Evolving Cooperation of Simple Agents for the Control of an Autonomous Robot

Ricardo A. Téllez and Cecilio Angulo

Knowledge Engineering Research Group (GREC)
Technical University of Catalonia
Rambla de l'Exposició s/n
08800 Vilanova i la Geltrú. Spain
r_tellez@ouroboros.org, cecilio.angulo@upc.es

Abstract: A distributed and scalable architecture for the control of an autonomous robot is presented in this work. In our proposal a whole robotic agent is divided into sub-agents. Every sub-agent is coded into a very simple neural network, and controls one sensor/actuator element of the robot. Sub-agents learn by evolution how to handle their sensor/actuator and how to cooperate with the rest of sub-agents. Emergence of behaviors happens when the co-evolution of several sub-agents embodied into the single robotic agent is produced. It will be demonstrated that the proposed distributed controller learns faster and better than a neuro-evolved central controller.

Keywords: autonomous robots, agents, distributed controller, evolution, neural networks

1. INTRODUCTION

At present, the design of autonomous robots can be faced in several ways being evolutive robotics a promising one (Nolfi and Floreano, 2000). Using evolutive robotics, Floreano and others (Floreano and Mondada, 1996; Floreano and Urzelai, 1997) demonstrated that a behavior based robot could be evolved by a genetic learning algorithm just specifying a simple fitness function, without direct encoding of the required behaviors for the task at hands. By using a centralized controller, implemented by a neural network, their real robot was able to move around a circuit avoiding obstacles and finding *food* resources. Avoiding obstacles, timing and looking for food behaviors emerged from the maximization of a fitness function, but those were never explicitly coded into the robot control law. A similar work is reported by Moriarty and Miikkulainen (1996), where the ESP evolutionary algorithm is used to evolve the controller of a robot arm, and by Steels (1994) who showed emergent functionality through on-line evolution.

Later, Han Yong and Miikkulainen (2001) showed that different cooperative behaviors between several agents could appear in a predator-prey domain when a distributed controller is evolved in a cooperative way. They evolved a group of agents, each one with its own controller, making them learn how to reach the prey, resulting in a distributed controller (as opposite to a centralized controller). Results confirmed that the distributed version is more powerful and learns faster and better than the centralized one.

Taking all those results as point of departure, and going in the direction suggested by Minsky (1988) about the society of mind, we are concerned to obtain a cooperative controller for an autonomous robot, by designing a robot composed by several general control units or information processing units called *Intelligent Hardware Elements*. We will create a group of sub-agents inside a single global agent and they will learn how to cooperate to correctly control the autonomous robot.

Each sub-agent will be represented by an Intelligent Hardware Unit. These elements embody a sensor or actuator and a small processing unit, acting the whole as a sub-agent inside any control system, learning to cooperate with other sub-agents to achieve a common goal. In this paper we show how an autonomous cooperative control system for a robot is possible using such normalized elements, and how its performance improves that of a centralized controller.

For the evolution of the controllers it will be used the co-evolution algorithm named Enforced SubPopulations (ESP (Gómez and Miikkulainen, 1997; Gómez and Mikkulainen, 1999)). Any other neuro-evolution algorithm could be used to obtain the controllers, however, in practice, ESP has specifically demonstrated be efficient when evolving such kind of controllers (Gómez and Mikkulainen, 1999).

This paper is distributed as follows: section 2 describes in a precise way the proposed architecture. In section 3, some experiments with centralized and distributed controllers are evaluated and analyzed. Discussion of the results is provided at section 4, and finally conclusions are displayed at section 5.

