Learning Reactive Neurocontrollers using Simulated Annealing for Mobile Robots

Philippe Lucidarme, Alain Liégeois

LIRMM , University Montpellier II, France, lucidarm@lirmm.fr

Abstract

This paper presents a method based on simulated annealing to learn reactive behaviors. This work is related with multi-agent systems. It is a first step towards automatic generation of sensorimotor control architectures for completing complex cooperative tasks with simple reactive mobile robots. The controller of the agents is a neural network and we use a simulated annealing techniques to learn the synaptic weights. We'll first present the results obtained with a classical simulated annealing procedure, and secondly an improved version that is able to adapt the controller to failures or changes in the environment. All the results have been experimented under simulation and with a real robot.

1. Introduction

Cooperation of multiple mobile "autonomous" robots is a growing field of interest for many applications; mainly in industry and in hostile environments such as planet exploration and sample return missions. Theoretical studies, simulations and laboratory experiments have demonstrated that intelligent, robust and fault-tolerant collective behaviors can emerge from colonies of simple automata. This tendency is an alternative to the all-programmed and supervised learning techniques used so far. The "animats" concept thus joins the pioneering works on "Cybernetics" published in the middle of the previous century [1], for example, the reactive "Tortoise" robot proposed by Grey in 1953.

Although human supervision would obviously remain necessary for complex missions, long and tedious programming tasks would be cut out with robots capable of self-learning, self-organization and adaptation to unexpected environmental changes. Previous works have shown many advantages for self-learning robots:

1. at the lowest level, complex legged robots can learn how to stand up and walk [2],
2. a mobile robot can learn how to avoid obstacles [3] and plan a safe route towards a given goal [4 and 5],
3. a pair of heterogeneous mobile robots can learn to cooperate in a box-pushing task [6],
4. efficient global behaviors can emerge in groups of robots [7].

The bottom-up approach for building architectures of robotic multi-agent systems automatically acquiring distributed intelligence appears to be simple and efficient. However, even if we do not ignore the needs, for some applications, for communicating indirectly information (by letting the robots deposit beacons for example) direct modes are of prime interest. It has been demonstrated that even very simple information sharing induces a significant enhancement of both the individual and group performance [7,8 and 9].

The aim of this paper is to use simulated annealing to learn a reactive controller. Previous works applied to the robotics deals with the optimization of a dedicated controller. This optimization is generally simulated or done off-line. We'll focus on the on-line learning of a generic controller.

This paper will focus on the learning of reactive controllers. Complex representations of the environment or of the agents are not considered here. A library of learned behaviors will be used to perform more complex tasks with heterogeneous team of robots. The agents have different capabilities; justifying that each one must learn its own controller. In the first part of the paper we will show now, the agent can automatically learn the synaptic weights of a neural network using a classical simulated annealing procedure. In the second part, we'll propose an improved version of the method that allows the agent to adapt its controller to changes or failures.

2. Experimental setup and task description

2.1 Hypotheses

The considered task is a safe and robust reactive navigation in a clustered environment for exploration purposes. The robots are programmed a priori neither for obstacle avoidance nor for extending the explored area, and nor for executing more complex actions like

- finding a sample,
- picking up a sample,
- returning to the home base,
- dropping the sample into an analyzer.

On the contrary, the agents have to find by themselves an efficient policy for performing the complex tasks. The idea is to quickly find an acceptable strategy that maximizes the reward rather than the optimality. Our goal is to build agents that are able to reconfigure and

adapt their own controller to hardware failures or changes in the environment.

2.2 Robot Hardware

All the experiments described in this paper have been implemented on the so-called Type 1 mobile robot developed at LIRMM [10]; the previous prototype is described in [11]. Type 1 has many of the characteristics required by the multi-agent systems. It has a 10 cm-height and 13 cm-diameter cylindrical shape (Figure 1). It is actuated by two wheels. Two small passive ball-in-socket units ensure the stability in place of usual castor-wheels. DC motors equipped with incremental encoders (352 pulses per wheel revolution) control the wheels. The encoders are used for both speed control and odometry (measurement of the performance index). 16 infrared emitters and 8 receivers are mounted on the robot for collision avoidance as shown on Figure 2. The sensors use a carrier frequency of 40 kHz for a good noise rejection. These sensors are also used to communicate between agents. The communication module will not be used here. An embedded PC (80486 DX with 66 MHz clock) operates the robot. Control to sensors and actuators is transmitted by the PC104 bus.

