back it would sense an obstacle. Positions A and B are mapped to the same state from the perception of the robot whereas they require different actions.

Because of partial observation, we have chosen a softMax probabilistic actions selection policy that allows better actions to be chosen according to how high their Q-value is, balancing exploration and exploitation (Equation 10). The temperature T is set so that it gradually decreases and remains a small but positive value to allow the actions that are nearly as good to have a chance to be selected.

$$p(s, a) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}} \tag{10}$$

TABLE II. Q-LEARNING PARAMETER VALUES FOR THE FIRST APPROACH

Parameter	Value
Learning Rate α	$\alpha_0 = 1$ $\alpha_{k+1} = \alpha_k / 1.0001$ $\alpha \geq 0.3$
Discount Factor γ	0.5
Temperature T	$T_0 = 20$ $T_{k+1} = T_k / 1.0008$ $T \geq 0.05$
Binary Threshold	0.5

The reward signal used at each time step is the same as the fitness function that we are aiming to optimize using PSO, given in Equation 1. Unlike most RL problems, in this problem there is not a concrete final goal. Instead, we are concerned with achieving a high fitness and maintaining it throughout the lifetime of the robot.

One key feature of RL methods which is not present in our problem formulation is the use of intermediate rewards in order to reach a goal or find a solution. Incorporating appropriate intermediate rewards can significantly speed up the learning process. However, we have chosen not to alter the reward signal from the fitness function due to two reasons. Firstly, not modifying the reward enables us to directly compare the progress of the learning in terms of fitness as a function of time with PSO. Secondly, the role of intermediate rewards in shaping the behavior of the robot makes defining an appropriate intermediate reward signal without creating misleading biases a challenging problem.

In our first Q-learning approach, the state and action space have been discretized to overcome the complexity arising from continuous state and action spaces when dealing with RL methods. Discretization decreases the size of state and action spaces, thus speeding up the learning, but also results in a performance drop in terms of fitness. We have discretized the sensory information using a predefined threshold to indicate safe and unsafe zones in terms of the chance of collision. This thresholding implies that we will have a binary coding of each sensors information and all sensors together will form the state of the robot in every step. We have set five possible speed levels for each wheel: $\{\pm v_{max}, \pm v_{max}/2, 0\}$. There are two wheels and nine distance sensors on each robot, resulting in a total number of 5^2 different possible actions and 9^2 possible states. The learning parameter values for the first Q-learning approach are shown in Table II.

In our second approach, continuous state and discrete action space Q-learning, the Q-values are stored in a neural network to allow a more compact representation of the states and also interpolation for the unvisited state-action pairs. In every step of the simulation, the robot senses the environment through its sensors, which are the input nodes of the neural network. The outputs of the network specify the Q-values for that state with each output corresponding to one state-action pair. Unlike [4], a softMax selection policy is used to select an action to be performed. The reward perceived from the environment is used to calculate the new value for the selected state-action pair using the Q-learning update formula (Equation 9).

The weights of the neural network are adjusted every step of the simulation using the back propagation algorithm (BP).

TABLE III. Q-LEARNING PARAMETER VALUES FOR THE SECOND APPROACH

Parameter	Value
Learning Rate α	$\alpha_0 = 1$ $\alpha_{k+1} = \alpha_k / 1.000001$ $\alpha = 0$ after episode 1000
Discount Factor γ	0.5
Temperature T	$T_0 = 20$ $T_{k+1} = T_k / 1.000007$

The error signal used for the adjustment is the difference between the new and old Q-values for the selected state-action pair. All other output nodes will have target values equal to their Q-values in the previous step, and therefore the error signal will be zero for all unselected actions.

In order to reduce the complexity of the neural network, the number of possible actions was reduced with respect to the first approach. There are three action levels to choose from, $\{\pm v_{max}, 0\}$ for each wheel, which makes the total number of possible actions for every state to be 3^2. The number of inputs is the same as the number of sensors which is nine. There are 18 nodes in the hidden layer, and nine output nodes, corresponding to the Q-values of the nine possible actions. The activation function used for the hidden and output layer is sigmoid. After the error signal is specified, the network weights are tuned $k = 4$ times with the same input and output for better adjustment. Table III contains the parameters used for this approach.