2. GENERAL METHOD

The main purpose of this work is to show how a distributed or cooperative controller for an autonomous

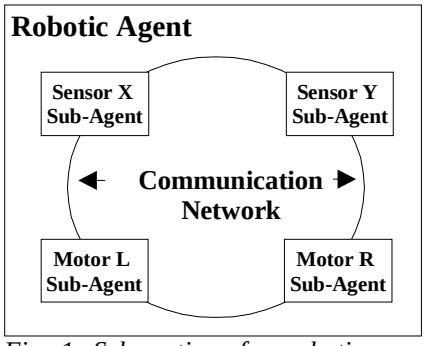

Fig. 1. Schematics of a robotic agent composed of four sub-agents, when a distributed controller is used.

robot is possible in order to accomplish specific tasks, and, moreover, that this controller performs better than a centralized one. The centralized controller is composed by only one artificial neural network with all the required inputs and outputs to control all sensors and actuators. The distributed controller will consists of several artificial neural networks, one for each sensor/actuator, interconnected through a communication network (Fig. 1).

2.1.The Intelligent Hardware Unit

In order to design the cooperative controller we are concerned about a general purpose hardware control element able to sense or act into the environment, and to process information in a simple way. We call this unit an intelligent hardware unit (Fig. 2). The unit is composed of two different parts: the first one is a sensor or actuator hardware element, depending on the task of the unit, to sense the environment or to act on it. The second part is the processing element.

The sensor/actuator part can be any type of sensor/actuator hardware element and its output/input is directly connected to the processing unit. Its role is to interface with the real world sensing something from it or actuating on it. The processing part of the unit is a simple element of calculus that interfaces with the sensor/actuator and processes its output/input in a simple way. It also has the ability to connect with the other processing units. At practice, it could be any type of control algorithm implemented in a simple micro-controller. The processing algorithm could be a neural network, a data processing algorithm or anything similar. Its main role is to process the data coming from the sensor or going to the actuator in order to obtain the desired *global* behavior. A second role of the processing unit is to interface with other intelligent hardware units. To do this, the element is also capable of sending and

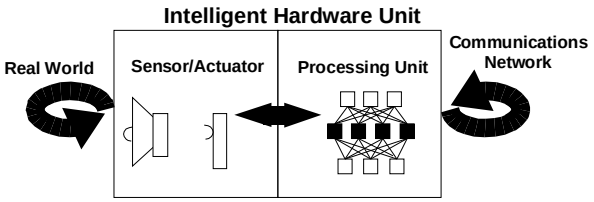

Fig. 2. Schematics of the Intelligent Hardware Unit

receiving information to/from other elements through a communication network.

In our distributed version of the robot controller we use four of those units, two for the control of two wheels (actuators) and other two for the control of two IR distance detectors (sensors).

2.2.The robot

The robot used for the experiments is a software simulation of a squared platform where two infrared sensors and two motors are attached. Three wheels control the movement of the robot: two wheels at the bottom, controlled by one motor each one, and a free wheel at the front, which gives stability but no control.

Sensors are also simulated. They imitate the behavior of infrared sensors and are placed at the upper-left corner of the robot, one pointing to the front and another pointing to the left. The first sensor measures the distance between the robot and an object in the front. The second one measures the distance of the left side of the robot and an object at his left. Because of that, they will be called the Y sensor and the X sensor respectively.

Sensors have been modeled to be able to detect objects from a range between 3 and 20 cm. Things out of that range are not detected, so it is possible for the robot to be in front of an object and not detect it because of been too close. The detection values of the sensors have been quantized allowing 4 possible levels (far, medium, close and very-close) in order to keep the whole system simpler. Motors, together with the two main wheels, are placed at both lower corners of the robot. Their range of velocities has been also quantized, only allowing 4 different levels (full forward, half forward, stopped and half backwards). In the centralized controller, motors and sensors are all controlled by one net, but in the distributed version, motors and sensors have each one their controller which are implemented using intelligent hardware units.

Physics of the robot movement was emulated by using a correspondence table, i.e., a table of values was constructed based on the measurements of the movements of a real robot, by specifying the translation and rotation amount that the robot would suffer when a specified voltage were supplied to the motors. The table was based on Rasmunsen (2000) work. A small bias on motor values due to imperfections was included, but no noisy effects were taken into account.

2.3.The sensor sub-agent

Sensor sub-agents require special attention. In other robot designs, sensors deliver the sensed values to a controller which decides what to do with them. A post processing of the sensed value could be applied, been most of the times a processing algorithm determined by the designer of the robot.

