IMPROVING REUSABILITY OF BEHAVIOR BASED ROBOT COGNITIVE ARCHITECTURES OBTAINED THROUGH EVOLUTION

R. J. Duro, J. A. Becerra, J. Santos
Grupo de Sistemas Autónomos, Universidade da Coruña, Mendizábal s/n, Ferrol, Spain
gsa@cdf.udc.es

ABSTRACT

In this paper we discuss some of the new work we have been carrying out with the objective of making evolutionarily obtained behavior based architectures and modules for autonomous robots more standardized and interchangeable. These architectures are based on a hierarchical behavior structure where all of the modules, as well as their interconnections, are automatically obtained through evolutionary processes. The emphasis of this work is to produce behavior based structures that work on real robots operating in real environments and to be able to obtain them as independent of the platform as possible. To address this problem we have introduced the concept of virtual sensors and effectors in behavior based architectures and studied different approaches to automatically obtain them.

1. INTRODUCTION

Mobile Robots are very good examples of systems that can be autonomous. They interact with their environment performing actions in order to achieve objectives as a function of perceptions and previous actions. Obviously, as Van de Velde (Ref. 1) indicates, for a system to be autonomous it must organize its own internal structure in order to behave adequately with respect to its goals and the world, that is, it must learn. Learning involves several aspects that affect the cognitive and physical structure of a robot. It must be carried out all the way from the organization of perceptual information to the synchronization of the actuation of the robot.

Behavior based robotics (Refs. 2-4) draws inspiration from natural phenomena and its emphasis is on behavior and fast reaction and not knowledge and planning. The main ideas behind these proposals are that representations are not necessary for intelligence, except in a very limited sense, as the world is a good enough model of itself, that there must be a direct interaction between the robot and the world and that the robot must be taken as a whole and constructed bottom up, and not as in traditional AI as a sum of independent knowledge modules. The focus of these reactive systems is on behaviors, which can be taken as "stimulus response pairs modulated by attention and determined by intention" (Ref. 5), where attention prioritizes tasks and provides some organization in the use of sensorial resources and intention determines the behaviors to be activated depending on the objectives or tasks the robot must achieve.

Thus, in the behavior-based approach to robotics, behaviors are the building blocks of the robotic cognitive system. Explicit abstract representations of knowledge are avoided. Animal behavior is taken as a model and the systems are inherently modular from a software point of view.

One of the main problems of behavior based architectures lies in the difficulty in scaling them. To make a real agent operate in a real environment doing something useful, it is necessary to provide it with many individual behaviors that interact with each other. This requires obtaining the individual behaviors and, what is more complex, generating the interaction patterns between them to achieve what is expected from the agent.

In this paper we have addressed this problem from the point of view of a methodology for automatically obtaining multiple module behavior based architectures for individual robots so that reuse of behaviors is maximized (intra-robot behavior reuse). Additionally, we have designed two ways of introducing virtual sensors and behavior interfaces so that behaviors generated contemplating one robot model could be used when designing behavior architectures for a different robot model (inter-robot behavior reuse).

2. INTRA-ROBOT BEHAVIOR REUSE

When but a few behavior modules are required in order to implement a robot behavior controller, the complexity in a design scales with the number of possible interactions among modules. This was already pointed out by Cliff et al. (Ref. 6). If the behaviors and their interconnections are designed by hand, two problems arise, one due to complexity and the other to the fact that the hand-designed behaviors are not necessarily the best, or, in some cases even adequate for the task.

The next problem that arises is how to structure the different behaviors so that meaningful tasks can be accomplished. It is important in any structure, or strategy for obtaining global controllers, to minimize the participation of the designer and maximize behavior reuse in order to simplify the overall architecture.

