Evolving the Walking Behaviour of a 12 DOF Quadruped Using a Distributed Neural Architecture

Ricardo A. Téllez, Cecilio Angulo, and Diego E. Pardo

GREC Research Group at Technical University of Catalonia, Spain

Abstract. This paper describes how a distributed neural architecture for the general control of robots has been applied for the generation of a walking behaviour in the Aibo robotic dog. The architecture described has been already demonstrated useful for the generation of more simple behaviours like standing or standing up. This paper describes specifically how it has been applied to the generation of a walking pattern in a quadruped with twelve degrees of freedom, in both simulator and real robot.

The main target of this paper is to show that our distributed architecture can be applied to complex dynamic tasks like walking. Nevertheless, by showing this, we also show how a completely neural and distributed controller can be obtained for a robot as complex as Aibo on a task as complex as walking. This second result is by itself a new and interesting one since, to our extent, there are no other completely neural controllers for quadruped with so many DOF that allow the robot to walk.

Bio-inspiration is used in three ways: first we use the concept of central pattern generators in animals to obtain the desired walking robot. Second we apply evolutionary processes to obtain the neural controllers. Third, we seek limitations in how real dogs do walk in order to apply them to our controller and limit the search space.

1 Introduction

The generation of a walking behaviour in a quadruped as complex as Aibo is a hard task. The robot has many degrees of freedom (twelve) and the coordination of all them to obtain a walking pattern becomes very difficult. In this paper, we evolve a neural controller for the Aibo robot only using neural networks distributed all over the robot. We base our design in the creation and utilisation of Central Pattern Generators (CPGs). Biological CPGs are composed of groups of neurons, capables of producing oscillatory signals without oscillatory inputs. It has been discovered that walking movements of cats and dogs are governed by those elements, and it is thought that humans too behave in the same way [1]. We will implement artificial CPGs using artificial neural networks (ANNs).

CPGs have been already employed by other researchers in the generation of gaits for robots, like for example by Ijspeert on lamprey [2] and salamander simulations [3], Kimura et al. on quadrupeds [4, 5] or Collins and Richmon in quadrupeds too [6]. Our work follows the research line taken by Ijspeert with

A.J. Ijspeert et al. (Eds.): BioADIT 2006, LNCS 3853, pp. 5–19, 2006.

their results in the generation of gaits for the lamprey and the salamander, but we try to slightly improve his results in several points: first, we use a more complex robot (every leg has three degrees of freedom that must coordinate); second, we apply it to a real robot; and third, we introduce a general architecture made of standard blocks (see our previous work [7]).

Previously to the present work, we developed a completely distributed architecture for the general control of autonomous robots. Our architecture provides a mechanism for the creation of controllers for autonomous robots in a modular and distributed way, and the architecture is completely based on neural networks. It has already been tested on other simpler tasks like standing up or standing [8], but not in such a complex task like walking. Walking requires special considerations since it contains a special dynamic that has to be taken into account (the robot movement has dependency on previous movement states). We say that while standing or standing up are static tasks (in the sense that for each sensory pattern there is a unique motor answer), walking requires the acquisition of the dynamics of the system, and for each sensory pattern, different motor answers could be applied, depending on the state of the movement (is not the same situation having the leg going than having the leg coming, even the sensory pattern at a specified position is the same).

To overcome the problem of capturing the dynamics of the system, we have used continuous time recurrent neural networks (CTRNNs), instead of using simple feed-forward neural nets like in our previous works. These are neural networks with internal states that allow the capture of the dynamics of a system and have already successfully been applied to the generation of walking patterns in other robots [9, 10].

In order to obtain the correct weights for the neural networks for the task we use neuro-evolution [11]. All the evolutionary process is performed under simulation using the Webots software [12], and once the simulation has the complete walking behaviour, we transfer the resulting ANNs to the real robot to test its validity on real life.

This paper is organised as follows: first we describe the architecture used for the experiments. Then the neural model used is described, followed by a description of the implementation of the architecture in the problem of walking generation is exposed in three stages. Last section discuss the results and points towards future work.