However, in our design, the sensor (in fact, the intelligent hardware unit) itself decides which kind of processing for the received signal from the hardware sensor is required. The intelligent sensor unit decides if processing is necessary and of which kind, depending on a number of factors: what other sub-agents are on, the type of used neural net, the sensor features, and the current value sensed. Processing is implemented on the neural network of the intelligent hardware unit that conforms the sensor, and will be learned by it during co-evolution.

2.4.The evolutionary procedure

When it is necessary to evolve different nets for different roles in a common task, then a co-evolution algorithm is required (Han yong and Miikkulainen, 2001). This is, to evolve a group of agents in order to show them how to *cooperate* to achieve a common goal, when every agent has its own and different vision of the whole system.

Co-evolution generates a control policy evolving the required knowledge through experience, including all the necessary behaviors to meet the specific demands of the domain (Gómez and Miikkulainen, 1996;). So, if a following-wall behavior is necessary for the agent, then the evolution algorithm will evolve networks that are capable of performing that way (Moriarty and Miikkulainen, 1996). An agent using this method will be able to learn behaviors, as for example avoiding obstacles, without coding them inside the agent by hand.

The method selected to evolve the nets is the ESP (Enforced SubPopulations) reinforcement learning algorithm. It presents several advantages such as: it is an unsupervised method with no need of training examples, it allows co-evolution (evolve different agents with different roles and a common goal) and it is specially designed to obtain a distributed controller. It also works perfectly when cooperation between agents (since *cooperative* co-evolution will be required (Balch, 1997), this is, all sub-agents sharing rewards and penalties for successes and failures). The ESP algorithm implemented here follows (Aedula and Dagli, 2002) description, including delta-coding to prevent premature convergence (Gómez and Miikkulainen, 1999).

3. EXPERIMENTS

Considered experiments consist of a series of computer simulations implemented in Scilab and C++. The C++ code is a modification of an implementation of the ESP algorithm by the UTCS Neural Nets Research Group (UTCS, 2002). The original UTCS code was created to solve the pole-balancing problem with a central controller approach, so the modifications implement the robot control problem with a distributed control. Results of the nets can be obtained in a visual way by means of the design of a robot simulator and its environment using Scilab (Scilab, 2003). The Scilab code takes the final neural nets generated by the ESP algorithm and uses them as the control of a robot, showing the behavior of the robot on the screen.

Table 1. Required values for the ESP algorithm evolution

N. subpopulations	12 for distributed 48 for central
Neurons per population	40
Mutation Rate	0.4
N. trials per neuron	10
Stagnation after	20 evaluations
N. Steps	300

The goal task is to obtain an autonomous robot able to find an object on his space and then orbit around the object in an endless loop. This behavior will have to emerge by the cooperation of the four sub-agents inside the robot. To obtain that behavior the following fitness function was defined:

$$F = \begin{cases} +1 \ when \ Sx = CLOSE \ \wedge \ Sy = FAR \ \wedge \ VLeftRight > 0 \\ 0 \qquad elsewhere \end{cases} \quad (1)$$

This function rewards the neurons only when the robot is running with both wheels, detecting an object on his 'left' at 'close' distance, and detecting nothing in front of him.

All the experiments correspond to computer simulations, since no real robot has been used yet, it is a work in progress. Some values required for the algorithm are presented on Table 1. On that table, the number of subpopulations is equal to the number of hidden neurons; the neurons per population parameter indicates the number of neurons available for use on each subpopulation; the mutation rate expresses the rate at which neurons are mutated; the number of trials neuron informs about the mean number of times that every neuron of every subpopulation must be tested before recombination is started; and the stagnation parameter defines the number of trials without improvement before delta-coding is invoked. The number of steps indicates the amount of step times that an evaluation is run and equals the maximum fitness reachable. Every experiment was run 10 times, and the average results are shown on tables 2 and 3.