We have conducted 2 sets of experiments for each Q-learning approach. The first experiment involves a single robot moving in the arena, and the second experiment involves 4 robots moving in the arena at the same time. When there are 4 robots learning at the same time, there is no direct communication to assist in solving the problem. The robots play the role of dynamic obstacles for one another, creating a harder more complex dynamic instance of the obstacle avoidance problem. In the distributed implementation of PSO, on the other hand, there is solution sharing between neighbors which speeds up the learning process.

For the first RL approach each robot performs 1000 learning episodes of 20 s. The final performance of each robot is the average of the rewards during the last 200 episodes. We have tested every experiment 20 times and the results are the average of all runs. For the second approach, each experiment was conducted 100 times for 1500 episodes of 20 s. The evaluation phase goes from episode 1000 to 1500.

IV. Results and Discussion

The results of this paper are presented as follows: Section IV-A shows the performance of PSO in terms of fitness and evaluation time. Section IV-B presents a similar analysis with Q-learning and discusses the differences between the two approaches. Finally, Section V concludes the paper with a summary of our findings and presents an outlook of our future work.

A. PSO Results

Figure 3 shows the final performance obtained by applying PSO under six different experimental conditions: 3 different discretization levels each tested with 2 different number of robots. The 3 different discretization levels are abbreviated as follows: CC stands for continuous sensors and continuous speeds, CD continuous sensors and discrete speeds (3 possible output speeds), and DD discrete sensors and discrete speeds (binary sensors and 5 possible output speeds). Each discretization level was tested with 1 and 4 robots.

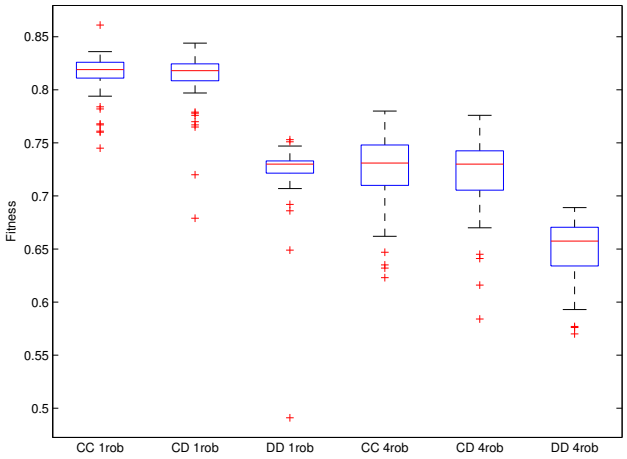

Fig. 3. Final performance for 100 runs of PSO for three different discretization levels, each implemented using 1 and 4 robots. CC stands for continuous sensors and continuous speeds, CD continuous sensors and discrete speeds, and DD discrete sensors and discrete speeds. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

The purpose of the discretized Braitenberg PSO optimization runs is to differentiate the impact of discretization from the learning when comparing with Q-learning, which works with discrete state and actions and therefore requires the discretization of proximity sensor inputs and wheel speed outputs. Discretizing only the output speeds (CD controllers) has no statistically significant impact on the final fitness when compared with the fully continuous controllers (CC), both in the single and in the multi-robot case (Mann-Whitney U test, $p = 0.32$ and $p = 0.34$ respectively). Discretizing also the proximity sensors (DD controllers), however, has a noticeable impact on the fitness. For the single robot case, the mean drops from 0.82 to 0.72, a statistically significant performance difference (Mann-Whitney U test, $p = 2.7e-34$). For the multi-robot case, the mean drops from 0.73 to 0.65 ($p = 9.9e-29$).

Figure 4 shows the progress of the PSO optimization as a function of evaluation time for continuous (CC) and discretized (DD) Braitenberg controllers with 1 robot in the arena. For all temporal progress graphs, the horizontal axis was converted from iterations to evaluation time in seconds to enable comparisons among algorithms regardless of how evaluation time is assigned (i.e. length of episodes, iterations, etc.) For both controllers, the fitness of the best solution found by the swarm increases rapidly during the initial 10000 s, and then continues to increase although at a much lower pace. Also, the standard deviation between runs decreases with time as the optimization process converges towards a high-performing solution. The discretization lowers the final fitness but it does not seem to affect the convergence time of the algorithm.

