Evolutionary Computation of Multi-robot/agent Systems

Philippe Lucidarme
University of Angers, Engineering Systems Laboratory (LISA)
France

1. Introduction

Evolutionary computation (EC) and multi-agent systems (MAS) are two research topics offering interesting similarities. Both are biologically inspired; Evolutionary computation takes its origins from genetic algorithms inspired by the Darwinian principle (Goldberg 1989) and multi-agent systems are inspired from ethological studies, mainly ant colonies. Both are based on a population, made of individuals (EC) or agents (MAS). Evolutionary computation is based on the iterative development of a population. Each population is made up of individuals partially selected from the previous generations. In the case of multi-agent systems the population is made of agents. The exact definition of what is an agent is highly dependant on the context. In robotics systems, agents are usually described as a robot or part of a robot. In the first case, the multi-agent system (or multi-robots system) is composed of several robots interacting or acting together. In the second case, a unique robot is composed of several agents, each managing an entity (an actuator, a sensor …). In both cases, multi-agent systems offer a breaking down of a complex problem into several simpler tasks. The last (but not least) common point between evolutionary computation and multi-agent systems is the similarity between emergence and evolution; previous works (Brooks, 1986; Arkin 1992; Drogoul & Ferber, 1992) have shown that a population is able to perform tasks that an isolated agent is not able to do. A similar phenomenon appears in evolutionary computation: it's the combination of several individuals that allows the increase of fitness throughout the generations.

2. Related works

Multi-agent systems have been studied for many years and the following definitions used in the rest of the chapter are accepted by the MAS community and described in this section. All of them are fully detailed and justified in (Ferber 1999).

2.1 Distributed and supervised multi-agent systems

Multi-agent systems can be classified into two categories: supervised or distributed systems. In robotics, most of the system may be viewed as supervised: a central agent (the computer or processor) gathers information (from sensors) and sends commands to the other agents (the actuators). The main drawback of such architecture is that a failure on the central agent may be fatal to the whole system. On the contrary, in distributed systems, agents are as much as possible autonomous and the loss of agents may be supported by the system.

2.2 Communication between agents

Previous works (Drogoul 1992; Tucker 1994) have shown that communication is a key point in multi-agent systems. Such systems are classified in 3 categories: without communication, with implicit communication and with explicit communication. When no communication isn't used at all, the agents are fully independent. Such systems are generally limited in term of cooperation. When the agents are communicating without using classical data transmissions, it is called implicit communications. Famous examples are the pheromones in ant colonies. To find their way back or to inform the rest of the colony from danger, the ants use pheromones which act as messages. In robotics, some example may be found based on implicit communication. For example, at the European cup of robotics in 2006, the purpose of the game was to gather and sort coloured balls on a playground. A team of student built a robot composed of several independent modules. The communication between modules was done using the presence or not of balls inside the module. The simplicity of this implicit communication based system allowed a high reliability, essential in these kind of challenge. But most of the robots use explicit communications, for example bus protocols like I^{2}C, BUS CAN or TCP IP are used to communicate between the different parts of a robot and radio frequencies or infrared signals are used to communicate between robots.

2.3 Heterogeneity in multi-agent systems

Heterogeneity is used in multi-agent systems to evaluate the difference between agents. When all the agents are physically identical and controlled by the same rules, the system is said to be highly homogeneous. On the other hand, when each agent is specialized or dedicated to a task, the system is said to be highly heterogeneous.

2.4 Multi-agent systems and evolutionary computation

An evolutionary algorithm may be considered as a particular case of homogeneous multi-agent system where agents are individuals. The rules governing these agents are crossovers and mutations. Crossovers may be considered as a supervised rule: a central computer gathers the fitness and the artificial chromosomes of the whole population before computing the next generation. Mutations may be viewed as distributed rules because the mutation of a chromosome is not dependant from the rest of the population. Almost all the previous works, applying evolutionary computation to robots and generally inspired by the works of Dario Floreano (Floreano & Mondada 1994; Nolfi & Floreano 2000), were based on this principle.
This chapter will focus on the combination between EC and MAS to create controllers for autonomous robots. According to the previous observations, the experiments presented in this chapter take advantage of both systems. Agents are based on reactive controllers allowing fast responses and reducing computation time. The presented systems are fully distributed.
In the first part of the chapter, an experiment shows that a genetic algorithm can be fully distributed into a group of real mobile robots (Lucidarme 2004). The second experiment, made on a humanoid robot (HRP2), will describe how a unique robot can be seen as an evolutionary optimized multi-agent system.

3. Fully distributed evolutionary robots

3.1 Hypotheses

In this multi-robot experiment, each robot is considered as an agent. A homogeneous population is considered, i.e. all the robots have the same capabilities of sensing, acting, communicating and processing. The system is fully distributed, i.e. information about the agents are never centralized. Communications between agents are explicit. Moreover, the considered task is safe and robust reactive navigation in a clustered environment for exploration purposes. The robots are programmed at the beginning of the experiment neither for obstacle avoidance nor for enlarging the explored area, and nor for executing more complex actions. On the contrary, the agents have to find by themselves an efficient policy for performing the obstacle avoidance tasks. The on-line self-learning procedure is based upon the principles of evolutionary algorithms that allow a faster convergence rate than the classical learning algorithms, as it is shown in the following. The algorithm's operation is then illustrated by considering the exploration problem, which is simple to evaluate and to implement on real mobile robots. The population size is constant.

3.2 Evolution of physical robots

Type 1 (Lucidarme & Simonin 2006) is a small mobile robot (shown on figure 1) with a diameter of 13 cm and a weight of 800 g (including batteries) used in this first experiment. It has many of the characteristics required by the evolutionary approach to autonomous robot learning. The robot is fully autonomous; an embedded PC (80486 DX with 66 MHz clock) manages the robot. Control to sensors and actuators is transmitted by the PC104 bus. Two wheels actuate it. Two small passive ball-in-socket units ensure the stability. DC motors equipped with incremental encoders (352 pulses per wheel's revolution) control the wheels. The encoders are used for speed control but also to measure the performance index detailed in the next sections. The robot is surrounded with 16 infrared emitters and 8 receivers. The sensors use a carrier frequency of 40 kHz for a good noise rejection. These sensors are also used to communicate between agents. A communication protocol has been developed to differentiate obstacles from other robots: if a signal is received when the robot has stopped emitting for a short period, this means that another agent is close to it.