2 Architecture Description

The architecture is based on several uniform modules, composed of ANNs, where each module is in charge of one sensor or actuator of the robot. Through the use of a neuro-evolutionary algorithm, modules learn how to cooperate between them and how to control its associated element, allowing the whole robot to accomplish the task at hands (in this case, generate a walking pattern). All the architecture description has been highly inspired by the concepts of *society of mind* of Minsky [13] and *massive modularity of mind* [14].

2.1 Hardware

We define the Intelligent Hardware Unit (IHU) as a module created around a physical device of the robot (sensor or actuator). Every IHU is composed by a sensor or an actuator and a micro-controller implementing an ANN that processes the information of its associated device (received sensor information for sensors, commands sent to the actuator for actuators). We say that the ANN is in charge of its sensor/actuator. This means that is the neural net the one that decides which commands must be sent to the actuator, or how a value received from a sensor must be interpreted. All IHUs are interconnected to each other in order to be aware of what the other IHUs are doing. So in some sense, the net is also in charge of deciding what to say to the other elements as well as to interpret what the others are saying. The structure of a IHU can be seen in figure 1, and figure 2 shows a neural controller for a simple robotic system with two sensors and two actuators.

It should be stated that when put several IHU together on a control task, each element has its own particular vision of the situation because each one is in charge of its own sensor or actuator. This leads to a situation where each unit knows what the others are doing but needs to select an action for its controller

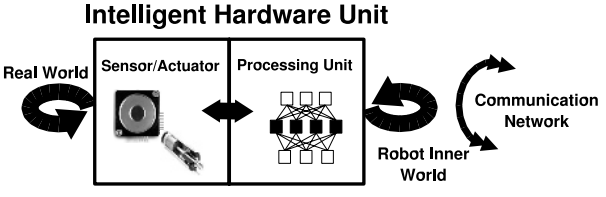

Fig. 1. Schematics of an IHU

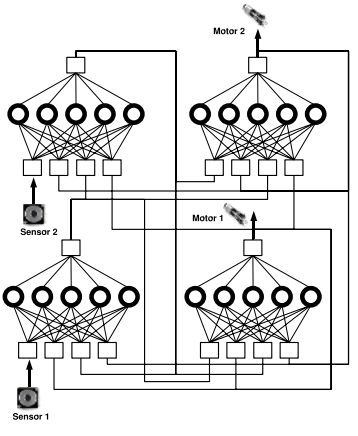

Fig. 2. Connection schema of four ANNs from four IHUs controlling a simple robot composed of two sensors and two actuators

or sensor output, and based on its knowledge of the global situation and that of its particular device, decides what the next action will be.

Even though in the original definition a microprocessor was required for any IHU element, on the experiments presented here it has been simulated the existence of the micro-controllers linked to each device by allocating some processing time in the robot central processor for each IHU, since it was not physically possible to have one dedicated micro-controller for each IHU, neither in the simulations, nor in the real robot tests. It will be assumed that the results are not very different from the original idea.

2.2 Neuro-evolutionary Algorithm

To teach the networks the coordination required for the task a neuro-evolutionary approach has been selected. For the co-evolution of the different networks and due to the necessity of evolving different ANNs for different roles on a common task, a co-evolutionary algorithm is required. By using such kind of algorithm it is possible to teach to the networks how they must cooperate to achieve a common goal, when every network has its own an different vision of the whole system.

The algorithm selected to evolve the nets is the ESP (Enforced Sub-Popula tions) [15][16], which has been proved to produce good results on distributed controllers [11]. This algorithm is also in principle free of bias for a special task. It is a general algorithm which produces good results in systems where several parts must interact to obtain the general view of the situation.

A chromosome is generated for each IHUs network coding in a direct way the weights of the network connections.

3 Neuronal Model

For the implementation of each of the neural elements of the IHUs we use a CTRNN. This type of neural network is composed of a set of neurons modelled as *leaky integrator* that compute the average firing frequency of the neuron [17]. All hidden neurons in one network are interconnected to each other (see figure 3).