In the late eighties and early nineties, artificial evolution was proposed as a means to automate the design procedure of these types of systems (Refs. 6-8). Many authors have taken up this issue and have developed different evolutionary mechanisms and strategies in order to obtain robotic controllers, mostly applied to single behavior modules in the form of Artificial Neural Networks (ANNs), Augmented Finite State Automata (AFSM), Classifier Systems (CS) or any other type of paradigm. When considering more complex structures it is necessary to provide mechanisms for reducing the participation of the designer so that the behaviors can evolve more freely and adapted to the specific characteristics of the robot. An attempt must be made to reduce complexity and the more modules that are reused the more economical the design process will become (Figure 1).

Monolithic architectures implement all the behaviors in the same controller, whether an ANN, a CS, or an AFSM, etc. The advantage they offer is that it is not necessary to have prior knowledge about potential sub-behaviors and the interrelations between them. The disadvantage is that it is not possible to reuse the individual behaviors. If a new behavior is required it is necessary to evolve the complete module again, even if it could have reused previously learnt or evolved behaviors.

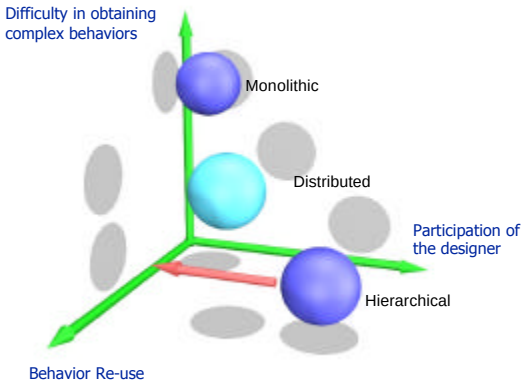

Figure 1. Comparison of relevant parameters for different general types of behavior architectures. An optimal one would maximize behavior re-use and minimize designer intervention and complexity. This is what we aim to achieve through the automatic generation of hierarchical structures.

In hierarchical modular architectures, the global behavior is decomposed, as necessary, into lower level behaviors that will be implemented in particular controllers. The higher-level controllers can take information from the sensors or from low-level controllers, and depending on the architecture, act over the actuators or select a lower level controller for activation. The advantage of these methods is that the behaviors can be obtained individually and then the interconnection between them can be established. Also, it is possible to reuse the behaviors obtained when implementing higher-level behaviors. The problem that arises is that the decomposition is not clear in every case, as it implies a specific knowledge of what sub-behaviors must be employed. This, in general, implies a greater participation of the designer in the process of obtaining a global controller.

Finally, distributed architectures, where there are no priorities and all the controllers compete for control of the actuators at each instant of time lead to less participation of the designer. They also preserve the level of behavior reuse. However, as a drawback, they induce a higher level of difficulty when obtaining complex behaviors.

3. AUTOMATIC DESIGN

In the work presented here, we have tried to combine, in a practical way, the advantages of a monolithic approach and a hierarchical modular structure so that complex behaviors could be generated automatically but took into account the experience accumulated through the implementation of previous behaviors.

In particular, using this method (Figure 2) a designer provides the system with whatever behaviors he has or decides that may be useful. This initial set need not be complete and may include many unnecessary behaviors. When obtaining the higher level controller, the evolution process will select those lower level behaviors from the initial set that are useful in order to perform the task assigned and will ignore the rest. If some part of the global behavior cannot be obtained through the interconnection of the available modules, a new monolithic module that handles this part will be co-evolved with the global controller

In a first step, a designer must identify sub-behaviors that may be useful to generate the global behavior required. The designer need not be exhaustive nor does he need to be concise, i.e. there is no problem if useless behaviors are included in this preliminary set.