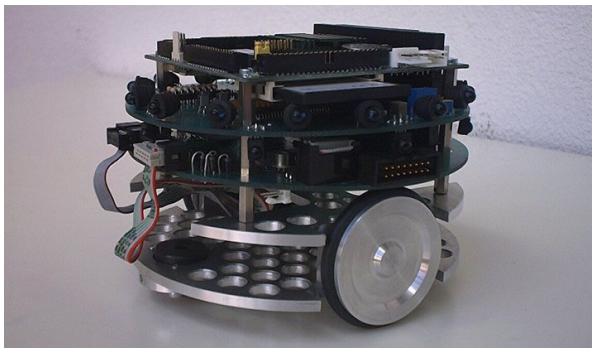

Figure 1: *The mobile robot Type 1*

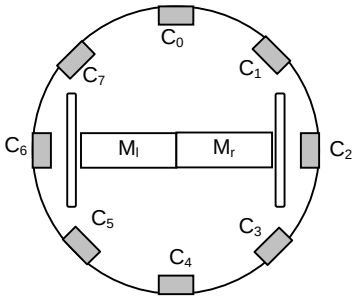

Figure 2: *Location of the sensors and actuators*

2.3 The Controller

Our purpose is to optimize the parameters of a generic controller. Many controllers for mobiles robots have been proposed, but our specifications are the following : the controller will be use for reactive task, its computation time must be small and the same controller must be applicable to different tasks. The latter specification is probably the more restrictive. It as been show that neural networks can be use to approximate many function ($R_n \rightarrow R_m$) and are not time consuming. This is why the controller used is a neural network without hidden layer. The inputs of the network are the returned values of the 8 infrared sensors (C_0 to C_7 on the Figure 2). The last input of the system is a constant equal to 1. The two outputs of the system are the commands applied to the left and right motors (M_l and M_r). The neural network is shown on Figure 3. It has been chosen for the following reason: the strategy will be learnt in the continuous space state, meaning that reactions of the agent will be proportional to its perception. In this network, there are 18 weights to learn. Each weight links an input of the network to a perceptron. As the transfer function of each perceptron is linear (Figure 4), analyzing the learned parameters will be easy. To protect the hardware during experiments the maximum speed of the robot is limited to $V_{max}=0.3$ m.s^{-1}. We'll use a simulated annealing technique to learn the 18 synaptic weights of the network. Each weight is ranging from –1 to +1.

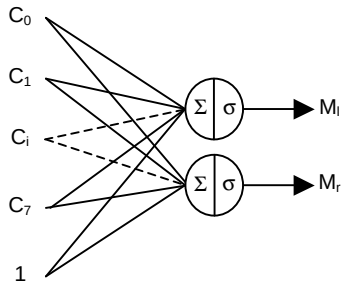

Figure 3: *The neural controller of our agent*

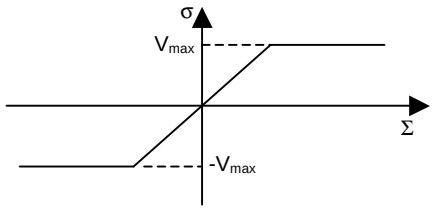

Figure 4: *The transfer function of the perceptron*

2.4 The fitness

In our application, the agent must be able to estimate its own performance also call fitness (in evolutionist algorithms) or reward (in reinforcement learning). During each elementary time step, the new average value of the fitness is computed as follow:

$$R_{N(i)}(i) = (1-\alpha(i))R_{N(i)-1}(i) + \alpha(i)F_{N(i)}(i)$$

Where

$$\alpha(i) = \frac{1}{1+N(i)}$$

N(i) is the number of time steps since the beginning of the estimation by agent *i*

$R_{N(i)}(i)$ is the estimated fitness at time *N(i)*

$F_{N(i)}(i)$ is the instantaneous fitness at time *N(i)*

The instantaneous fitness is the average rotation speed of the two wheels. An incremental encoder equips each motor of the robot. The returned value of each encoder is used to compute the fitness. It was important for this experiment to choose a non-restrictive reward. In previous works [12], the reward used to train a genetic algorithm has 3 components:

- Maximizing the speed of the robot,
- Minimizing the rotation speed of the robot,
- Minimizing the number of collisions.