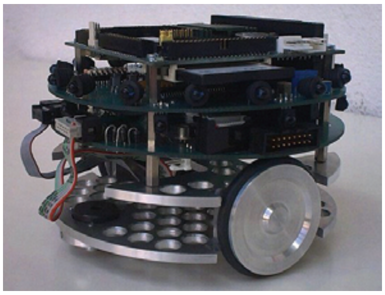


Figure 1. Picture of the mobile robot Type 1

3.3 Chromosome encoding

The controller of the robot is inspired from reinforcement learning (based on Markovian processes) where the policy is described by the association between states and actions. Such controller allows the comparison with other learning approaches. The inputs of the sensorimotor system are composed of five states listed in Table 1.

State 1	No obstacle
State 2	Left obstacle
State 3	Right obstacle
State 4	Front obstacle
State 5	Robot is jammed

Table 1. Different states of an agent's sensory system

These states can easily be recognized by the proximity sensors of any robot. Of course, more inputs would be available if fuzzy processing was applied. On the other hand of the control system are the elementary behaviors of the robots given in Table 2.

Behavior 1	Go forward
Behavior 2	Turn right
Behavior 3	Turn left
Behavior 4	Go backward

Table 2. The set of elementary actions

Actions actuate immediately the wheel motors. An individual of the population considered for the evolutionary learning is the list of connections between inputs and outputs. It encodes thus the "synapses" of the robot's sensorimotor control system. Such states and elementary actions have been chosen to guarantee that the behavior is learnable during the battery life. The chromosome of a robot is the concatenation (a string) of N words in a subspace of the {0,1}M space. N is the number of inputs, and M the number of outputs. Of course, each word contains a single bit '1' (Table 3). The population of robots will evolve following the evolution of the embedded chromosomes under the action of genetic operators.

0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0

Table 3. An example of a chromosome string

3.4 Operators and rules

Initialization: at the beginning of the experiment, the various chromosomal strings are filled at random or with the same behavior, like a string of "go ahead". For the simulations and experiments reported in this paper, the strings have been randomly generated to increase the diversity of the population.

Fitness: in many learning techniques, some kind of supervisor exhibits templates, and gives a rating to the agent's resulting behavior. In the case of genetics algorithms this upper level may also evaluate the fitness of each individual of the population with respect to a required performance. The individuals are then ranked (the selection operation) before applying the genetic operators. As we are looking here for a fully distributed evolution, a capability of local self-evaluation is given to each robot, but it is never conscious of the global population efficiency. However, following the general principles of self-learning algorithms, each individual (here the agent numbered i) computes its own current fitness using equation 1.

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

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 reward at time *N(i)*
$F_{N(i)}(i)$ is the instantaneous reward at time *N(i)*
In this obstacle avoidance problem, the instantaneous reward is computed as being the forward distance traveled during an elementary time step. This distance is computed using the motor's encoders. Thus, the current reward resulting from applying the current policy (the chromosome) is the average distance since initialization or since the last change of the agent's chromosome by crossover or mutation. Such a computation automatically penalizes too numerous turns and reverse motions.
Crossover: most of the solutions proposed by the others investigators call for global communication between the agents. In this way, the classical genetic algorithm technique is used: selection by considering the whole population, then crossover of two chromosomes at a periodic average rate. This method suffers from the lack of parallelism since only two agents can be concerned at the same time. It even may require a slackening of the robot moves to allow the complex communication, agent recognition and signal processing. In order to avoid these major drawbacks, the solution proposed here uses a local and simple communication. This way, any pair of robots meeting each other may perform crossover. The formal conditions are the following: the robot-to-robot distance is short enough to communicate and both robots have not recently performed a crossover or mutation operation. The first condition ensures possible parallelism, while the second prevents any robot from changing its current policy before having evaluated it over a significant number of steps. When crossovers are complete, agents i and j have new chromosomes, i.e. policies, according to the probabilities given by equation 2.

$$P(i) = \frac{R_{N(i)}(i)}{R_{N(i)}(i) + R_{N(j)}(j)} \tag{2}$$

Crossovers are not sufficient to ensure the convergence of the system towards the best solution, especially when the population size is small; the optimal chromosome may not be dispatched in the initial population preventing crossovers from creating this optimal chromosome string. In that case, the agents may be trapped by a local maximum, and the population no longer evolves from that common state. As usual, mutations are necessary to escape from local extrema and to explore a wide domain of the chromosomal state space.
Mutation: in the classical genetic algorithms, mutations are performed at random. In our application, this could be an important drawback since we are looking for on-line learning. It cannot be admitted that a robot changes its policy to a far less efficient one and waits for a long time before re-improving it by a new mutation or by crossover. To solve these problems, the formal conditions for mutation are adopted: the agent has not performed a previous mutation recently and the agent's fitness is low. Notice that again such a method allows multiple simultaneous mutations of several agents. As for crossover, the first condition ensures that the robot has had enough time for computing a good estimation of its

own fitness. The second condition prevents from changing good policies into worse ones. The adopted probability of mutation is dependent on the individual performance index. It ensures that the worst policies are more likely to mutate, while the best ones possess a non-zero probability for escaping a possible local maximum.

3.5 Simulation results

The algorithm was first evaluated via Matlab allowing simulations with a large number of robots. The moving agents never compute their absolute position and orientation. The knowledge of these variables only serves for the displays. On the contrary, the real sensory system is simulated and provides the agent with its local perception. The real wheel control is also simulated. An example of display is shown on Figure 2.