The equations governing each hidden neuron are the following:

$$\tau_i \frac{dm_i}{dt} = -m_i + \sum w_{ij} x_j$$

$$x_i = (1 + e^{(m_i + \theta_i)})^{-1}$$

where m_i represents the mean membrane potential of neuron i, x_i is the short-term average firing frequency of neuron i, θ_i is the neuron bias, τ_i is a time constant associated with the passive properties of the neuron's membrane, and w_{ij} is the connection weight from neuron j to neuron i. Calculation of each neuron output is performed using the Euler method for solving differential equations with a step of 96 ms. More complicated models for the hidden neurons are

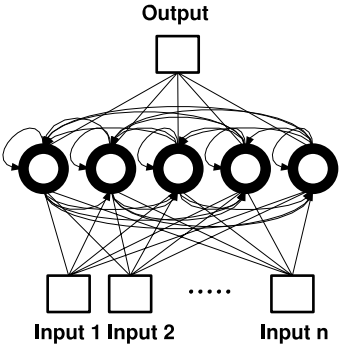

Fig. 3. Schematics of the CTRNN used in the walking controller

available and have been demonstrated quite useful in the evolution of gaits for quadrupeds [18], but they require higher computational resources that we try to avoid in order to make the robot as autonomous as possible.

For each neural element of the IHUs we use a neural net like the one showed in the figure 3, where the number of inputs depends on the stage of the evolution (see next section), the number of hidden units is five, and the number of output units is one.

For each network it is necessary to evolve the weights, the neuron bias and the time constant of each hidden neuron. However, as we will see in the next section, inter-neuron weights, bias and time constant are only evolved in the first stage (that is the stage that creates the CPGs). Later stages only evolve the interconnections between different CPGs.

4 Staged Evolution of the Walking Behaviour

For the generation of the walking behaviour we implement the explained architecture of section 2. We do not try to indicate that this may be the neural architecture on real dogs, but to show that our architecture is capable of performing the required behaviour.

We implement with our architecture what has been shown to be the way animals perform rhythmic movements like walking, and it is by implementing CPGs on each of its joints. In real animals, there is a CPG for each joint and they are interconnected only with the nearer CPGs. In our case, Aibo's joints are composed of a sensor (that obtains the position of the joint at every moment) and an actuator (that moves the joint). Since our architecture indicates that there must be an IHU for each sensor and actuator, we implement a CPG for each joint by the coupling of a neural net for the joint sensor and a neural net for the joint actuator (see figure 5).

Another difference between real CPGs and our architecture is that in real, only contiguous CPGs are connected to each other. In our case, as the architecture specifies, all IHUs must be connected to all IHUs. Then, all CPGs are connected

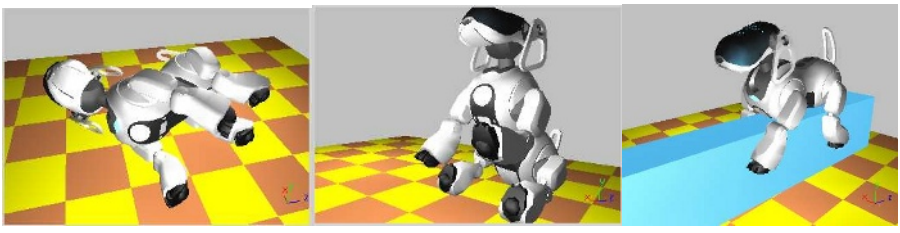

Fig. 4. Simulations used for the first and second stages. From left to right: evolving one leg joints, evolving two leg joints, evolving four leg joints.

to all CPGs. Nevertheless, all connections are not evolved at the same time, since, if that was the case, the search space for the evolutionary algorithm would be too high and the required walking solution never be found. For this reason, a staged evolution should be performed, in order to guide the evolutionary process a little bit to the correct solution. The different stages for the generation of the walking are: generation of the CPG oscillator, where a segmental oscillator is evolved for each type of joint, generation of a layer of joints of the same type that oscillate in counter phase by using the previously generated CPGs, and coupling of the three layers to obtain the final walking behaviour.

