Neuro-evolved Agent-based Cooperative Controller for a Behavior-based Autonomous Robot

Ricardo A. Téllez[1], Cecilio Angulo[2]
Knowledge Engineering Research Group (GREC), U.P.C.,Vilanova i la Geltrú, Spain
[1]r_tellez@ouroboros.org, [2]cecilio.angulo@upc.es

***Abstract*--we present in this work a cooperative control paradigm for the autonomous navigation of a mobile robot and demonstrate that the cooperative controller learns faster and better than a centralized one. Behaviors emerge from the neuro-evolved controller, in order to achieve a designed task and without been defined at design stage. In our proposal, a robotic agent is divided into sub-agents, each one controlling one sensor or actuator element of the robot. Meanwhile the sub-agent learns to handle the element, it also learns to cooperate with the other ones. The emergence of behaviors happens when the co-evolution of several sub-agents embodied into the single robotic agent stabilizes. A distributed version of the ESP neuro-evolving algorithm is used for the evolution of the overall distributed controller.**

***Index Terms*--autonomous robots, agents, co-evolution, neural networks**

I. INTRODUCTION

In the field of autonomous robots, three main control techniques are used: the hierarchical control technique, where a main brain generates a mental representation of the space and uses it to guide the robot, the behavior based technique, where different behaviors are defined and they control the general robot behavior, and the hybrid control approach, which tries to use the best of the two previous control techniques [17].

Although the hybrid control type is a very promising technique, at present, is the behavior based control the most successful one. It consists on defining several behaviors required for the robot job, organized in different layers. Interaction between layers is allowed only from top to bottom, and behavior conflict is a common problem not completely and satisfactorily solved yet [14][16][11]. A common implementation of those principles is the one found in the subsumption architecture [15]. A way to avoid those conflicts and even the necessity of creation and definition of behaviors is presented in this paper. Initially, no behaviors are defined, just the final task to achieve (in the form of a fitness function). All needed behaviors for the completion of the task will be learnt by the robot (will emerge when required) by means of a reinforcement learning algorithm and its associated fitness function.

Similar results have been reported by Floreano and others [2][3] where a real robot was capable of emerge an *avoid obstacle* behavior and even a *search for food* one. Though, the main difference between the methods of Floreano and ours is the type of controller evolved.

We have taken as point of departure the idea presented by Minsky [12] where societies of sub-agents cooperate to control a single agent. In this paper a robot is seen as an agent composed by several units called sub-agents. Each sub-agent is in charge of one sensor/actuator unit of the robotic agent and it is implemented by a neural network. Sub-agents will learn to cooperate by communicating with the others, in order to accomplish the agent task. While learning to cooperate, necessary behaviors for the global task will emerge without been specified.

Learning the required robot controller can be done by several methods. One demonstrated successful is *neuro-control*, it is, the use of neural networks to learn to implement the control policy.

Neuro-control learning can be implemented using supervised training or neuro-evolution. While supervised learning uses a series of examples to train the networks by methods such as backpropagation, neuro-evolution uses genetics algorithms to evolve the required neural nets. The advantage of neuro-evolution over supervised learning is that it doesn't needs training examples to acquire the control knowledge. Neuro-evolution will also allow the robot to learn any required task to complete the job, without been specified before hand.

Due to the necessity of controlling multiple sub-agents and to allow cooperation between them, a co-evolution method to obtain a proper control policy will be required. Co-evolution is a neuro-evolution method for the evolution of different nets with different roles in a common task. The co-evolution algorithm used here is a version of the *Enforced SubPopulations* algorithm (ESP [5],[6]) with some modifications to obtain a distributed controller.

This paper is distributed as follows: section II describes the architecture taken. Then, section III describes the neuro-evolution method used, with special interest in the distributed controller approach. The paper continues with section IV, describing different implementations of the ESP algorithm inside a robot simulator and the results obtained with several experiments. Future work and conclusions are exposed in sections V and VI.