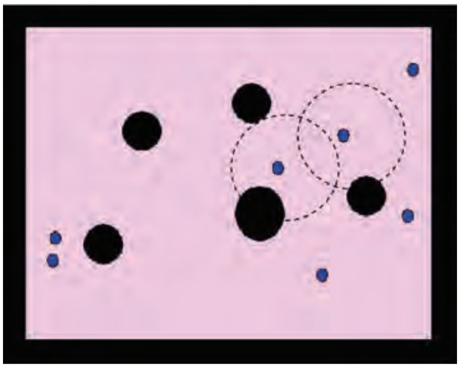


Figure 2. Snapshoot of the simulator with 7 robots (small circles) and circular obstacles (in black)

During a simulated elementary sensorimotor cycle of time, each agent updates its current state, and either continues its current policy or performs mutation (if allowed) or crossover (if possible). To make crossover possible, the sensory protocol differentiates another agent from a passive obstacle, and is able to compute the robot-to-robot distance. The two dashed circles surrounding two robots in Figure 2 show the sensor ranges. When a crossover is allowed, it takes several cycle times due to the need for communicating information. Otherwise, each of the two robots considers the other as an obstacle.

The first step is to set the simulation parameters. The communication range is measured on real robots and scaled to the simulated environment. Then it is necessary to adjust the coefficients in the probability function for mutations. The answer was found thanks to simulations. If the delay is short, the mutation occurs frequently. The search state is explored very quickly, but the performance estimation is erroneous. On the contrary, if the delay is long, the space state is explored very slowly, but the estimation is very good. The delay between two crossovers was also studied. The conclusions are close to those of mutations. If the delay is too short, robots will always crossover with the same agent. On the contrary, if the delay is too long, it takes a lot of time to explore the policy space state. The minimal number of cycles is chosen to fulfill the following condition: two agents having the optimal policy will not meet twice successively. Here about 300 cycles are necessary.

Figure 3 shows the end of a simulation. The first observation is the explored part of the environment (in white). It results from the obtained emergent behavior due to the reactivity of the multi-agent system. We have also checked that the optimal solution, given in Table 3, is always reached whatever the initial conditions and the shape of the environment.

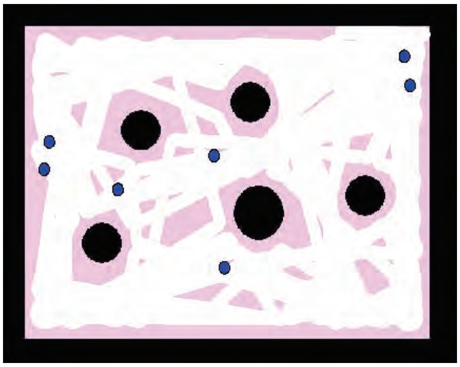


Figure 3. Snapshoot of the simulator at the end of an experiment

At the end of the experiment, the chromosome encodes the optimal sequence (table 3).

1: No obstacle	1: Go forward
2: Left obstacle	2: Turn right
3: Right obstacle	3: Turn left
4: Front obstacle	2 or 3 : Turn left or right
5: Robot is jammed	4: Go backward

Table 3. Different states of an agent's sensory system

In most cases, the observed evolution is the following:

1. At the beginning, the agents try random strategies. A collision quickly occurs, and the robot is jammed. The average traveled distance drops very fast, and a mutation occurs soon. Mutations stop when the behavior 4 (go backward) is associated to the state 5 (robot is jammed).
2. The agents are free in the environment, and one of them is likely to associate the first behavior (go forward) to the first state (no obstacle). It generally travels a long distance in the environment and propagates its chromosomal string to other agents by crossover.
3. As many agents move fluently in the environment, more crossovers occur. The optimal policy is thus given to at least one agent. Its own fitness grows quickly.
4. As soon as such a robot performs the best sequence, it travels all over the environment and meets more agents than the other ones. By doing so, it transmits its string to many robots and all the population quickly performs the best behavior. The non-null mutation rate imposed in "good" populations ensures that the absolute maximum is obtained.

Another important parameter is the size of the population, i.e. the number of robots. Intuitively, the more agents there are, the faster the best solution is found. This hypothesis is true, if the condition of convergence is: "at least one agent finds the optimal sequence". We decided to stop the simulations when each individual had performed the chromosomal sequence described in table 3. For each number of robots, twenty simulations were performed and the average number of cycles was recorded. The results are shown on Figure 4.

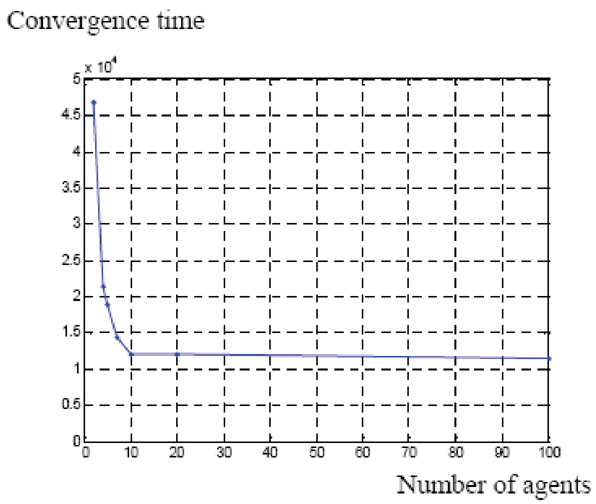


Figure 4. Convergence time versus number of robots

It can be seen on figure 4 that, when less than ten robots are used, the average convergence time decreases with the number of robots as in classical evolutionary algorithms. With more than 10 robots, the convergence time becomes constant. The explanation is that when the number of agents is high, the best behavior is quickly found (because the search space is quickly explored). But the time to propagate this optimal chromosomal string into the whole population is longer when the population is big, this phenomena counterbalances the fast exploration of the state space.