3.1. Centralized controller

The UTCS software was used to solve the robot centralized control problem by modifying the program from its original role (the pole-balancing problem). The original UTCS code used ESP to evolve a central controller, so the first attempt was to check the possibility to evolve a central controller able to exhibit the required behavior.

For the central-controller approach, only one artificial neural network was required. The learned net, obtains the quantized data from the motors and sensors and generates the required responses for the motors (no

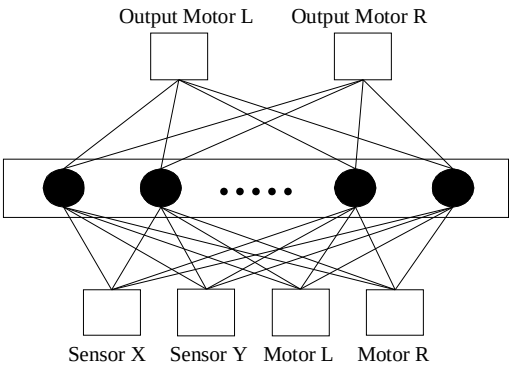

Fig. 3. Artificial neural network that implements the central controller of the robot.

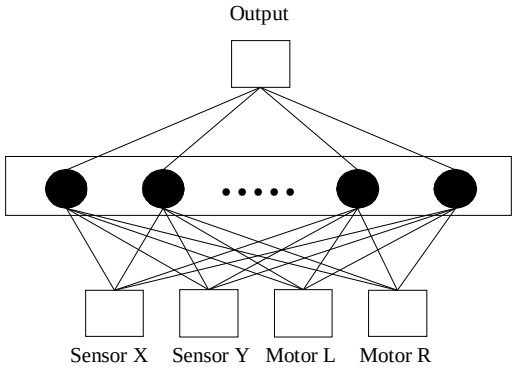

Fig. 4. Artificial neural network used for the processing unit of each intelligent hardware unit implementing a sub-agent in the distributed controller.

Table 2. Results for the central-controller evolution

Average number of generations required	140
Max. Fitness	239

Table 3. Results for the cooperative-controller evolution

Average number of generations required	78
Max. Fitness	254

output is required for sensors since they are *passive* elements). Fig. 3 shows the net employed, where the hidden neurons are generated by using the ESP algorithm.

The required behavior was obtained after an average of 140 generations, being the average maximum fitness obtained of 239 out of 300 steps. When evolution was run for several hours, until generation 1500 was reached, fitness maximum did not improved.

3.2. Distributed controller

The *robotic agent* is decomposed into a group of *sub-agents.* Each sub-agent is in charge of one of the sensor/actuator units of the robot and is implemented using one Intelligent Hardware Unit (however, everything is emulated by a software program). The processing part of the Intelligent Hardware Unit learns how to use its hardware part (the sensor or actuator) at the same time that learns how to cooperate with other sub-agents, i.e., sub-agents do not learn in a standalone manner but in a cooperation environment.

Now, the UTCS software was modified to have one artificial neural network like that in fig. 5 for every sub-agent, implementing in this way the control structure shown in fig. 3. All the neural networks were evolved at the same time. By doing this, a distributed controller was obtained. When the same parameters than those of table 1 are used, and the same fitness function of the second experiment, the same behavior is evolved after 78 generations.

Significant differences respect the central-control experiment can be observed: first, the average number of generations required to obtain the same behavior was drastically reduced. Secondly, when evolution is left run some more generations, the maximal fitness obtained reached values of about 260 out of 300.

Comparing results can be seen that a cooperative (or distributed) controller for a robot can learn faster and better than a central one, confirming the idea stated in (Han yong and Miikkulainen, 2001) for the predator-prey game.

3.3. The real job of sensor sub-agents

When using four sub-agents to control the robot, the question about the necessity of using neural networks in the sensors arises. Since its only job is to receive the value from the sensor and deliver it to the rest of neural networks, is it really necessary to use those networks in such passive elements?. Is the neural network attached to the sensors doing any job?. To answer those questions we performed two more experiments.