Such reward proved to be too restrictive because the second term is already included in the first one. The robot can't turn and maximize its average speed at the same time. A great advantage of learning is that the agent finds good strategies, which could not be straight forward for the operator. Then the chosen fitness in our application is only the average speed of the robot.

3. First experiments: simulated annealing procedure

3.1 Description

Initialization

$R_{max} \leftarrow 0$
Initialize each weight (w_j) to a small value.
$T° \leftarrow F_t(0)$

Main loop

```
While (T° > small value) do
{
      Apply the current strategy, and
      compute the fitness R_N(i)
      If (R_max<R_N(i))
      {
            R_max=R_N(i)
            For each weight : W_max(j) ← w_j
      }
      T° ← F_t(cycle)
      For each weight (w_i), randomly
      compute a new value centered on
      W_max(i) with a distribution
      proportional to T°.
}
```

Table 1: *The algorithm used to train the neural network*

First, our agent learns the weights of the neural network using a classical simulated annealing algorithm. The algorithm is described in Table 1. To avoid having the robot jammed every time it hits an obstacle, an "unjam behavior" have been implemented. If the returned value on each encoder is equal to zero during a pre-defined time, the program will considered that the robot is jammed, and will execute a small procedure to unjam it. During this procedure, the fitness is always computed. As the robot moves back, the execution of the procedure penalizes the agent, such that being jammed is never profitable.
The learning process is divided into cycles. One cycle lasts 23 seconds and is also called evaluation of the strategy. One cycle is composed of 2000 elementary time steps, which represent the duration of a sensori-motor update.

3.2 The parameters

A main drawback in the use of a simulated annealing procedure is the setting up of the parameters. In this section, each parameter will be described in details. To implement the neural network, each value ranges from –1 to +1. Each sensor returns a value between 0 and 1 depending on whether no obstacle is detected or is very close respectively. The applied command on the motors also ranges from -1 to +1. We arbitrary chose a linear decreasing function F_t(cycle) for the temperature as indicated on Figure 5. Another functions will be tested.

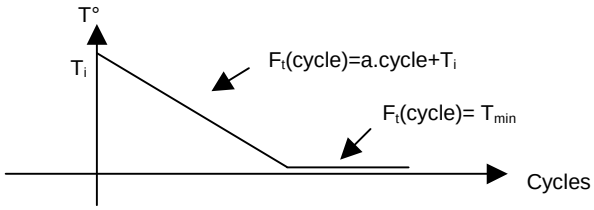

Figure 5: *Evolution of the temperature versus the number of cycles*

The function is linear until a very small value of the minimal temperature $T_{min} = 5.10^{-3}$. This enables the algorithm to converge into the maxima when the learning process is over. The initial temperature (T_i) is equal to 1. We first simulated the learning process with Matlab. We voluntary chose a very small negative value for a = -5.10^{-3}. Decreasing slowly the temperature guarantees that the state space will be explored and the optimal solution will be found. For real experiments, the autonomy of the robot is about 90 minutes. We

decomposed this time in two parts: about 1 hour of learning and 30 minutes with the temperature equal to T_{min}. The evaluation of a policy requires 23 seconds. To reach T_{min} in one hour, the parameter a must be equal to -6.10^{-3}.

3.3 Results

Simulation results: we realized many experiments with the simulator, mainly to study the influence of the parameters. Our first analyze is that the algorithm always converges to the optimal strategy if the temperature decreases slowly and if the evaluation period is long enough. The evolution of the strategy is always the same. Figure 6 shows the average speed and the rotation speed of the robot versus the number of cycles. We can see that the first strategy learned is to turn slowly at the same place. This is a local maxima because turning on place ensures that no collision occurs. Then the rotation speed increases quickly as well as the average speed of the robot. During the last part of the learning process, the radius of the circles described by the robot increases slowly until that the trajectory can be considered as a straight line. At the end of the learning process the agent avoids obstacles with the best strategy:

- turning on the left if an obstacle is detected on the right,
- turning on the right if an obstacle is detected on the left,
- going straight otherwise.