3.6 Experimental results

The experimental environment is shown on Figure 5. It is 4.80 m long and 3.60 m wide. Four autonomous robots are used in the experiments, and obstacles can be moved, added or suppressed. The maximum speed of the robots is 1 m/s but the speed has been limited to .3 m/s to prevent eventually violent collisions. The sensorimotor cycle time is about 15 ms. The range of the infrared system is typically of .5 m: it ensures here the required compromise between the needs for enough crossovers and for safety with respect to collisions. Following these numerical data and the experience gained during the simulations, the minimum time between two successive crossovers has been set to 3 seconds.

The first conclusion of this real experiment is the fact that the algorithm always converge to one of the two optimal solutions in a reasonable time; less than ten minutes for the simple obstacle avoidance task (5 states and 4 actions). This real experiment has also confirmed the emergence of an important phenomenon: when a robot performs an efficient strategy, it naturally increases its chance to be in the next generation i.e. it travels smoothly in the environment and increases its chance to meet the others robots; he becomes a more popular candidate for reproduction.

Figure 5. Distributed evolution of real robots

3.7 Conclusion
This first experiment has proved the possibility of transposing evolutionary computation into a real group of robots. Experimental results have shown that this solution takes effectively advantages (fault tolerance and emergence) from multi-agent systems as explained in the introduction. As the system is fully distributed it becomes fault tolerant; for example, if a robot is jammed or broken, the rest of the population still works. During the experiments, robots has been removed and added into the population without any trouble. Having a group of robots decreases the learning duration because the search space is more quickly explored compared to an equivalent experiment made on a unique robot (Floreano & Mondada 1994). But having a population of robots may also have drawbacks; the size of the environment must be proportional to the number of robots. Simulation results have shown that, over a threshold, increasing the population size doesn't decrease the convergence time in regards with the size of the environment. Using at the same time several complex or expensive robots is not always possible; i.e. using ten humanoid robots at the same time to process learning is not actually feasible. The next section has been motivated by this conclusion and will describe an experiment where a unique robot is seen as an evolutionary optimized multi-agent system.

4. Evolutionary optimized multi-agent system

4.1 Hypotheses
As explained previously, a unique robot may be structured as a multi-agent system where each part of the robot is a communicating and acting agent. The task considered here is the tracking of a target with the hand of a humanoid robot without falling or being unbalanced. We assume that the target is reachable by the robot without walking, i.e. feet are considered as linked on the floor and the target is included in the workspace of the body. The position of the target is assumed to be known by the robot; it may be done by using the vision system of the robot for example. The body of the robot is decomposed in agents each controlling a joint. The system is fully distributed, i.e. information about the agents are not centralized. Of course existing robots use a central processor and have not been designed to support such architecture. Currently, the distribution is simulated by sharing the central processor, but it may be possible to embed a controller in each joint to perform the agent behavior. Moreover, it is actually done on existing robots (Murase and al. 2001) for controlling the

joints with a PID. To allow this distribution of the system, each agent only communicates with its neighbors.
A homogeneous population is considered, however, as each agent controls a joint of the humanoid robot, the computed model and parameters varie. As the robot had a human shape, it may be a plus to provide natural (human-like) motions. An evolutionary strategy is used to optimize the parameters of the system. The purpose of the presented experiments is to show that after the evolution, the system automatically manages redundancy and failures on actuators.

4.2 Description of the architecture

The robot considered in the following is HRP2. This humanoid robot has 30 degrees of freedom (6 by leg, 6 by arm, 1 by hand, 2 for the hip and 2 for the neck). This robot has been chosen for its high redundancy, even if many other robots may have been used.

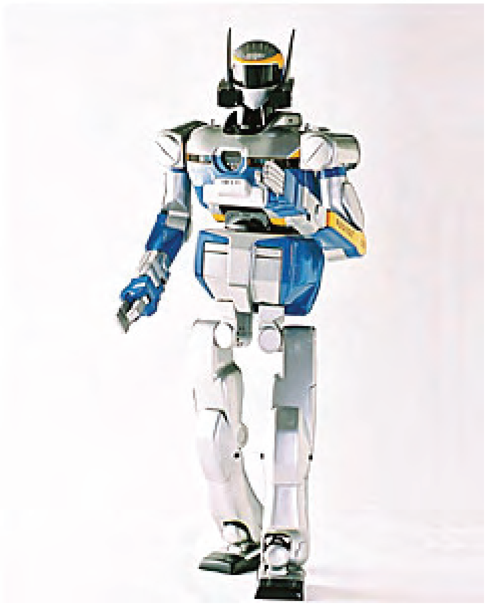


Figure 6. The humanoid robot HRP2

In the proposed distributed approach each actuator q_i is respectively associated with the agent A_i. Each joint or agent acts independently from the other joints in order to minimize the Euclidian distance ε between the hand and the target, i.e. reaching the target with the hand. Each agent is described by the following 3 items : input, output and behavior .
Input : information or data, to which the agent can access. The agent A_i knows the following variables:

- ${}^{0}T_{i-1}$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_{i-1}. This information is communicated by the agent A_{i-1} except for the first agent.

- ${}^{i}T_n$: transformation matrix linking the joint q_{i+1} to the end effector. This matrix is communicated by the agent A_{i+1} except for the last agent.
- q_i : current measured position of the joint controlled by the agent A_i. This position if provided by a sensor on the joint (in the case of HRP2, an encoder provides the position of the joint)
- P_{Target} : coordinate of the target in the frame F_0. We assume that this information is known by each agent. On a real physically distributed system, this information may be computed by a dedicated controller (based in the head of the robot for example) and transmitted to the closest agent. This agent transmits this information to the neighbors and the coordinates are propagated in the entire system.

Output: each agent A_i must provide two kinds of outputs: command on the actuator and information to the neighbors:

- Δq_i : command applied on the joint (angular speed).
- ${}^{0}T_i$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_i. This information is communicated to the agent A_{i+1}.
- ${}^{i-1}T_n$: transformation matrix linking the joint q_i to the end effector. This matrix is communicated to the agent A_{i-1}.
- P_{Target} : coordinate of the target transmitted to the agents A_{i-1} , A_{i+1} or both.