In the first one, we created a distributed controller with only two sub-agents, one controlling each motor. Outputs from sensors X and Y were directly connected to the neural networks of those actuators (no neural nets were attached to those sensors). Results showed that the ending robotic agent was able to have the required behavior, but his fitness dropped down a little bit (an average of 232 out of 300, against 254 out of 300 for the four agents distributed controller). From that result, we stated that sensor sub-agents are really doing some helpful job, it is, they are leaning something that improves the robot behavior.

In the second experiment, we tried to see if the learned function of the neural nets of the sensor sub-agents was always the same or was it affected (adapted) by the nature of its associated sensor. To test this, we plotted the function learned by sensor X network under two different circumstances:

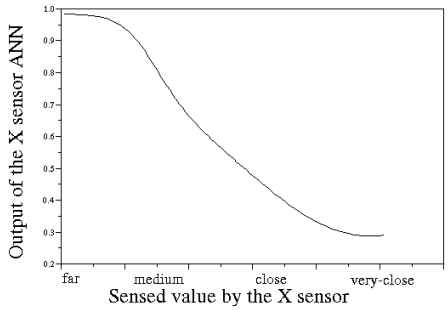

Fig. 5. Function learned by the X sensor network. Even that the function depends on 4 variables, Y sensor, Left motor and Right motor have been fixed, showing only the dependence of the network output in front of sensed X values.

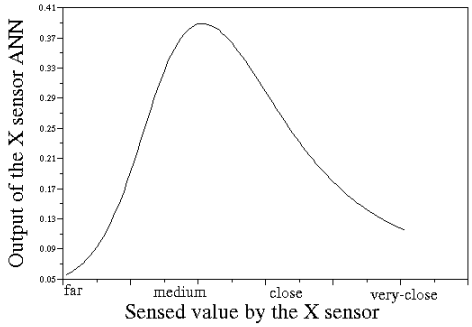

Fig. 6. Function learned by the X sensor network when the sensor has a malfunction. Same conditions as in Fig.5 have been applied.

first one, showed on Fig. 5, shows the function learned by sensor X when Motor L and Motor R value 'half-speed' and sensor Y senses nothing.
On the second one, we modified the code that emulates the sensor X and introduced a modification, emulating a sensor failure. Failure consisted on only allowing to sense objects at 'close' and 'very-close' distances (no 'far' distances allowed). After this modification, we evolved the 4 sub-agents again, and plotted the resulting sensor X network when Motor L and Motor R value 'half-speed' and sensor Y senses nothing (Fig. 6).

Comparing both figures, it can be seen that nets are learning how to process the input signal, and that this processing highly depends on the sensor behavior. First of all, figures 5 and 6 show that the sub-agents are doing a job and it is not only taking the value sensed and sending it to the rest of nets. Second, this job depends on the global situation. It would be possible then, to use nets on the sensors able to adapt themselves to a noisy environment and learn what kind of treatment would be necessary to take the most of the environment to perform the task required. This will be confirmed with further work.

4. DISCUSSION

Results on 3.1 and 3.2 show that a distributed controller performs better that a centralized one when evaluated on the same task. However, there are no big arguments that could justify the use of one of them over the other one, because both controllers were able to present the required behaviors with just small differences in performance. Nevertheless, the idea behind the distributed controller is that of providing a universal platform to be used on *any* control system. Given its modularity, the distributed controller presented here is suitable for use in any other domain requiring complex control and where multiple information from several sensors and actuators must be taken into account at the same time, like for example the control of a domotic house (our european Cicyt ACCUA project is at present dealing with this), and it looks like it will be more capable to handle this situation when the number of sensors and actuators increases (current job under development). Given its distributed nature, our controller should allow for a better scalability and a more robust control system in front of errors or even malfunction of one of the Intelligent Hardware Elements. Nevertheless those objectives are out of the present paper been our main concern to show that the distributed controller is a real alternative. Universality, scalability and robustness of this kind of distributed controllers will be explored in future papers.