The only difference between experiments is the priority when a front obstacle is detected. Our robot is equipped with a central distance sensor (C_0 on Figure 2). When this sensor detects an obstacle, the network gives arbitrary the priority to the left or to the right. There are two global maxima in the state space, witch both represent the optimal strategy.

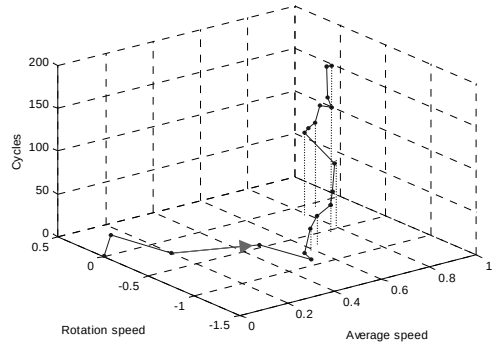

Figure 6: *Evolution of the best known strategy during learning*

Experimental results: Figure 7.a. shows the evolution of the weights. Convergence is ensured, and the reward is maximized as shown on Figure 7.b. even though the global solution is not always found. There are two differences between simulations and real experiments: the parameter a and the noise. 10 simulations have been performed with the same value of a, and the convergence to the global minima is always reached. Analyzing the results demonstrates that the same behavior may return different fitness with an important distribution, depending of many parameters like the initial position of the robot. During the experiments, one of the first evaluations can give a better reward than the average expected for this strategy, and this best fitness enables another strategy to overwrite this one. Let's take a critical situation for example: the agent performs the following strategy: "always going straight". If the initial position of agent allows him to perform a straight trajectory without meeting obstacles, the reward will be high and yet the strategy is not so good. A solution to this problem is to increase the duration of an experiment in order to decrease the standard deviation of the fitness witch is at the expense of the learning time.

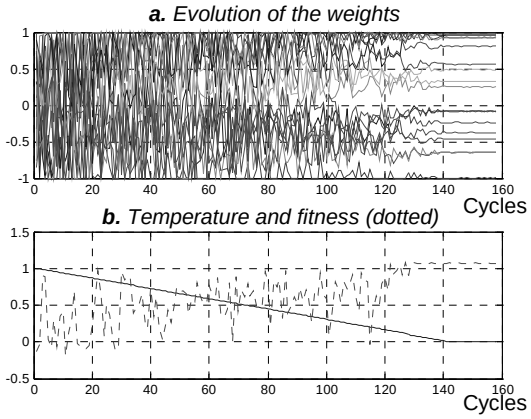

Figure 7: *results of an experiment*

4. Improved simulated annealing procedure

4.1 Description

With the previous method, when the temperature reaches a very small value, the strategy of the agent is frozen. If a failure or a change in the environment occurs the agent will not be able to adapt its controller to the changes. The only way to detect such event, without using complex representation of the agent structure or environment map, is to exploit the information returned by the fitness. If a change occurs, the reward will decrease, otherwise this change has not affect the performance of the agent and adaptation is unnecessary. The main idea of this adaptive method is to allow the growth of the temperature when the fitness is small as in the real simulated annealing process. To generalize; the temperature is a decreasing function of the best known fitness as shown on Figure 9.

The drawback is that the system will probably be trapped into local maximas. Our philosophy is the following: if the fitness function has been well chosen,

we don't care if the learned strategy is a local or global maxima while the agent maximizes its reward. The best known fitness (R_{max}) is decreased during each cycle of the main loop. If the learned strategy is enough efficient, R_{max} is currently updated and the controller stays stable. Otherwise, if the fitness is small, R_{max} will decrease, allowing the growth of the temperature. The algorithm is described on the Table 2.