A general view of the architecture, showing the information exchanged between the agents is shown on figure 7.

Behavior: this is the way the agents link the input to the outputs. Note that, as the system is distributed, the behaviors are local, i.e. the agent *Ai* does not know the way the other agents will act. The global goal is to reach the target with the end-effector of the robot. Internal collisions are not considered here. The behavior of the agent *Ai* consists of computing at each time step the value Δq_i that minimizes the Euclidian distance between the position of the end effector *Pn* and the position of the target P_{target} in regard to the information known by the agent. Due to its efficiency and its simplicity, the gradient descent technique has been chosen. This choice was motivated by the following reasons:

- The mathematical relationship between the actuator's position and the distance to minimize is a known function. The derivative of this function may be easily computed.
- There are few parameters to set, just one for each agent and evolutionary computation is well suited for tuning these parameters.
- The main advantage is probably that the algorithm slides into the minima. It does not provide just the final solution, but provides a trajectory that slide from initial to final configuration. In the case of robots, we need more than the final position: we need a trajectory. Gradient descent provides both.

Each agent updates its output according to equations 3

$$\Delta q_i(t) = -\alpha_i . \frac{d\varepsilon}{dq_i} \tag{3}$$

ε is the Euclidian distance between the hand of the robot and the target
q_i is the current position of the joint *i*

α_i is a parameter that determine the behavior of the agent A_i . The influence of this parameter is described in the next section.
The derivative of the Euclidian distance is computed using the equation 4.

$$\frac{d\varepsilon}{dq_i} = \frac{(x_{Target} - x_n).\frac{dx_n}{dq_i} + (y_{Target} - y_n).\frac{dy_n}{dq_i} + (z_{Target} - z_n).\frac{dz_n}{dq_i}}{\sqrt{(x_n - x_{Target})^2 + (y_n - y_{Target})^2 + (z_n - z_{Target})^2}} \quad (4)$$

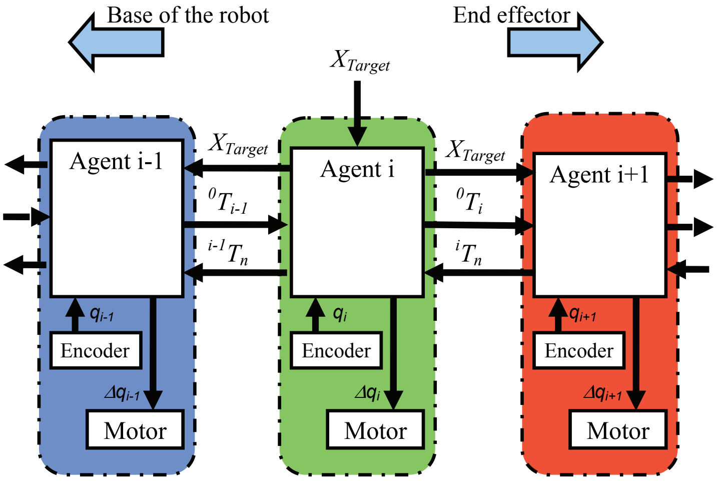


Figure 7. Overview of the multi-agent architecture

Note that, as the system is fully distributed, the agent A_i cannot compute $\frac{dx_n}{dq_i}$, $\frac{dy_n}{dq_i}$ and $\frac{dz_n}{dq_i}$ by using the jacobian because according to our hypothesis, this agent A_i cannot access to the other agent's dq. In spite of this, the matrix product described on equation 5 can compute the derivative of the end-effector position according to the joint i.

$$\frac{dP_n}{dq_i} = {}^0T_{i-1}.\frac{d^i-^1T_i}{dq_i}.{}^iT_n.X_0 \quad (5)$$

This relationship described on equation 4 is only true in the case of serial robots. Only one transformation matrix ($^{i-1}T_i$) is expressed in term of q_i, t. The following terms can be considered as scalar and don't need to be derivated: $dX_n\ dq_i$, $^0T_{i-1}$ and ^iT_n.The behavior of the agent Ai is described in the algorithm presented on the figure 8.

```
Algorithm **AgentBehavior**

input : $^{O}T_{i-1}$, $^{i}T_n$, $P_{Target}$, $q_i$
output : $\Delta q_i$, $^{i-1}T_n$, $^{O}T_i$

At each time step :
        Compute : $^{i-1}T_i$ and $\frac{d\,^{i-1}T_i}{dq_i}$
        Compute : $\frac{dP_n}{dq_i}$ and $\frac{d\varepsilon}{dq_i}$

        Update : $^{O}T_i = {}^{0}T_{i-1} \cdot {}^{i-1}T_i$
        Update : $^{i-1}T_n = {}^{i-1}T_i \cdot {}^{i}T_n$
        Update : $\Delta qi = -\alpha_i \cdot \frac{d\alpha\varepsilon}{dq_i}$
```

Figure 8. Algorithm of the agent A_i

4.3 Influence of the parameters α_i

The behavior of each agent is dependant on its α_i coefficient. To explain the influence of these coefficient, let's take as example the 3R planar robot described on figure 9. Such robot is simple to compute, its behavior is easy to understand and it evolves in two dimensions. It's a redundant robot, *i.e.* several configurations may exist for a given position of the end effector (Note that, in the case of a 3R robot, it is no longer true for position and orientation). The figure 10 shows the motion of the robot for different settings of the parameters α_i. In the first example (left figure), the agent A_3 (closer to the end effector) had the highest value, the second one A_2 had a medium value, and the first one A_1, the smallest; the robot mainly uses its last link to reach the target. In the second example (central figure), the agents A_1 and A_2 are set with the same coefficient and in the last example (right figure), each agent is set with the same parameters. The global trajectory is far from the optimal solution: the robot moves its end-effector away from the target to come back closer. As shows figure 10, the global behavior of the robot is highly dependant on the parameters.