A very important point is that the generation of the oscillatory patterns is not performed aside of the robot, it is, we have not evolved an isolated oscillator unrelated to the robot. We evolve the oscillatory pattern over the robot itself (in the simulation). This allows the neural nets to capture the dynamics of the (simulated) robot, producing an oscillatory signal that takes into account inertias, decelerations, etc. All these features are important for a robot of the size of Aibo.

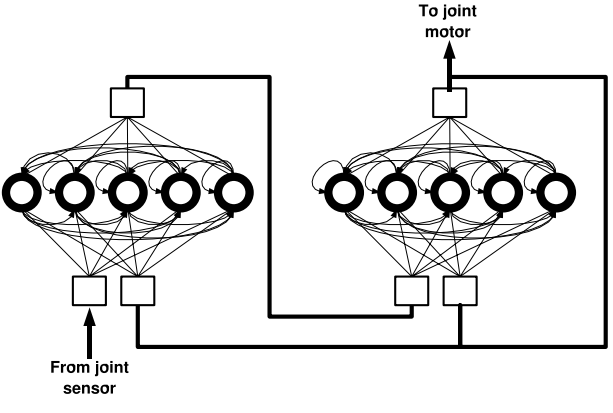

Fig. 5. Schematics of the coupling between two neural nets of a joint. This represents the coupling between the IHU of the joint sensor and the IHU of the joint motor.

4.1 First Stage: Generation of the CPG Oscillatory Pattern

At this stage we must obtain an oscillator capable of generate an oscillatory pattern for each type of joint of the robot. Joints in the robot legs are of three different types that we will call J1, J2 and J3. J1 is in charge of the rotatory movement, J2 of the lateral movement and J3 of the knee movement. Each joint is physically implemented using different PID controllers. Also their movement limits are different. For this reason, we must implement a different type of CPG for each type of joint (this is, three types of CPGs). Nevertheless, the process for the generation of each type is exactly the same, been the limitation of movements the only difference between them.

For each joint, we implement each CPG by the coupling of two CTRNN networks one for the sensor of the joint and another for the actuator (the motor). It is like we apply the architecture described in section 2 to a unique joint.

Both nets are interconnected as the architecture specify but each one is in charge of a different element (the sensor net is in charge of the sensor, and the motor net is in charge of the motor). At each step of the evolutionary process, the value of the sensor is read and entered in the sensor IHU. Then the output is computed and given to the actuator IHU. The output of the actuator IHU specifies the velocity that has to be applied to the motor, and, after escalation, it is directly applied to it. The evolution of the oscillatory movement is then performed over the robot, allowing this to include in the networks the effects of inertias and general dynamics of the robot leg (see figure 6).

The weights of the nets are evolved using the ESP algorithm and a fitness function that rewards the production of an oscillatory pattern in the motor joint. We do not specify the type of pattern to obtain but only that has to be periodic and between some oscillatory limits. Aibo joints can oscillate between very large limits, but those are too large for an appropriate walking behaviour. We limit then here the limits of oscillation by looking at the limits of real dogs and how

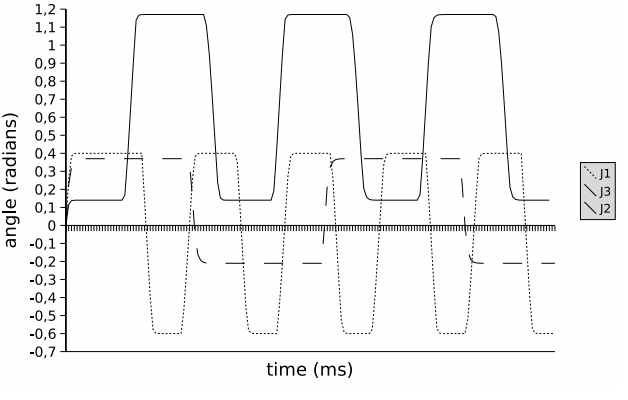

Fig. 6. Oscillatory patterns obtained for all three types of joints. Every joint has its own range of oscillation.