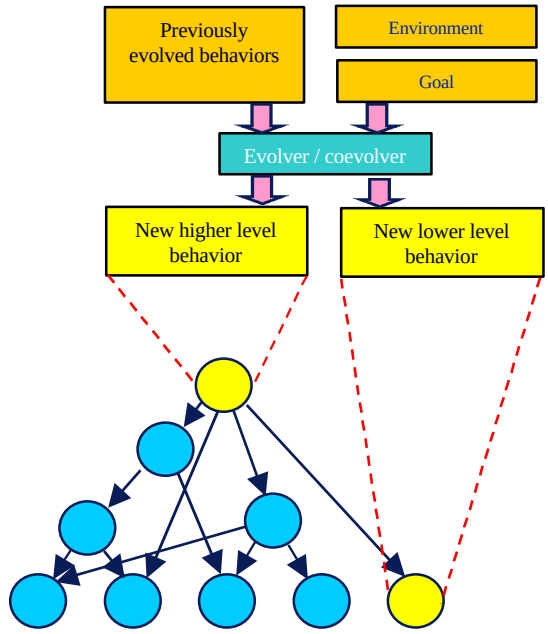

Figure 2. Schematic of the operation of the automatic behavior architecture evolver.

The selection and interconnection of these behaviors are carried out by means of an additional ANN that is evolved. Thus, the designer does not have to specify which low-level module must be executed at each moment in time. The evolutionary algorithm will evolve the ANN so that it makes this selection appropriately, based on data from sensors or from the outputs of other behavior controllers. This higher-level ANN differs from the lower level ones in that its output does not always act over the actuators; it mostly selects the adequate lower level behavior. There is nothing in this architecture to limit the number of levels to two. The lower level behavior selected by a given higher-level behavior may itself select a lower level behavior that acts over the actuators.

Using this structure, the controllers may be used in multiple composite behaviors without duplicities. As new behaviors are evolved, this evolution becomes faster and simpler due to the fact that a lot of experience in the form of previously evolved behaviors is available.

To prevent the problem of the designers having to be exhaustive in their determination of all the necessary lower level behaviors, we have included the possibility of cooperatively coevolving lower and higher-level behaviors. That is, a higher-level behavior may be evolved by itself using previously evolved lower level behaviors, or it may be coevolved with part of the lower level behaviors and use the previously evolved ones. This way, when the designer is faced with a problem where he is only able to identify part of the behaviors that may be involved, the unidentified ones will be evolved at the same time as the higher-level controller. This option makes the evolution of complex behaviors a lot easier for the designer, as it works even when the designer is not capable of identifying any of the behaviors involved. The designer can provide a set of previously evolved behaviors as a guess and let the evolver do the rest.

4. INTER-ROBOT BEHAVIOR REUSE

Until recently most behavior modules, whether implemented by means of neural networks or any other paradigm, were evolved or created for a given robot type in a certain environment (Refs. 6, 9). Environment that was particular in terms of noise and/or features.

An attempt to increase the reusability of modules generated for one robot in a different one is made in the work by Floreano and Mondada (Ref. 10), in which the authors evolve controllers for a Khepera robot that are used as seeds in the evolution of controllers for a Koala robot. This second robot is similar to the first one but larger in size. This method would not be valid for robots that present different sensing capacities, which is the general case.

To prevent all of these drawbacks and, at the same time, preserve the inherent advantages of behavior based architectures in terms of autonomy, and of evolutionary robotics in terms of automatic development of systems, in our group we have addressed these problems through the use of virtual sensors and effectors. For us, a virtual sensor is just a stage that integrates, interprets and homogenises information from a single or several sensors and provides a standardized output that is taken up by the behavior based control mechanism. A virtual effector is a stage that interprets standardized inputs and converts them into motor commands for a given effector or set of effectors. Thus, a first approach to produce controllers that may be used in different robots implies changing from a structure that links the environment directly to