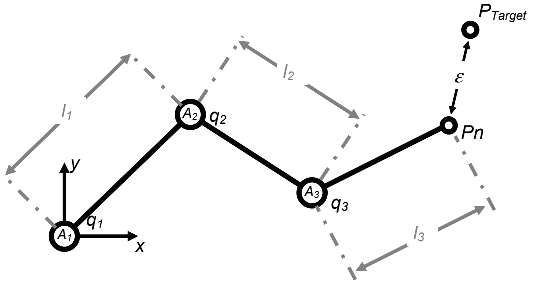
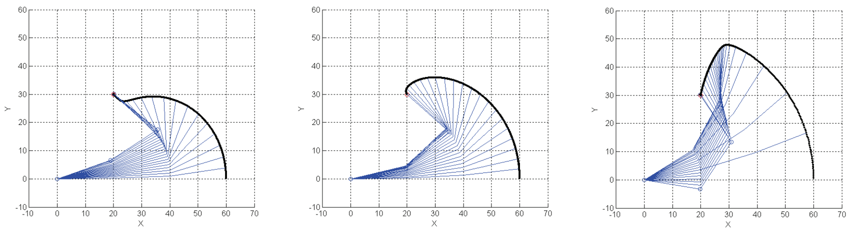


Figure 9. 3R planar robot



α_1=2.10^{-5} α_2=5.10^{-5} α_3=2.10^{-4} α_1=2.10^{-5} α_2=5.10^{-5} α_3=2.10^{-4} α_1=2.10^{-5} α_2=5.10^{-5} α_3=2.10^{-4}

Figure 10. Example of trajectories for several sets of parameters

4.4 Stability

In order to apply the algorithm to a humanoid robot, the control of the static stability has been added. This is based on the same principal that for target tracking, rather than minimizing the distance between the hand and the target, the goal is to minimize the distance between the projection on the floor of the center of mass (COM) and the center of the footprint. To deal with the two objectives at the same time, the formula presented on the equation 6 is used.

$$\Delta q_i = -\gamma.\alpha_i^{Target}.\frac{d\varepsilon_{Target}}{dq_i} - (1-\gamma).\alpha_i^{COM}.\frac{d\varepsilon_{COM}}{dq_i} \tag{6}$$

Where :

$$\gamma = \frac{D_2}{D_1 + D_2}$$

ε_{COM} is the distance between the projection of the center of mass and the center of the footprint

D_2 is the distance between the projection of the center of mass and the center of the footprint

D_1 is the distance between the projection of the center of mass and the closest edge of the footprint

These equations ensure that when the robot is perfectly stable (γ=*1*) the algorithm will control only target tracking. On the other hand, when the projection of the center of mass is on the edge of the footprint (ie. The robot is at the limit of stability and γ=*0*) hundred percent of the command is dedicated to stability. Between these two extremes, the ratio is linear. This algorithm had been implemented on the model of HRP2. The first simulations have immediately shown the efficiency of the method and the emergence of behavior. For example, the robot dedicates its right arm only for stability because the right arm is not useful for target tracking. One actual drawback of this technique is that the designer has twice as many parameters to set. One set of parameters for target tracking, and one set of parameters for stability. Experiments have shown that the parameters can be set arbitrary, likely due to redundancy. But finding a set of parameters that provide an optimal and/or human-like motion is not trivial. This is the reason why evolutionary computation has been used to evolve these parameters.

4.5 Evolutionary computation of the parameters

Evolutionary algorithm has been chosen for the following reasons: this algorithm doesn’t need a model of the system, it provides several good solutions, it can optimize any kind of criteria and it can extract from local minima.

Chromosomal encoding: our goal is to optimize the parameters $\alpha_i^{T\arg et}$ and α_i^{COM}. These parameters are directly encoded in the chromosomal string. The chromosomal string of an individual is composed of 44 float numbers in the case of HRP2: 22 for the target tracking objective, plus 22 for the stability. Twenty two is the number of joints minus six for the right leg (in order to ensure the feet are not moving on the floor; the right and left legs have been linked, i.e. each joint of the right leg is performing the same motion than the left leg joints) and two for the head. (the head joints are not considered here; we assumed that on the real robots these joints must stay available for the motion of the embedded vision system).

$\alpha_1^{T\arg et}$	$\alpha_2^{T\arg et}$		$\cdots$		$\alpha_{22}^{T\arg et}$	$\alpha_1^{T\arg et}$	$\alpha_2^{T\arg et}$		$\cdots$		$\alpha_{22}^{T\arg et}$

Table 4. Description of a chromosomal string

Initialization: at the beginning of the experiment, the various chromosomal strings are randomly filled in order to increase the diversity of the population. The range of the parameter is [-1 ; +1]. There is no need to extend this range; the important fact is the difference between the parameters, not their values. For example, by setting all the parameters to 1 the robot will have exactly the same behavior than by setting all the parameters to 0.1. The only difference is the speed of the motion. By weighing all the parameters with a coefficient K_s it becomes possible to control the speed of the motion. Having negative values is important to push the robot back from unstable positions.