II. THE ARCHITECTURE

The *robotic agent* used here is decomposed in a group of *sub-agents.* Each sub-agent is in charge of one of the sensor/actuator units of the robot, and learns how to use that unit at the same time that learns how to cooperate with other sub-agents. It is, sub-agents do not learn in a standalone manner but in a cooperation environment. It is said that they learn how to use sensors and actuators for a *common good.*

Communication between sub-agents becomes a very important feature in this implementation [21]. It allows sub-agents to send information to each other in order to correctly coordinate. During the training phase, the information sent between sub-agents is used to learn

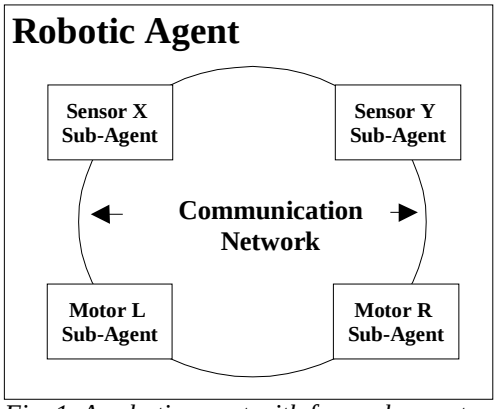

Fig. 1. A robotic agent with four sub-agents

coordination. When training is finished, sub-agents have learn how to treat information about the state of other sub-agents to coordinate with them, and they will use communication to maintain that coordination. Furthermore, communication gives reaction capabilities in front of unexpected situations to the robotic agent, as showed in [1].

On the design presented here, communication consists of sending to the other sub-agents information about the next step that a sub-agent will take. A total communication between all the agents has been implemented here through a communication network which connects all sub-agents between them (fig. 1).

Similar architectures have successfully been applied in real time control systems, like for example the control of two building elevators, or warehouse management, where a complex communication system including contracts and requests, was required for a proper collaboration of the different agents [7][8].

III. CO-EVOLUTION OF SUB-AGENTS

A. Definitions

To teach the agent how to behave in his world a training mechanism is required and *neuro-evolution* is the selected method. Neuro-evolution is concerned with the use of genetic algorithms to evolve neural networks. Neuro-evolution generates a control policy evolving the required knowledge through experience, including all the necessary behaviors to meet the specific demands of the domain [5]. This means that, if a following-wall behavior is necessary for the agent, then the neuro-evolution algorithm will evolve networks that are capable of performing that way [4]. So, an agent using this method will be able to learn behaviors, as for example avoiding obstacles, without telling the agent about it, and without coding it inside the agent by hand.

When it is necessary to evolve different nets for different roles in a common task, then a *co-evolution* algorithm is required [1]. This is, 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.

There are two types of co-evolution methods: *competitive co-evolution*, where the role of each agent is against the role of the other agents (what an agent loses another agent wins); and *cooperative co-evolution*, where agents share rewards and penalties of successes and failures. The last type will be the one used for this problem since each subpart is evolved on its own population and interacts and cooperates with the others to solve the problem, contributing its best to the final solution.

To score the evolved neural network a *fitness* function is created. Once a neural network has been evolved, it must be tested on the domain to see how it performs. The result of this test is given by the fitness function, and says how good or bad is the present net. The score will guide the evolution in one way or another.

The fitness function has to be calculated (and often, readjusted) experimentally. It mathematically defines the *global behavior* required for the robot.

B. The ESP co-evolution algorithm

There exists several co-evolution algorithms. The one used here is the ESP algorithm.

ESP stands for Enforced SubPopulations. It is a neuro-evolution method to evolve sub-populations of neurons forming a global neural network. In this process, groups of single neurons are created (called *sub-populations*). A neuron is selected from each sub-population to form a hidden layer unit of a global neural network. This neural network is evaluated on the problem (for example, the Predator-Prey domain in fig. 2), been the fitness passed back to the participating neurons.

The fitness is an evaluation score of the neuron's performance inside the global ANN. Usually all the participating neurons receive the same fitness.

Every neuron inside one of the groups of sub-populations encodes with real values the state of the connections of such neuron within the global ANN. The information coded on every neuron is called the *genome* and it is the information that is evolved by the genetic algorithms.

The ESP algorithm implemented here follows the description given in [20], including delta-coding to prevent premature convergence [6].

C. Multi-agent ESP architecture

ESP can be used to evolve the neural network to control more than one agent. In fig. 2, ESP is used to generate a single controller for 4 different agents in the Predator-Prey domain. There exists a neural network called the Global Neural Network which controls the 4

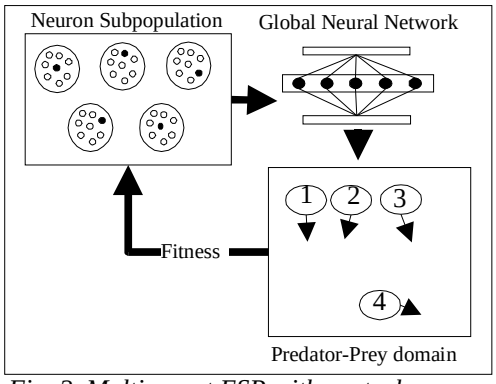

Fig. 2. Multi-agent ESP with central controller approach

agents, it is, the outputs of the net encode in some way what each agent must do at next time-step.