Figure 5 shows the progress of PSO as a function of

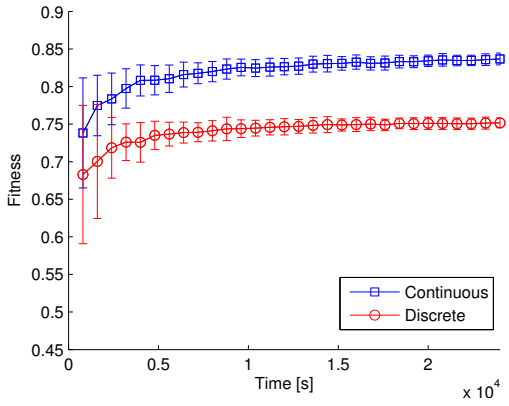

Fig. 4. PSO best fitness as a function of time for continuous and discretized Braitenberg controllers with 1 robot in the arena. The curves are the average of 100 independent PSO runs, markers are placed at each iteration, error bars represent one standard deviation.

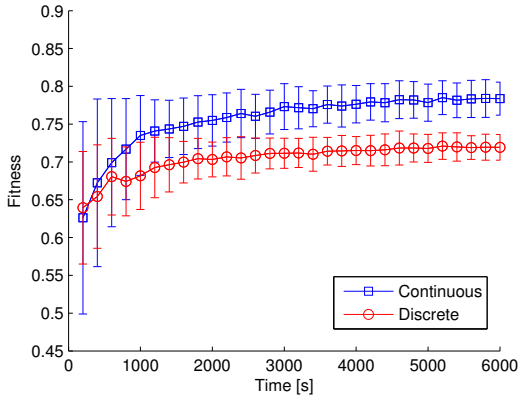

Fig. 5. PSO best fitness as a function of time for continuous and discretized Braitenberg controllers with 4 robots in the arena.

evaluation time with 4 robots in the arena, again for continuous (CC) and discretized (DD) Braitenberg controllers. When comparing between 1 and 4 robots, it can be noted that in the multi-robot case the fitness is not only lower but also much noisier due to the effect of other uncoordinated robots in the arena. Additionally, due to the distributed PSO implementation, the total evaluation time employed is reduced by a factor of 4 in the multi-robot case.

In order to separate the effect of the learning from the number of robots in the arena, we performed an additional control experiment using continuous controllers (CC) with 4 robots in the arena, where one robot is learning and the other 3 robots are avoiding obstacles with a previously optimized controller. Figure 6 compares the final performance obtained with one robot learning (single robot in the arena), one robot learning and 3 other robots avoiding, and 4 robots learning in the arena. The performance in both cases with 4 robots in the arena is significantly lower than the case with 1 robot due to the fact that the added robots represent more obstacles in the same area. However, there is no significant difference in the final fitness between one robot learning and 3 avoiding, and 4 robots learning (Mann-Whitney U test, $p = 0.44$), which shows that distributing the adaptation process has no significant

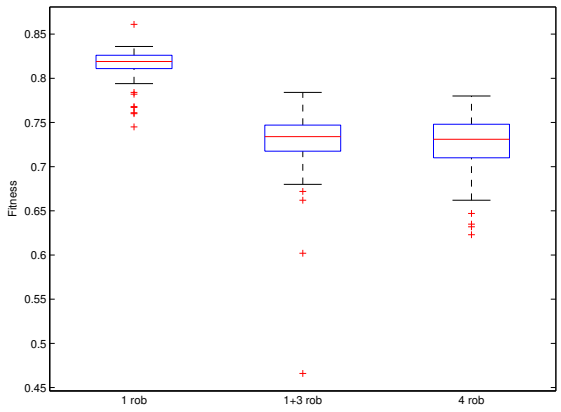

Fig. 6. Performance of final evaluations for 100 independent runs of PSO with 1 robot learning, 1 learning and 3 avoiding, and 4 robots learning, using continuous controllers (CC).

TABLE IV. MEAN AND STANDARD DEVIATION OF THE FINAL PERFORMANCE FOR THE DIFFERENT EXPERIMENTS.

Algorithm	Sensors	Speeds	N_{rob}	Mean	Std
PSO	discrete	discrete	1	0.72	0.04
PSO	continuous	discrete	1	0.81	0.02
PSO	continuous	continuous	1	0.82	0.02
PSO	discrete	discrete	4	0.65	0.03
PSO	continuous	discrete	4	0.72	0.03
PSO	continuous	continuous	4	0.73	0.03
Q-learning	discrete	discrete	1	0.72	0.15
Q-learning	continuous	discrete	1	0.73	0.25
Q-learning	discrete	discrete	4	0.55	0.17
Q-learning	continuous	discrete	4	0.72	0.22

impact on the final fitness even though it reduces the required total evaluation time by a factor equal to the number of robots. The results of the final performance for PSO under the different experimental conditions are summarized in Table IV. The next section presents the Q-learning results and compares them with the PSO results discussed in this section.