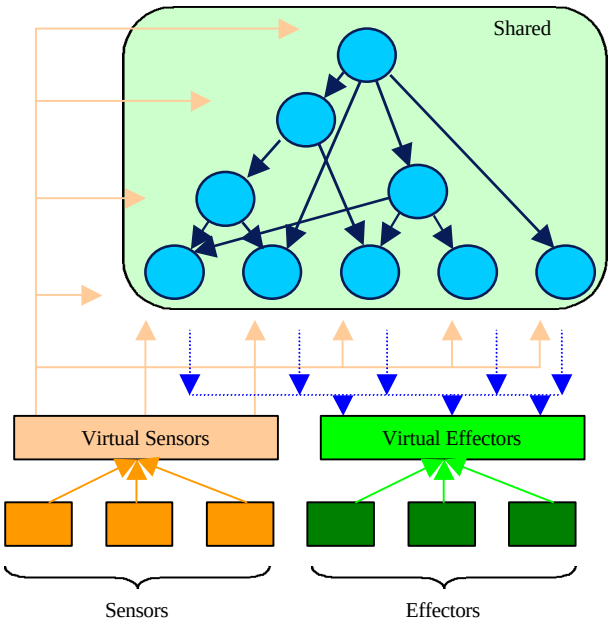

Figure 3. Architecture where the whole behavior structure is shareable between different types of robots using virtual sensor and effector interfaces.

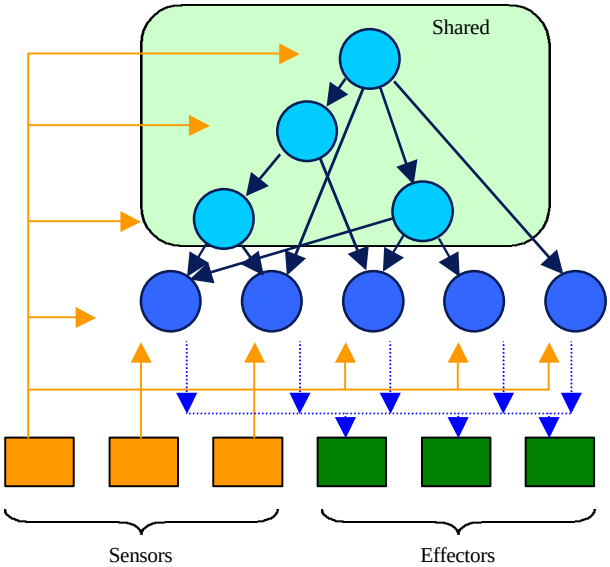

Figure 4. Architecture where part of the behavior structure is shareable between different types of robots and part is particular to each robot. The interface is at the behavior level.

the behavior based control mechanism, to a structure that implies intermediate modules, which are in charge of common pre-processing tasks (figure 3). In a sense, these modules regularize and structure common sensorial input and output spaces for the control modules. That is, the virtual sensors and effectors act as an interface between different sensor and effector sets (in the same or different robots) with the same behavior module. It is important to note that, in order to prevent an inadequate selection by the designer, we may also obtain the virtual sensors and virtual effectors using automatic procedures, such as learning or evolution.

A second approach to improving inter-robot behavior reusability, and one that is much closer to the concept of behavior based robotics is to include the interface between robot-specific and shared modules at the level of behaviors. In our case, as the behavior architectures generated by our systems are basically a hierarchy of behaviors, a division between low-level, robot specific behaviors and higher-level shared behaviors is relatively easy to obtain. Thus, even though the lower-level behaviors are robot specific, different robots can share the higher-level structure. This idea corresponds to the diagram shown in figure 4. Note that obtaining lower level behaviors for robots is, in general, quite simple as compared to the whole structure

5. APPLICATION EXAMPLES

All of the behavior architectures we use are based on Artificial Neural Network modules. They are evolved in simulated environments and then transferred to real robots. For a simulation to be appropriate for the transference of behaviors developed using it to the real world, it must meet some criteria, such as those established by Jakobi (Ref. 11), which usually imply handling different levels and types of noise. We have taken these criteria into account in our simulator adding generalization noise, random noise, temporal noise and systemic noise.

Regarding the simulation/evolution environment, we have selected a Macro Evolutionary Algorithm (MA) (Ref. 12). In our case, the original structure of MAs has been adapted to the development of ANN based robot controllers by uniformly distributing the initial population throughout the search space and subdividing it into races to prevent the clustering that arises when a random initial distribution of the low fitness individuals of small populations is employed.