Fitness: the objective of the presented experiment is to have a human-like behavior in order to make the robot friendly with the people using it. Such behavior is not easy to transpose into equations which is why three fitness functions are considered: minimizing times during initial and final position of the hand, minimizing the traveled distance of the hand, and minimizing the energy according to equation 7. The behavior of the robot for each fitness function is discussed in the next section. For each individual, the fitness function was initially evaluated on 7 randomly selected targets. The obtained behaviors can’t be generalized at any point of the workspace. The number of targets has been extended to 18 points strategically positioned into the workspace as shown on figure 11.

$$F = \sum_{t=0}^{t=T} \sum_{i=1}^{n} \frac{1}{2} m_i . v_i^2(t) \tag{7}$$

Where :

F is the fitness function

T is the duration of the motion in time steps

n is the number of body composing the robot

m_i is the mass of the body i

$v_i(t)$ is the speed of the body i at time step t

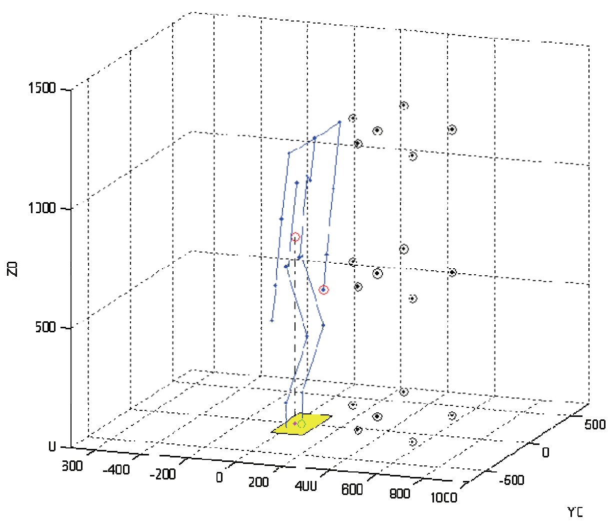


Figure 11. Position of the 18 targets during the evolutionary process

Crossovers: at the end of the evaluation of the population, the new generation is automatically generated. For each new individual, two chromosomal strings from the previous generation are selected using the roulette-wheel reproduction as described in (Nolfi & Floreano 2000). The new chromosomal string is generated by a random selection of the coefficients. For each new coefficient a random selection is made between the two same values in the two "parents". This strategy is based on the principle of uniform crossovers (Mitchell, 1997).

Mutations: after the crossover process, ten percent of the population is randomly selected to mutate. Such percentage has been arbitrary chosen to ensure the equilibrium between a good exploration of the search space and the preservation of the best known individuals. A mutation consists of the selection of a coefficient α_i^x where *i* and *x* are randomly chosen. This coefficient α_i^x is then replaced by a new one, randomly selected in the range [-1; 1].

4.6 Experimental results

Experiments have been conducted by simulation using the geometric model of the robot HRP2, including the center of mass of each body allowing the computation of the global center of gravity of the whole robot. Each joint is bounded in the same interval as on the real robot. The simulator is of course based on a centralized processor that simulates at each time step the behavior of each agent. This allows the simulation of the distributed system. The figure 12 shows the evolution of the fitness function in the first experiment where the purpose is to minimize the necessary time to reach the target from the initial position. Similar graphics have been produced for the other fitness functions. In the three cases, a convergence is observed that confirms the relevance of the chosen parameters.

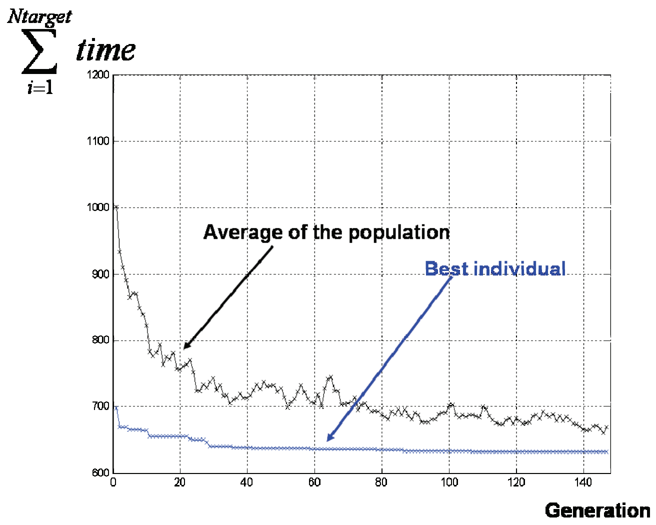


Figure 12. Evolution of the fitness function (time) for the whole population (average) and for the best individual of each generation

Time and distance: the fitness functions minimizing time and distance produce similar behavior on the robot. In both cases, the robot is able to reach the targets in any configuration. As the robot has been programmed to reach targets with the left hand, the right one has no influence in the target tracking. The coefficients have only been optimized by the evolutionary process to keep the stability of the robot. For example on figure 13, when the robot leans forward it puts back its right arm to compensate the move of the center of gravity. However, depending on the position of the target, the general motion of the robot is sometimes far from natural or human-like behavior. These observations motivated the implementation of a new fitness function consisting of minimizing the energy.

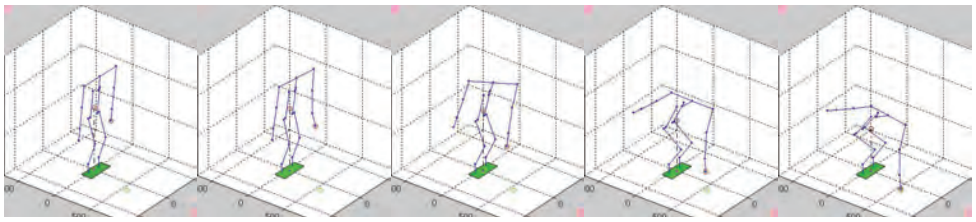


Figure 13. Example of motion of the robot

Energy: minimizing energy provides interesting results. First of all, the targets are still reached at any position of the workspace. The behavior of the robot is very smooth and natural, except for the targets over the shoulders. The robot moves its torso backward and then moves forward. This behavior is not really natural, because a human has the faculty of anticipating the end of the motion. However, in the present case a human would probably move one step backward in order to hold out its arm. Such behavior is not possible here because of the hypothesis that feet are linked to the floor.