The situation in fig. 2 is said to have a *central-controller* approach. However, an *autonomous-controller* or *distributed-controller* approach is also possible. In the distributed-controller case, every agent is controlled by its own neural network (fig. 3). It has been demonstrated for the predator-prey game that evolving cooperative distributed controllers is more effective than evolving a central controller [1]. In this paper we test if this result can be applied to the control of an autonomous robot, composed by several cooperative sub-controllers instead of one big central controller (fig. 3).

During co-evolution, the nets of all sub-agents must have a reward provided by the fitness function. In [19] Balch shows that rewarding the whole team with the same fitness produces *cooperation* between agents, while rewarding each agent with its own fitness value induces more *competitive behavior* (because every agent tries to maximize its own reward without paying attention to the group's goal). Since the experiments tried to achieve a cooperation between sub-agents to successfully control the robot behavior, all sub-agents were rewarded with the same fitness value.

IV. EXPERIMENTS

The experiments consisted 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 [13]. The original UTCS code was created to solve the pole-balancing problem with a central controller approach, so the modifications implemented the robot control problem with a distributed control.

To see the results of the nets in a visual way, a simulator of a robot and its environment was created using Scilab [18]. The Scilab code takes the final neural nets generated by the ESP algorithm and uses them as the control of a simulated robot, showing the behavior of the robot on the screen.

A. Description of the physical platform

The goal is to obtain an autonomous robot able to find an object on *his* space and then start orbiting around the object in an endless loop. This behavior will have to emerge by making cooperate four sub-agents *inside* the robot. All the experiments correspond to simulations into a computer, since no real robot has been used yet (work in progress).

The robot used for the experiments consists 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 emulate 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 front of him. The second one measures the distance of the left side of the robot and an object at his left. So, they will be called the Y sensor and the X sensor respectively.

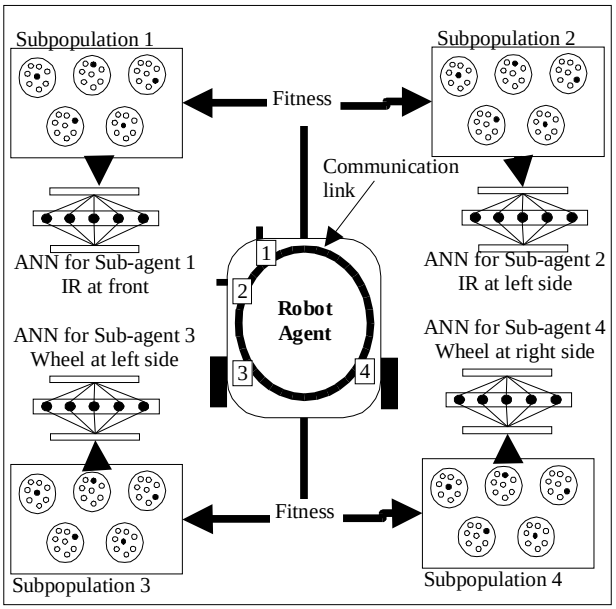

Fig. 3. An autonomous robot controlled by four sub-agents

Sensors have been modeled to be able to detect objects from a range between 20 and 3 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 in order to keep the whole system simpler. It will be considered that an object is:

- FAR from the robot, when distance is greater than 20 cm.
- HALF distance from the robot, when distance is between 20 and 10 cm.
- CLOSE to the robot, when distance is between 10 and 6 cm.
- VERY-CLOSE to the robot, when distance is between 6 and 3 cm.

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 values:

- FULL FORWARD
- HALF FORWARD
- STOP
- HALF BACKWARDS

Physics of the movement of the robot was emulated by using a correspondence table; it is, based on the measurements of the movements of a real robot, a table was constructed, 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 work [9]. This included a small bias on motor values due to imperfections, but no noisy effects were taken into account.

B. Description of the software platform

In this case 4 different sub-agents are required to represent the whole system (one for each IR sensor and one for each motor). Each sub-agent is implemented by a feed-forward artificial neural network with sigmoid

activation function and one hidden layer with 12 neurons.