The evolutionary algorithm can evolve a single low level behavior, a high level behavior that makes use of the low level behaviors we provide it with or co-evolve both simultaneously. The encoding scheme employed for the Neural Networks that implement the controllers, is just a direct genotypic representation of the phenotype in terms of synaptic weights and delays, bias and sigmoid slopes when appropriate.

Each individual in the robot population must live its life out in the environment interacting with whatever objects are included in it. The fitness criteria, as described elsewhere (Ref. 13), involves the robot eating food in the environment. The food is distributed according to the task we want the robot to perform.

In the next two sections we present two examples of control structures developed using the methodology described above. The first one involves a hexapod robot for which we have developed a virtual sensor. This sensor must make use of temporal information to obtain appropriately precise information of where objects are located from an imprecise infrared sensor that is swept. Once the sensor is obtained the robot may use behavior modules developed for any other robot whose input requires a reasonably precise position of the objects in front of it. The second example involves a behavior hierarchy that was developed according to the second approach mentioned above. The higher-level modules are shared and the lower level ones robot specific.

6. VIRTUAL SENSOR FOR HERMES II

To present the power of virtual sensing we will make use of a simple, but very illustrative, example. It uses a time delay based ANN to increase the precision in object-detection of an infrared (IR) sensor of the Hermes II robot (figure 5).

Infrared sensors are very common in robots because they are quite robust and simple to use, but they are not very precise given their ambiguity regarding the exact position of an object. These devices, usually couple an IR radiation emitter tuned to a given wavelength and an IR receiver that selects reflections of the same wavelength. The infrared sensors in our Hermes II robot are located on the top of each leg so we have six IR

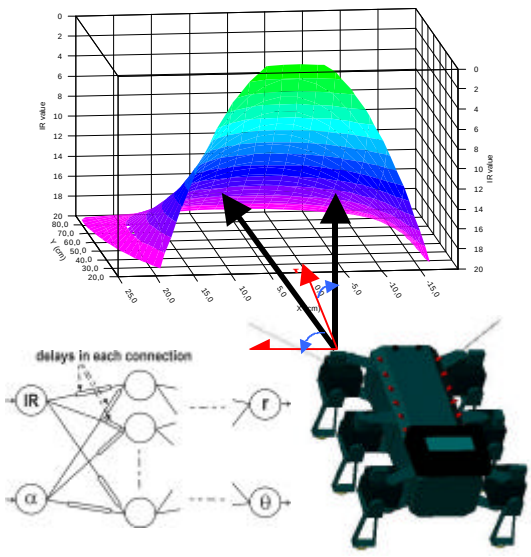

Figure 5. The low accuracy problem with the response of the Hermes II infrared sensor. At the left is the network with time delays used as virtual sensor.

sensors and they sweep a given area according to the motion of the leg. In figure 5 we display the response of the front-left IR sensor.

The output from the robot is encoded by means of a value between 0 (near) and 20 (far) and, as we can see, the shape of the curve makes it impossible to discriminate with precision the exact distance and angle at which the object is located. For example, if we place the same object in the two positions pointed by the arrows in figure 5, the output of the Hermes IR sensor would be 16 in both cases. This problem exists for all the parallel lines in the IR value axis, so we have a very important positional ambiguity problem. Notice also that this particular sensor detects the objects better if these are placed in front of it or to its right (although this is not a practical problem if we use all the IR sensors of the robot).

Through the inclusion of time in our IR sensing, and consequently through the introduction of previous information, the position of an object can be disambiguated. We use the IR sensor values taken in one sweep of the front-left leg and a synaptic delay based neural network manages these values.