Failures: in order to test the robustness of the system failures have been simulated. The first test was a failure on an actuator. As the agents are independent, the system continues working; the other agents naturally compensate the inactivity on a joint. The second test simulated failures on several joints. When too many actuators are inactive, the system is not always able to reach the target, sometimes even if it's always theoretically possible. These experiments have shown that the presented architecture is mainly intended for redundant robots. An observed conclusion is that the more the system is redundant, the more it can extract from local minima, mainly due to the fact that minima are not concerning the whole population. When an agent is attracted to a minima, the others, when moving, drag him from it, in the same way that a failure on an agent is compensated by the other agents.
Computation time: measurement of the computation time is now addressed. Note that the measurement has been made under Matalb, so we may expect better results with a compiled software. The computation for the whole robot requires 0.68ms per cycle. Half of this time is needed to compute the kinematic model, and about 44% for the computation of the derivative, 3% for the computation of the distances and 3% is needed to apply the command on the actuators. On a real distributed architecture, these delays may be divided by the number of agents.

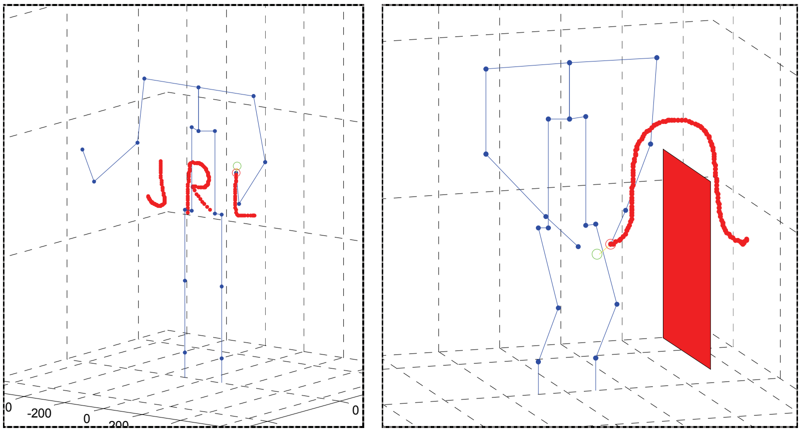


Figure 14. Example of trajectory followed by the hand of the robot (left), and simulation of obstacle avoidance (right).

Trajectory: improvements have been performed on the agent's behavior in order to perform trajectory following. The final goal of these improvements is to perform obstacle avoidance with external obstacles. The designed architecture is not described here, but the experiment has shown that following a trajectory with the end effector is possible as illustrated on figure 14. Controlling the speed of the end effectors is also possible, but the speed controller is currently centralized and the advantages of the distributed system are lost.
The results have been partially confirmed by the implementation of the algorithm on the OpenHRP platform where the dynamic model of the robot is fully implemented. Unfortunately, as the internal collision detection is not yet operational, an implementation on the real robot is currently too risky.

5. Conclusion and future works

This chapter deals with the combination between multi-agent systems and evolutionary computation. The first section describes an experiment of the evolutionary learning of an autonomous obstacle avoidance behavior. This first experiment proves the possibility of distributing a genetic algorithm into a real robot population. In the proposed architecture, new crossovers and mutation techniques have been proposed allowing the distribution of the algorithm.

The second part of the chapter deals with the decomposition of a unique robot into a multi-agent system. In the distributed proposed architecture, the behavior of the robot is dependant from a set of coefficients influencing the motion of each agent. These coefficients are evolutionary optimized. Several fitness functions have been experimented: time, distance and energy. Simulation results show that minimizing the energy is the best strategy, the system may be fault tolerant and the computation of the algorithm is very fast.

Future works will be oriented on a survey on the behavior of the robot around singular configurations and the implementation of a collision avoidance module, preventing the robot from hurting himself. So as to keep the advantages of the presented architecture, a nice strategy would be to distribute the obstacle avoidance module. This implementation looks like being really challenging, because it will highly increase the communication between the agents. Once the collision avoidance is functional, the implementation on a real robot will be planned.

6. Acknowledgements

This work has been supported in part by the Robotics, Microelectronics and Computer Science of Montpellier (LIRMM, Montpellier, France), the Joint Robotics Laboratory (JRL-AIST, Tsukuba, Japan) and by a Japanese Society for the Promotion of Science (JSPS) grant.

7. References

Arkin R.C. (1992). Cooperation without communication: multi-agent schema-based robot navigation, *Journal of Robotic Systems*, Vol 9, No.3, (April 1992), pp 351-364.

Brooks R.A. (1986). A robust layered control system for a mobile robot, *IEEE Transaction on Robotics and Automation*, Vol. 2, pp.14-23.

Drogoul A. & Ferber J. (1992). From Tom Thumb to the Dockers: some experiments with foraging robots, *Proceedings of the 2nd International Conference on Simulation of Adaptative Behavior*, pp. 451-459, Honolulu, 1992.

Ferber J. (1999). *Multi-agent systems. An introduction to distributed artificial intelligence*, Addison Wesley, London

Floreano D. & Mondada F. (1994). Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot. *From Animals to Animats 3: proceedings of Third Conference on Simulation of Adaptive Behavior*. Cambridge, MA: MIT Press/Bradford Books.

Goldberg D.E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, reading, Mass, 1989.

Lucidarme P. (2004). Anevolutionnary algorithm for multi-robot unsupervised learning, *Proceedings of the IEEE Congress on Evolutionnary Computation*, pp. 2210-2215, Oregon, Portland.

Lucidarme P. & Simonin O. (2006). Le robot mobile Type 1. *Journée des démonstrateurs en automatique*, Angers, France

Mitchell. T. (1997). *Machine learning*, McGraw Hill, ISBN 0070428077.

Murase Y. Yasukawa Y. Sakai K. Ueki M. (2001). Design of a Compact Humanoid Robot as a Platform. *Nippon Robotto Gakkai Gakujutsu Koenkai Yokoshu*, Vol 19, pp. 789-790.

Nolfi S. and Floreano D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-organizing Machines*, Bradford book, MIT Press, Cambridge, Massachusetts.

Tucker B. & Arkin R.C. (1994), Communication in reactive multiagent robotic systems. *Automonous robots*, Volume 1, No.1, pp. 27-52.