Table 1. Limits for Aibo joints based on real dogs movements when walking

Joint	Max	Min	Mean
J1, fore	0.3936	-0.5837	-0.0982
J2, fore	0.3702	-0.2163	0.0904
J3, fore	1.1732	0.1435	0.6305
J1, hind	0.0059	-0.7848	-0.4200
J2, hind	0.4215	-0.2163	0.1034
J3, hind	1.6599	0.9907	1.2499

do they perform when walking and making a scale conversion to our robotic dog [19]. From that gathered data we obtain the limits indicated in table 1.

The fitness function applied to the neuro-evolution algorithm is defined to reward regular oscillations within the limits of each joint. We want the system to generate a joint movement around the mean value of the table, and maximal variance within the limits of each joint. The fitness function is then:

$$fitness = [V - (A - M)] * C^2$$

been V the variance of the position of the joint during the 200 steps, A its average value, M the mean value of the joint obtained from table 1, and C the number of crossings that the joint performed through the mean position value.

Results: For each type of joint we carried ten runs starting with different initial random populations (weight values between -6 and 6). Each run was composed of 200 simulation steps of 96 ms on a first stage. After 13 generations all runs converged to networks capable of maintained oscillations within the range specified, and the number of steps was augmented to 400 for other ten generations, and later to 800 steps for five generations more. After this final stage, the networks were capable of a continuous oscillatory pattern on an unlimited amount of time.

As an additional note, indicate that the same oscillatory mechanism was obtained in some evolutionary test we performed where the CPGs were only composed of the actuator IHU, it is, no sensor IHU was included and the joint sensor was directly connected to the motor IHU. However we decided to include the sensor IHU for architecture's coherence, and having in mind future benefits. This will be more discussed in section 5.

4.2 Second Stage: Generation of Three Layers

From previous stage we obtained a group of different CPGs each one for a type of joint (three types). In this stage we are going to replicate the CPG of each type in the two fore legs in a first step, and for the four joints in all legs in a second step.

What we do in this case is to duplicate for each joint the CPG formed by the couple of two IHUs from one leg to the other. Duplication and new evolution is performed for one type of joint at a time. Once we have an IHU couple controlling

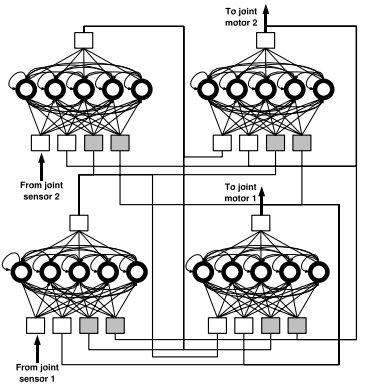

Fig. 7. Connections between four IHUs corresponding to two joints of the same type

each leg, then we apply again the architecture definition, that indicates how all IHUs must be interconnected between them. This implies that each neural net will have to add two more inputs coming from the outputs of the other two neural nets duplicated (see figure 7). The evolutionary process will only evolve the new connections between IHUs, but not the internal connections of the neurons obtained from the previous stage. Since the oscillation has already been obtained in the previous stage, this stage will not have to evolve it, but the synchronisation between the two CPGs. The type of synchronisation to evolve will depend on the type of gait required.

In this case, since we want the robot to implement a simple walking gait, we need a phase relation between those two legs of 180º (in all types of joints). The fitness function will be then that which punctuates the phase difference between the legs that is close to those 180º and rewards a continuous movement of both legs. To implement this function we divided it in three parts: two parts are the fitness function of the first stage for each leg. The third part is the one that measures the variance between the movements of both legs, and tries to maximise it.

$$O_i = [V_i - (A_i - M_i)] * C_i^2$$

$$VL = \frac{1}{N} \sum (diff_j - AvDiff)^2$$

$$fitness = O_1 * O_2 * VL$$

where O_i, V_i, A_i, M_i and C are the variance of the position of the joint during the N steps, the average value, the mean value of the joint obtained from table 1, and the number of crossings that the joint performed through the mean position value, respectively, for each joint *i*. VL is the variance between legs trajectories, N the number of steps, $diff_j$is the difference of positions between legs for each evaluation step *j*, and $AvDiff$ is the average of difference positions between legs.

Results: For each type of joint we carried ten runs to evolve only the connections between CPGs. Each run was composed of 400 simulation steps of 96 ms. After

14 generations, 90% of the networks were capable of a counter-phase oscillatory pattern.