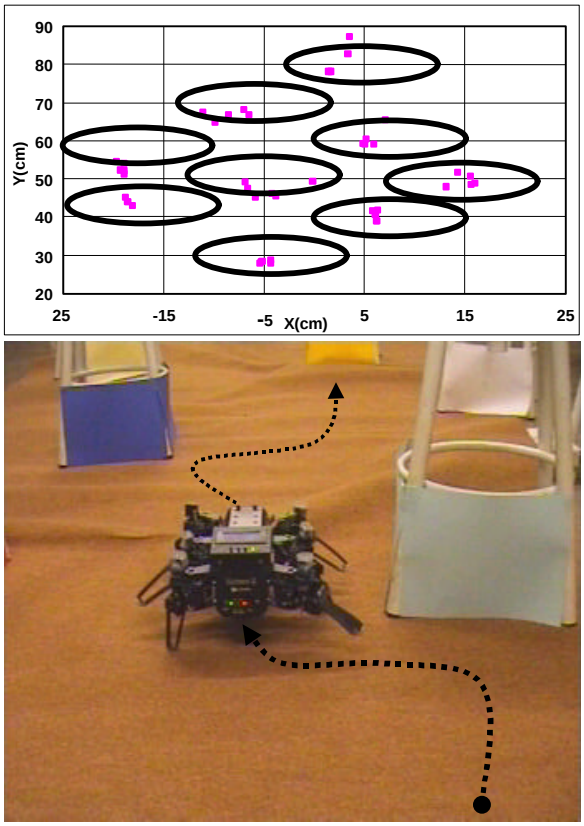

Figure 6. Top: Position data obtained by the trained networks (squares) for objects in different locations (ellipses). The sensor is located in position (0,0). Bottom: Trajectory of the Hermes robot using the virtual sensor and a single IR detector.

In figure 5 we also display the experimental setup including the angles and distances being considered. In each sweep we obtain the different values of the IR reading and the leg angle (á) so we have the two inputs to our neural network. The outputs of the network are the distance to the object (r) and the polar angle (è) at which it is located from the IR sensor. The neural network in the example has two input neurons; two hidden layers with 15 neurons each and both output neurons. We have assumed the requirement that two consequent measurements should overlap (the IR sensor covers exactly 21º in each IR detection), so we took 12 values of IR and á to cover the whole sweep of one leg (84º).

Figure 6 displays results for an experiment where we placed a tree-like object, represented by ellipses, at different positions in front of the robot (positioned at x=y=0) and plotted 5 outputs (different sweeps) from the network for each ellipse. We can see that the accuracy of our sensing is now adequate; the robot is able to locate the object independently of its lateral position. Using larger networks and more training examples may increase the accuracy. It is only limited by the repeatability of the outputs provided by the IR sensors. Another way to increase precision could be to make a finer sweep. In this case, the one obtained was quite enough for the task the robot had to perform, and that is navigate through an environment with tree like objects it had to avoid.

We have tested this network with the Hermes II robot in motion using a single sensor and it was able to detect the obstacles in its path and avoid them by going around them to one side or the other depending on the exact position of the obstacle in relation to the robot, as can be seen in bottom part of figure 6. On one hand, the sensing precision while the robot is in motion has been highly increased and. On the other hand, which is the point we want to make here, any behavior module developed for a different robot which uses sensorial information in a similar range regarding the position of objects in the robot path can now be used directly in the robot following the structure shown in figure 3 and does not need to be redeveloped for this particular animat.

7. SHARED HIGHER-LEVEL BEHAVIOR

The second example was implemented with the behavior hierarchy shown in figure 7. This behavior corresponds to a compound wall following task that comprises modules for finding a wall, following it and, if the robot collides, allowing it to get unstuck (collision avoidance). In this compound behavior we include a virtual sensor for the detection of collisions (developed for both a Rug Warrior robot and contact sensors and a Pioneer II robot using sonar and wheel encoders) and a virtual sensor for the detection of objects closed to the robot (employing IR sensors with the Rug Warrior and sonar sensors with the Pioneer II).