B. Q-learning Results

Figures 7 and 8 show the mean reward as a function of time for 20 runs of the Q-learning algorithm for single and multi-robot learning for the first RL approach. In the case of multi-robot learning, the average performance of the 4 robots is depicted. In the single robot case, we can see that the algorithm converges at around 8000 seconds which corresponds to episode 400. In the 4 robot case, the performance keeps improving after 8000 seconds until the end of the experiment, although at a much slower pace than during the initial episodes.

Both the single and the multi-robot case show a larger standard deviation between runs than PSO (see Table IV). This increase may be due to the probabilistic nature of the softMax policy, as other sources of uncertainties were kept constant between the different experiments.

Figure 9 shows the performance of the single robot using the second approach: continuous state Q-learning. We can see a convergence in the performance of the robot after the first 25000 seconds of the simulation, which shows a lower learning speed comparing to the first approach. This is partly due to the higher exploration and more gradual decrease of the temperature for the softMax Policy in the second RL approach. The standard deviation increases with time, but the

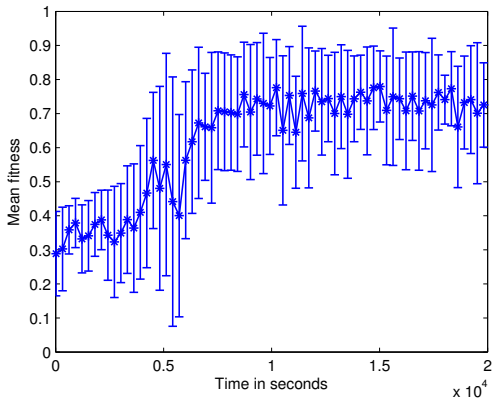

Fig. 7. Mean and standard deviation of fitness for a single robot using the first RL approach.

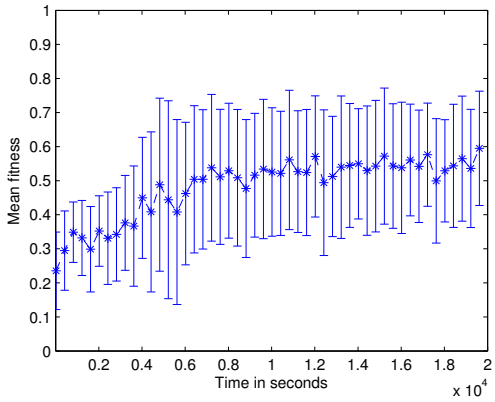

Fig. 8. Mean and standard deviation of fitness for 4 robots using the first RL approach.

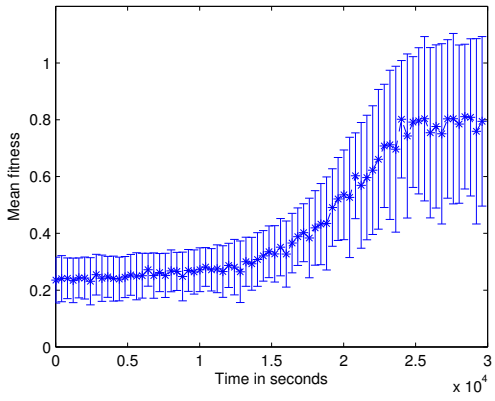

Fig. 9. Mean and standard deviation of fitness for a single robot using the second RL approach.

coefficient of variation (ratio of the standard deviation to the mean) remains constant at a value of 0.34. The behavior seen is avoiding the obstacles and moving about but mainly in the center of the arena. For the single robot case, the final performance obtained with both Q-learning approaches is very similar to the one obtained with PSO with discrete sensors and speeds, around 0.72 (see Table IV).

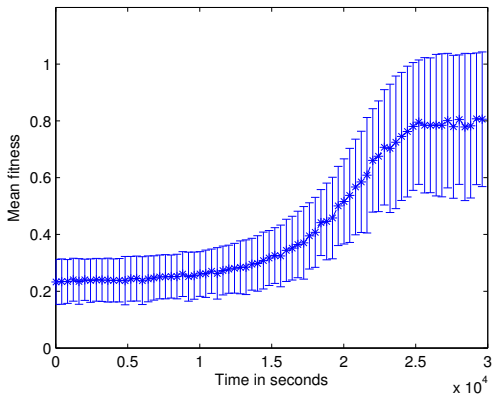

Fig. 10. Mean and standard deviation of fitness for 4 robots using the second RL approach.