Initialization

$R_{max} \leftarrow 0$
Initialize each weight (w_j) to a small value.
$T° \leftarrow F_t(0)$

Main loop

```
While (true) do
{
      Apply the current strategy, and
      compute the fitness R_N(i)
      If (R_max<R_N(i))
      {
            R_max=R_N(i)
            For each weight w_i: W_max(i)
← w_i
      }
      T° ← F_t(R_max)
      Decrease R_max
      For each weight (w_i), randomly
      compute a new value centered on
      W_max(i) with a distribution
      proportional to T°.
}
```

Table 2: *The adaptive algorithm used to train the neural network*

4.2 The parameters

The network parameters are the same as previously. The new function of the temperature is also linear. The purpose is to get a very small temperature for high fitness values and on the contrary a temperature close to 1 when the fitness is small. As the best expected reward is close to 1, we simply choose the function shown on Figure 9.

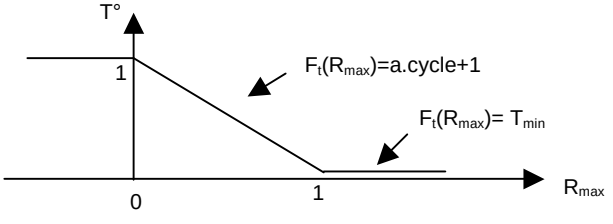

Figure 9: *Evolution of the temperature versus the best known fitness*

$T_{min}=5.10^{-3}$ as with the previous algorithm and a=-1 to linearly reach T_{min} when R_{max} is close to 1. A decreasing step of R_{max} is equal to 7.10^{-2}. This value has been arbitrary chosen to ensure that the system will keep the same behavior if its fitness is high. This parameter represents the adaptive faculty of the system. A high value allows the system to quickly jump into a new strategy. However, the drawback is that in some cases the current strategy, which is promising, will not be completely explored.

4.3 Results

Simulation results: since the simulation and experimental results proved to be very close, we'll mainly present the second ones with a real robot.

Experimental results: first of all, the convergence is quickly obtained. The system is quickly trapped into a local maxima. Figure 10.a. shows the evolution of the weights. After a few minutes (about 15 cycles) an acceptable strategy is found. The temperature suddenly decreases and traps the controller around this strategy. As the temperature is never equal to 0, it allows the weights to slowly slide into the best local solution. Figure 11. represents the influence of each sensor. Both values on each diagram represent the right and left command applied to the motors. There is no hidden layer in the network, and then the global behavior is a linear combination of each diagram. For example, the Figure 11.a. shows the current direction of the robot when no obstacle is detected: the robot is going straight. This figure shows that the reached strategy is not the global optimal one: on Figure 11.f. when an obstacle is detected on C_6 (see on Figure 2 the location of the sensors), the robot is going straight instead of turning on the right. In spite of this, the agent is able to avoid the obstacles, and to maximize its reward. The influence of C_7 (Figure 11.e.) compensates the lack of reactivity on C_6.

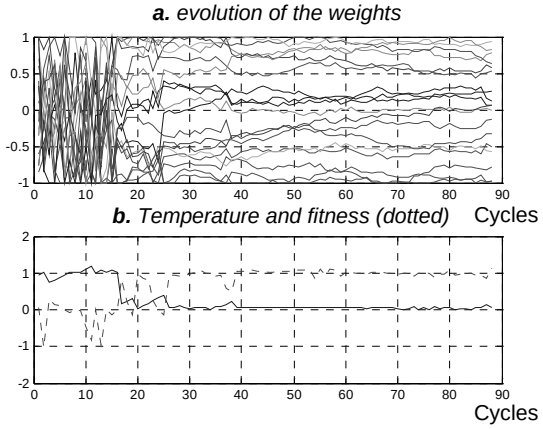

Figure 10: *results of an experiment*

At the 25th cycle (the average of all experiments is about 10 seconds), the controller is locked. 37 cycles after the

beginning of the experiment, we disabled the sensor C_1 by obstructing the receptor, to test the adaptability. Figure 10.b. represents the fitness and the temperature (dotted line), we discern the fitness peak. The new solution is very close from the previous one: the algorithm reinforces the influence of the closer sensors (C_0 and C_2) to compensate the lack of C_1, and the system quickly becomes stable again. More serious failures (a failure was simulated on many sensors) have been tested, and the system reenters in a new exploration of the state space as in the first cycles of the experiment. If the failure is too much important, the agent will not receive a sufficient reward. The temperature will not decrease, and the convergence will never be reached.