Once we have this two legs oscillatory coupling, we replicated it to the rear legs, and repeated the evolutionary process to evolve only the weights of the new connections between the new IHUs. In this case, we needed to evolve 4 connections per network, having a total number of IHUs of 8 per type of joint. We evolved the whole group by imposing that the oscillations from the fore legs must have a 90º phase difference with the oscillations from the rear legs. To impose this condition, we calculated the difference of positions between fore legs (*diff_F*) and the difference of positions between rear legs (*diff_R*), in the same way as was done for the oscillation of two legs. Then we calculated the difference between the differences:

$$totalDiff = diff\_F - diff\_R$$

So finally the fitness to obtain the coordination was composed of five parts: one part for each leg that express the necessity of oscillation, one part that express the necessity of maximal variance between the fore legs, and one last part that express the necessity of maximal variance between differences fore-rear.

This is specified in the following fitness function.

$$O_i = [V_i - (A_i - M_i)] * C_i^2$$

$$VL = \frac{1}{N} \sum (diff_j - AvDiff)^2$$

$$varF\_R = \frac{1}{N} \sum (diffF\_R_j - AvDiffF\_R)^2$$

$$fitness = O_1 * O_2 * O_3 * O_4 * VL * varF\_R$$

where $diffF\_R_j$ is the difference between the difference of positions for fore legs and the difference of positions of rear legs for each evaluation step j, and

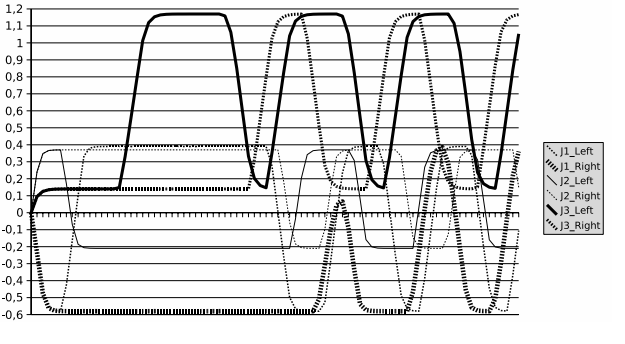

Fig. 8. Oscillations obtained for each type of joint when two joints are evolved

$AvDiffF\_R$ is the average value of such differences. varF_R measures the variance between fore and rear legs.

Results: We carried ten runs for each type of joint. Each run was composed of 400 simulation steps of 96 ms each. After 26 generations, 90% of the networks