Experiments on section 3.3 demonstrate that the use of sensor sub-agents improve the performance of the controller and allows to take the maximum profit of any information received from a sensor. If a sensor behaves badly, only the neural networks that take the best of that faulty sensor will evolve. This could lead to completely ignore the information sent by the bad sensor (if it is too noisy) or just to pay attention to determined values. It could be said that the sub-agent performs an adaptation between the sensor and the robot *brain*, and the sub-agent itself must find the best adaptation.

All those results could suggest that a good direction to move on when designing the control of a robot could be to only specify the architecture and the connections, and let the robot find his best controller. Sensor and actuator integration would be achieved during evolution in the way the robot thinks is best. Nothing would be hard coded but the neuron connections and the fitness function (that can be very unrestrictive to allow uncountable behaviors, as shown in (Floreano and Mondada, 1996)). Further experiments will need to be done to confirm this point.

5. CONCLUSIONS

It has been shown that it is possible to make cooperate several sub-agents to control a small robot, at least at a small scale, by using co-evolution algorithm ESP. Directly demonstrated advantages of this approach are the automatic emergence of required behaviors and a faster learning and better performance than a centralized controller. Also, introducing sub-agents on passive sensors improves the results and allows the engineer to forget about the problems of processing and integration of sensors values. Potential advantages include scalability and robustness.

Further work is in progress to demonstrate scalability and direct application to real robots, and it will be presented during the conference.

ACKNOWLEDGEMENTS

This research is partially supported by the Cicyt ACCUA project number TIC2003-09179-C02-01. We thank Mario Martin for insightful suggestions and comments.

REFERENCES

Aedula, V. and Dagli, C. (2002). Collective Behavior in Robots Using Evolutionary Neural Networks. *Presentation.*

Balch,T. (1997). Learning roles: behavioral diversity in robot teams. *College of Computing Technical Report GIT-CC-97-12*

Floreano, D. and Mondada, F. (1996). Evolution of Homing Navigation in a Real Mobile Robot, *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396—407

Floreano, D. and J. Urzelai, J. (1997). Evolution and Learning in Autonomous Robotic Agents, In *Bio-Inspired Computing Systems*, D. Mange and M. Tomassini,191-36

Gómez, F. and Miikkulainen, R. (1996). Incremental Evolution of Complex General Behavior. The University of Texas at Austin, *Technical Report AI96-248*

Gómez, F. and Miikkulainen, R. (1999). Solving Non-Markovian Control Tasks with Neuroevolution. *Proceedings of the IJCAI99*

Han Yong, C. and Risto Miikkulainen, R. (2001). Cooperative Coevolution of Multi-Agent Systems. *Technical Report AI01-287*, Department of Computer Sciences, University of Texas at Austin

Minsky,M. (1988). The Society of Mind. *Touchstone Books*

Moriarty, D. and Miikkulainen, R. (1996). Evolving Obstacle Avoidance Behavior in a Robot Arm. In *From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptative Behavior*.

Nolfi, S. and Floreano, D. (2000). Evolutionary Robotics : The Biology, Intelligence, and Technology of Self-Organizing Machines.MIT Press.

Rasmussen, J. (2000). A general robot simulator for evolutionary robotics. *Master's Thesis in Computer Science*, University of Copenhagen.

Scilab (2003). Scilab software for Linux and Windows: http://www.scilab.org

Stanley, K.O. and Miikkulainen, R. (1996) Efficient Reinforcement Learning through Evolving Neural Network Topologies.

Steels L. (1994). Emergent Functionality in Robotic Agents through On-Line Evolution, in Brooks R.A., Maes, P.(eds.): *Artificial Life IV, Proc. of the Fourth Int. Workshop on the Synthesis and Simulation of Living Systems*, MIT Press

UTCS (2002) Neuro-evolution software at the UTCS Neural Nets Research Group: http://www.cs.utexas.edu/users/nn/pages/software/software.html

Werner and Dyer (1991). Evolution of communication in artificial organisms. In *Proceedings of the Workshop on Artificial Life 1990.*