Figure 10 shows the performance of the second Q-learning approach in the 4 robot scenario. The fitness and the convergence time are nearly the same as in the single robot case (compare with Figure 9). The final fitness of 0.72 is very similar to the one obtained with PSO with continuous sensors and discrete wheel speeds, and it is significantly higher than the one obtained with the first Q-learning approach.

It is noteworthy to mention that Q-learning tries to find a probabilistic mapping from states to actions, whereas the Braitenberg controller calculates the output as a linear function of the inputs. Thus, the smoother behaviors seen with PSO are partly due to the nature of the controller, whereas it is easier to see behaviors with discontinuous movements like going back and forth with Q-learning. It is therefore difficult to decouple the effects of the learning from the influence of the underlying control structure. The second Q-learning approach reduces these differences with the use of a continuous state space, but the outputs of the neural network are the Q-values of every action, and not the action itself. Therefore, the neural network should not be interpreted as the robots' controller, as the output speeds are still determined with a probabilistic state-to-action mapping.

V. Conclusion

We have presented multi-robot obstacle avoidance as a benchmark robotic task for optimization in the presence of uncertainties. The four sources of uncertainties for the given performance metric were the proximity sensors' noise, the robots' wheel slip, the initial pose, and the behavior of other robots in the arena.

We have applied two different evaluative learning techniques to solve this task: a noise-resistant variation of PSO and Q-learning. PSO was used to optimize 20 parameters of a linear Braitenberg controller. Three levels of discretization were implemented to compare with the Q-learning approaches: continuous sensors and speeds, continuous sensors and discrete speeds, and discrete sensors and speeds. In the case of Q-learning, two different approaches were presented. In the first approach, a probabilistic policy that maps discrete states to actions was learned. For the second approach, a neural network enabled us to store the Q-values for continuous states and

use the conventional Q-learning method to find an appropriate policy.

We showed that the discretization of the proximity sensors had the highest impact on the fitness for both learning algorithms. Continuous PSO had the highest fitness overall, and Q-learning with continuous states significantly outperformed Q-learning with discrete states.

Regarding the learning time, PSO and Q-learning with discrete states required a similar amount of total evaluation time for the single robot case. Both techniques converged to a solution in less than 10000 seconds. Q-learning with continuous states required more time to converge, but achieved a higher final fitness than Q-learning with discrete states. In the multi-robot case, both Q-learning approaches converged in a similar amount of time as in the single robot case but the time required by PSO was significantly reduced due to the distributed nature of the algorithm.

As future work, we intend to expand the set of robotic benchmarks with new tasks of differing complexity and to employ different controller architectures. We are also interested in testing distributed Q-learning implementations along with algorithmic variations and hybridizations of PSO and Q-learning that can be implemented in a distributed fashion. Finally, we would like to evaluate different techniques for handling uncertainties in the scenarios discussed in this paper. Our final goal is to devise a set of general guidelines for fast, robust adaptation of high-performing robotic controllers.

REFERENCES

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

[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] 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.

[4] 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.

[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] J. Chang, S. Chu, and J. Roddick, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science*, pp. 809–818, 2005.

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

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

[17] 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)*. [Online]. Available: http://infoscience.epfl.ch/record/182403

[18] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, ser. Adaptive Computation and Machine Learning. MIT Press, 1998, vol. 9, no. 5.

[19] J. Kober and J. Peters, "Reinforcement Learning in Robotics : A Survey," *Reinforcement Learning*, pp. 579–610, 2012.

[20] W. Smart and L. Pack Kaelbling, "Effective reinforcement learning for mobile robots," in *IEEE International Conference on Robotics and Automation*, 2002, pp. 3404–3410.

[21] T. Yasuda and K. Ohkura, "A reinforcement learning technique with an adaptive action generator for a multi-robot system," *From Animals to Animats 10*, pp. 250–259, 2008.

[22] M. Matarić, "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[23] H. Iima, Y. Kuroe, and K. Emoto, "Swarm reinforcement learning methods for problems with continuous state-action space," in *IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2011, pp. 2173–2180.

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