The Rug Warrior robot is a very small round robot with two very imprecise and noisy infrared sensors and three contact sensors. The Pioneer II robot is a much

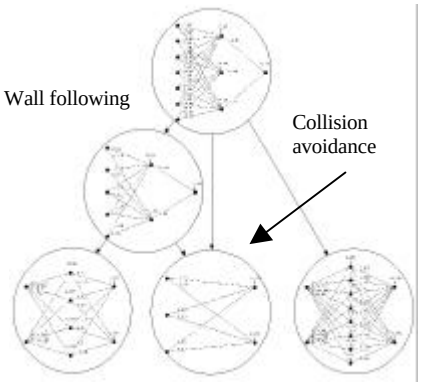

Figure 7. Behavior based architecture with the top module shared by the Rug Warrior and the Pioneer II robot.

bigger, almost rectangular, robot with sonar sensors and encoders. The Pioneer II moves much faster than the Rug Warrior.

In the example, the inclusion of these virtual sensors allows both robots to make use of the same higher-level behavior module (developed using the Rug warrior) to carry out a wall following task in an efficient manner, as shown in figure 8. In the tests we carried out both robots operated satisfactorily, finding the walls, following them with very few or no collisions and when a collision occurred, getting unstuck and continuing with the task.

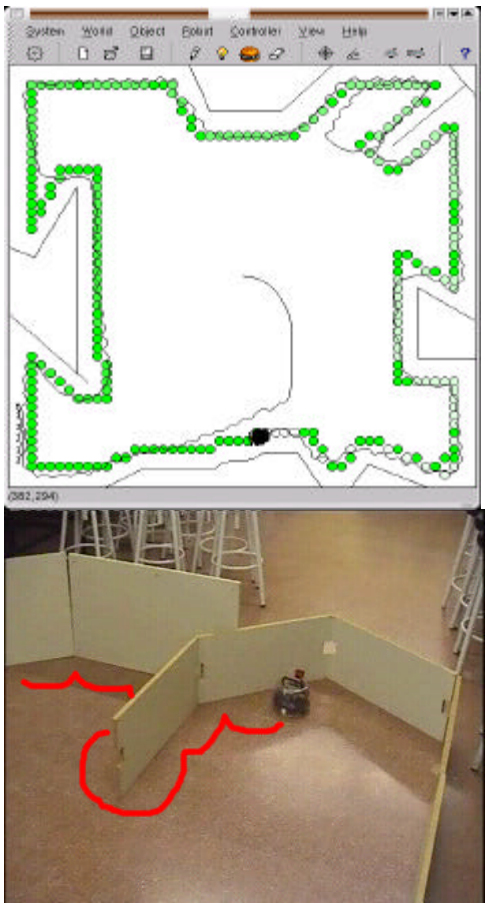

Figure 8.Operation of the Pioneer II robot (top) in simulation and the Rug Warrior (bottom) using the behavior based structure shown above.

The lower level behaviors, in this case, are particular to each type of robot, and thus make better use of their idiosyncrasies leading to better results in cases where the sensing apparatus of the robot is better, as in the case of the Pioneer II. In this case, the more elaborate sensing apparatus with sonar sensors all around the body of the robot, allows it to obtain a better wall following behavior, and without the need of use of temporal information as in the case of the Rug Warrior due to the binary nature of its infrared sensors. In addition, this structure is much more coherent with behavior based robotics than the previous one used in the Hermes II robot, but it does add an element of complexity and a new design decision must be made of where the interface between shared and particular behaviors must be established.

8. CONCLUSIONS

In this article we have presented a structured way of obtaining compound behaviors in different robots. This implies a hierarchical structure where the different behaviors are implemented through ANNs, some of them using temporal delays to consider time related events. Through this structure, the controllers may be used in multiple compound behaviors without duplicities, and, as new behaviors are evolved, this evolution becomes faster and simpler due to the fact that a lot of experience in the form of previously evolved behaviors is available.