Communication between sub-agents is performed by connecting the outputs of the neural networks to the inputs of the other nets (including itself). That is, neither special language nor protocol has been designed for the communication between sub-agents, just a plain communication of the decision taken by the nets. Since only four agents are required, all the nets have four inputs: one for the output of the net controlling sensor X, one for the output of the net controlling sensor Y, one for the output of the of net controlling the left motor (L) and one for the output of the net controlling the right motor (R).

All the nets only have one output neuron, encoding the value of the input between 0 and 1. For both motors and sensors, the outputs of their nets are quantized. The quantization of the values implies that even though the outputs of the nets can produce any value between zero and one, only a few values are allowed as entries to the input neurons. This quantization makes the training faster and the whole system simpler.

The desired behavior for the robot is the following: the robot will be placed randomly in some free point of the space. That space contains a square object in the middle. First, the robot will have to look for the object using a random search algorithm. Once the robot finds the object he will start orbiting around it forever at a CLOSE distance (between 10 and 6 cm).

In order to avoid an endless loop when no object is at a certain detectable distance of the robot, a random search routine was created. It happens that, when no object is close enough to the robot to be detected, the robot generates a movement in a loop manner. Even that this loop can have different shapes, if an object can not be detected in any of the positions of the robot trajectory, the robot would remain forever doing the loop without finding the object. To avoid that behavior, a random search routine was activated every time the sensors sensed nothing, bypassing the orders given by the controller formed by the nets.

Two types of experiments were done: one first experiment using ESP with a central controller and a second one using the distributed controller version of the ESP.

Some values required for the algorithm are presented bellow:

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

Table 1. Table of values required for ESP evolution

C. Simulation with a central controller approach

In this case, the UTCS software was used to solve the robot control problem changing the program from its original role (the pole-balancing problem). The UTCS code used ESP to evolve a central controller, so the first attempt was to see if it was possible to evolve a central controller capable of maintaining the required behavior.

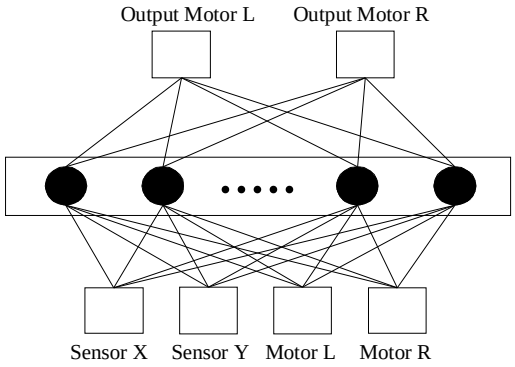

Fig. 4. ANN for the central-controller approach

For the central-controller approach, only one neural network was required (the central controller). The net obtained the (quantized) data from motors and sensors and generated the required responses for the motors (no output were required for sensors since they are *passive* elements). Fig. 4 shows the net used, been the hidden neurons generated using the ESP algorithm.

The number of subpopulations equals the number of hidden neurons, the neurons per population parameter indicates the number of neurons available for use in each subpopulation, the mutation rate expresses the rate at which neurons are mutated, the number of trials per neuron says the mean number of times every neuron of every subpopulation must be tested before a 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.

In order to obtain the required behavior the following fitness function was defined:

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

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.

The required behavior was obtained after an average of 140 generations, been the maximum fitness obtained of 239 out of 300 steps.

Average number of generations required	140
Max. Fitness	239

Table 2. Results for the central-controller evolution

Another interesting result is that the evolution was left running for several hours until generation 1500 was reached, but fitness never improved, having its maximum at 239.

D. Simulation with a cooperative controller approach

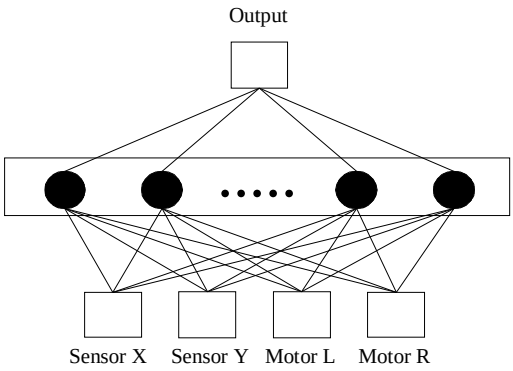

Fig. 5. ANN used for each sub-agent

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

Average number of generations required	78
Max. Fitness	237

Table 3. Results for the cooperative-controller evolution

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

Both results show that a cooperative (or distributed) controller for a robot can learn faster and better than a central one, as was stated in [1] for the predator-prey game.