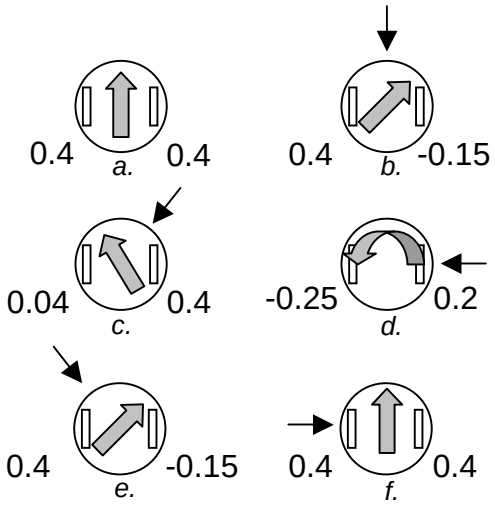

Figure 11: *Influence of each sensor on the global behavior. Black arrows indicates obstacles.*

5. Conclusion

Sections 3 and 4 have presented results of experiments using simulated annealing techniques to learn reactive behaviors. We have first experimented an algorithm to find the parameters of the neurocontroller. In safe circumstances, the method allows the agent to find the optimal solution, but the learning time is very long (one hour). Moreover disturbances on sensors and actuators, as well as the initial configuration of the robot, may prevent from finding the best parameters. This first algorithm is not well suited for our application; witch motivated the implementation of a second one able to adapt the controller to changes or failures. This algorithm does not guarantee to reach the optimal solution and may fail to adapt the controller if serious failures occurs, but it can quickly find an acceptable solution (10 minutes) and cope with some failures by adapting its own controller. We are currently working on the learning of new behaviors: target tracking, picking an object with an arm, docking a robot to a working station, etc. Once these neurocontrollers will be learned, we'll combine these behaviors to perform more complex tasks as foraging, or cooperative box pushing for example.

4. References

[1] N. Wiener, 1948, "Cybernetics, or control and communication in animals and machines", *Wiley*, New York.

[2] R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE Trans. on Robotics and Automation*, volume 2, 1986, pp. 14-23.

[3] D. Floreano, and F. Mondada, "Evolution of plastic neurocontrollers for situated agents", *Simulation of Adaptive Behaviors 4*, Brighton, 1996, Cambridge, MA, MIT Press.

[4] H-S Lin, J. Xiao, and Z. Michalewicz, "Evolutionary navigator for a mobile robot", *proc ICRA'94*, San Diego 1994, volume 3, pp. 2199-2204.

[5] J. Xiao, Z. Michalewicz, and L. Zhang, "Adaptive evolutionary planner/navigator for mobile robots", *IEEE Transactions on Evolutionary Computation*, volume 1, No. 1, 1997, pp.18-28.

[6] L. E. Parker, "Alliance: an architecture for fault tolerant multirobot cooperation", *IEEE Trans. on Robotics and Automation*, volume 14, No. 2, 1998, pp. 220-240.

[7] T. Balch, and R. Arkin, "Communication in reactive multiagent robotic systems", *Autonomous Robots*, volume 1, No. 1, 1994, pp. 27-52.

[8] O. Simonin, A. Liégeois, and P. Rongier, "An architecture for reactive cooperation of mobile distributed robots", *proc DARS-4*, Knoxville 2000, pp. 35-44.

[9] E. Yoshida, T. Arai, M. Yamamoto, and J. Ota, "Local communication of multiple mobile robots: design of optimal communication area for cooperative tasks", *Journal of Robotic Systems*, 15(7), 1998, pp. 407-419.

[10] P. Lucidarme, O. Simonin, and A. Liégeois, "Implementation and evaluation of a satisfaction/altruism-based architecture for multi-robot systems", *Proc Int. Conf. On Robotics and Automation 2002*, Washington D.C., pp. 1007-1012.

[11] P. Lucidarme, P. Rongier and A. Liégeois, "Implementation and evaluation of a reactive multi-robot system", *Proc. AIM'01*, Como 2001, pp. 165-170.

[12] D. Floreano, and F. Mondada, "Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot", *Simulation of Adaptive Behavior 3*, Brighton, 1994, pp. 421-430.