To make things even easier, we have presented two ways of using behavior modules developed for one robot in the behavior structure of a different type of robot. These modules can be introduced as possible tools during the evolution of the global controller and thus, experience in the form of behavior controllers obtained in one platform can be readily used in a different one. The first approach implies using a virtual sensor and effector based interface between the real sensors and effectors of the particular robot and its behavior based architecture. Doing things this way has the advantage of simplifying the whole process and being able to use complete behavior architectures developed for one robot in another. The main drawback the approach presents is that the behavior architecture does not really take into account the advantages provided by the idiosyncrasies of each type of robot for performing particular tasks, and, consequently, leads to the problem of having to use the features of the least capable system as a reference. It also implies deciding the interface beforehand and thus losing part of the grounding property of the behavior based alternative. Additionally, from a conceptual point of view, it represents a trade-off between behavior-basedness and ease of implementation, thus hybridizing the paradigm.

The second approach implies developing lower level behaviors that are particular to each robot, but which are easier to obtain, and only sharing higher-level behaviors. This approach seems conceptually sounder

and, in practice, makes the unstructured development of complex architectures more straightforward. The alternative makes use of all the possibilities of the sensor apparatus of each robot, its range of detection and particular position of all the sensors; thus, the resulting behavior will be grounded. The compound wall following behavior with the Rug Warrior and Pioneer robots is an example of that, where the behavior is obtained with low level modules tuned to the different sensor apparatus of each robot. The drawback is that we move the decision of the interface between dependence and independence of the platform one step up, and thus we need to obtain these low level behaviors in all the robots, as opposed to the first alternative in which it would only be necessary to obtain the lower level behaviors once.

ACKNOWLEDGEMENTS

This work was funded by Xunta de Galicia under project PGIDT99PXI10503A.

REFERENCES

[1] Van de Velde, W, 1993, *Towards Learning Robots*, MIT Press, Cambridge, MA.

[2] Brooks, R.A. 1991, Intelligence without Representation, *Artificial Intelligence*, Vol. 47, 139-159.

[3] Maes, P., A 1990, Bottom-up Mechanism for Behavior Selection in an Artificial Creature, *Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90)*.

[4] Mataric, M.J. 1992, Integration of Representation into Goal Driven Behavior Based Robotics, *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 3, 304-312.

[5] Arkin, R.C. 1998, *Behavior Based Robotics*, MIT Press, Cambridge, MA.

[6] Cliff, D., Harvey, I. and Husbands, P. 1992, *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour*, Tech. Rep. No. CSRP256, Brighton, School of Cognitive and Computing Sciences, University of Sussex, UK.

[7] Harvey, I., Husbands, P., and Cliff, D. 1993, Issues in Evolutionary Robotics, *J-A. Meyer, H. Roitblat, and S. Wilson (Eds.), From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, MIT Press, Cambridge, MA, pp. 364-373.

[8] Beer, R.D. and Gallagher, J.C. 1992, Evolving Dynamical Neural Networks for Adaptive Behavior, *Adaptive Behavior*, Vol. 1, No. 1, pp. 91-122.

[9] Nolfi, S. 1997, Using Emergent Modularity to Develop Control Systems for Mobile Robots, *Adaptive Behavior*, Vol. 5, No.3-4, pp. 343-363.

[10] Floreano, D. and Mondada, F. 1998, Evolutionary Neurocontrollers for Autonomous Mobile Robots, *Neural Networks*, Vol. 11, pp. 1461-1478.

[11] Jakobi, N. 1997, Evolutionary Robotics and the Radical Envelope of Noise Hypothesis, *Adaptive Behavior*, Vol. 6, No. 2, pp. 325-368.

[12] Marín, J. and Solé, R.V. 1999, Macroevolutionary Algorithms: A New Optimization Method on Fitness Landscapes. *IEEE Transactions on Evolutioanry Computation*. V3, N4, 272-286.

[13] Becerra, J.A., Santos, J., and Duro, R.J. 1999, Progressive Construction of Compound Behavior Controllers for Autonomous Robots Using Temporal Information, *Advances in Artificial Life, Dario Floreano, Jean-Daniel Nicoud and Francesco Mondada (Eds.), Lecture Notes in Artificial Intelligence*, Vol. 1674, pp. 324-328, Springer-Verlag, Berlín.