E. Additional results

It must be said that, due to the sharpness of the central object that the robot must detect and orbit, and to the fact that the neural nets used are simple feed-forward nets, the robot looses contact sometimes during his orbiting. It is comprehensive that this happens because of the existence of the random search algorithm. When the robot is following the object and it suddenly ends, the robot senses nothing. He suddenly goes from being on the track to been completely lost. This situation activates the random search routine, making him to sniff for the trace of the object. The same *sniffing* behavior was observed in all the experiments and it should be avoided by using recurrent neural nets instead of plain feed-forward nets.

As an additional result, the robot was allowed to detect the walls of the environment. The center object was also put very close to the wall, so the robot would detect the wall when orbiting around the object. In the simulation, the robot reached the point where he was orbiting around the central object but detected the wall. At that point, the robot switched his focus from the central object to the wall, and started to follow the wall clockwise.

This result is important when switching from the simulation to a real robot, because it will allow the robot to orbit around any shape object and also to avoid obstacles (even that the *avoid obstacle behavior* was never mentioned).

V. FUTURE WORK

Further work is planned based on the robot architecture explained. This includes:

1. Implementation of the neural networks on a real robot: this work is in progress, with a real robot containing two ultrasonic sensors, two light sensors, one line detector, one battery detector and two wheels. Eight sub-agents are then required.
2. Scale up *width*: more sensors and actuators. It needs to be tested if this architecture can still work where large arrays of sensors and actuators are necessary.
3. Scale up *height*: more sub-agent layers. It needs to be tested how different layers of sub-agents can cooperate and coordinate, and also how to make use of lower layers.
4. Apply this architecture to other control processes, like domotics. It looks like this kind of structure can be applied to multiple domains [1][4][7][8], not only controlling autonomous robots, but any system requiring control.

VI. CONCLUSIONS

It has been shown in this work the possibility of making cooperate several sub-agents for the control of a small mobile robot, in a first stage at a small scale by defining a multi-agent version of the standard co-evolution algorithm ESP. Advantages of this approach are: it is not necessary to specify what kind of behaviors are required for a robot to perform a job, since all the required behaviors will emerge by themselves; the distributed controller learns faster and better than the distributed controller (in accordance with [1]); furthermore, it is expected that the distributed control will allow for more scalability than the central control. This point will be confirmed with further work.

VII. ACKNOWLEDGMENTS

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

VIII. REFERENCES

[1] C. Han Yong and R. Miikkulainen, *Cooperative Coevolution of Multi-Agent Systems.* Technical Report AI01-287, Department of Computer Sciences, University of Texas at Austin,1999

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

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

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

Behavior,1996

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

[6] F. Gómez and R. Miikkulainen, *Solving Non-Markovian Control Tasks with Neuroevolution.* Proceedings of the IJCAI99

[7] G. Schrott, *A Multi-Agent Distributed Real-Time System for a Microprocessor Field-Bus Network*. In Proc. of 7th Euromicro Workshop on Real-Time Systems, Odense, Denmark, Juni 14-16 1995

[8] G. Schrott,*Reactive Real-Time Programming with Distributed Agents.* In Euro-Par '97 Parallel Processing, Third International Euro-Par Conference, 1997

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

[10] K. O. Stanley and R. Miikkulainen, *Efficient Reinforcement Learning through Evolving Neural Network Topologies.* Machine learning, 22:11-32, 1996

[11] M. J. Mataric', *Behavior-Based Control: Main Properties and Implications*. In Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems,1992

[12] M. Minsky, *The Society of Mind.*Touchstone Books,1988

[13] Neuro-evolution software at the UTCS Neural Nets Research Group:
http://www.cs.utexas.edu/users/nn/pages/software/software.html

[14] P. Maes, *The Dynamics of Action Selection*. In IJCAI-89

[15] R. A. Brooks, *A Robust Layered Control System for a Mobile Robot*. In IEEE Journal of Robotics and Automation, RA-2, April 1986

[16] R. A. Brooks, *Integrated Systems Based on Behaviors*. In the special issue of SIGART on Integrated Intelligent Systems of July 1991

[17] R. C. Arkin, *Behavior-Based Robotics.* The MIT Press,1998

[18] Scilab software for Linux and Windows: http://www.scilab.org

[19] T. Balch, *Learning roles: behavioral diversity in robot teams, 1997*

[20] V. Aedula, C. Dagli, *Collective Behavior in Robots Using Evolutionary Neural Networks*,Presentation, *2002.*

[21] Werner and Dyer, *Evolution of communication in artificial organisms*. In Proceedings of the Workshop on Artificial Life 1990.