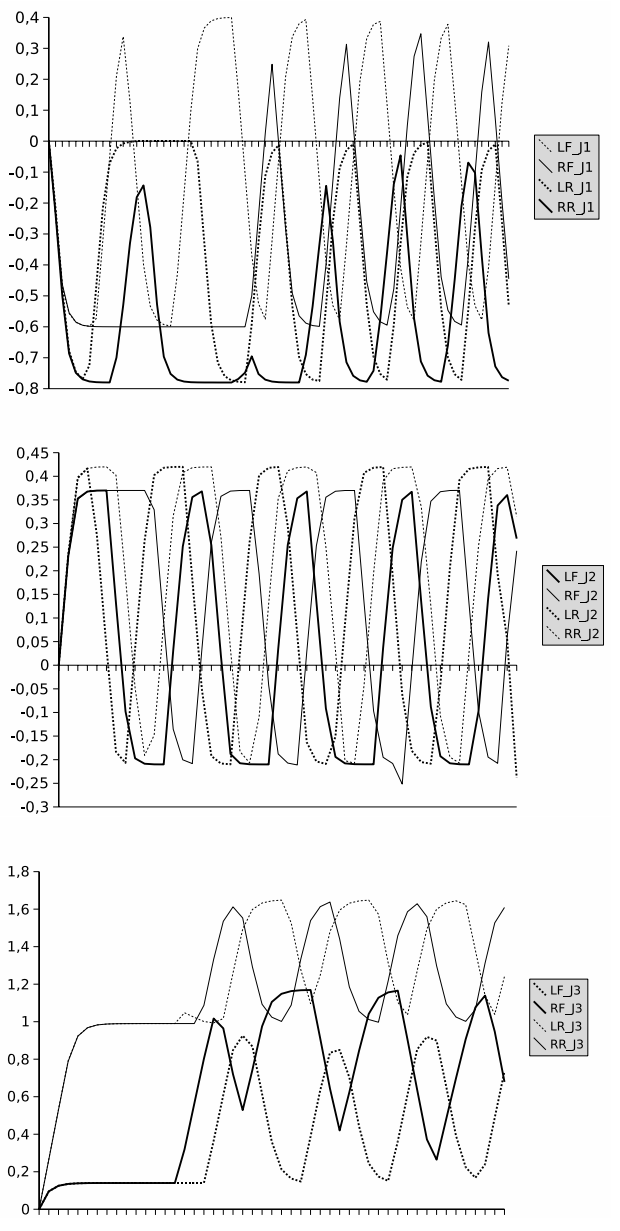

Fig. 9. Oscillations obtained for all types of joints in all legs

were capable of the typical oscillatory walking pattern 0º, 180º, 90º, 270º (for the legs sequence fore_left, fore_right, rear_left, rear_right). The oscillatory patterns obtained can be seen in figure 8.

4.3 Third Stage: Coupling Between Layers

Last stage is the coupling between layers of joints. From previous stage we have three different layers, one per type of joint, of four joints of the same type oscillating together with a *walking* phase relation. Now we need to connect the three layers between them in order to have the complete architecture finished. We will have then to evolve the connections between layers to finally obtain the whole robot walking with the full architecture completed. The connection between layers should bring coordination at walking between the different types of joints that have been evolved separately.

For this stage the fitness function is only the distance d walked by the robot, when the robot does not fall. Zero otherwise.

$$fitness = \begin{cases} d & when\ final\ height > 0 \\ 0 & otherwise \end{cases}$$

Results: A walking behaviour was obtained after 37 generations for about 88% of the populations. A sequence of the walking obtained is shown in figure 9.

The resulting ANN based controller was then transferred to the real robot using the Webots simulator cross-compilation feature that we have collaborated to develop with Cyberbotics. This cross-compilation process takes the exact controller developed in the simulator (the best of the evolved ones), and automatically translates it to Aibo OPEN-R code that is executed on the real robot.

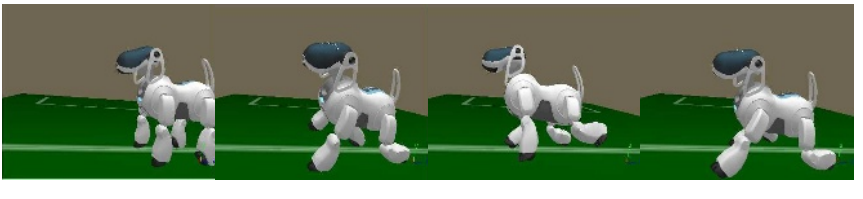

Fig. 10. Simulated Aibo walking sequence

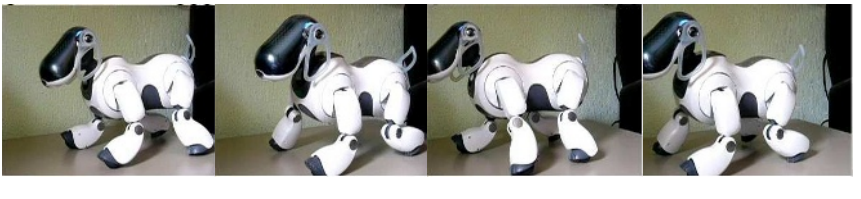

Fig. 11. Real Aibo walking sequence

The result was an Aibo robot that walks in the same manner as the simulated robot with some minor differences. A sequence of the walking obtained is shown in figure 10.

5 Discussion

The present paper shows how a distributed architecture can be used for the generation of gaits in a very complex robot. It also shows that a completely neural network based controller is possible for the generation of a walking behaviour in a quadruped of 12 degrees of freedom. Both of them are new results in the area of autonomous robots and intelligent control systems.

We have implemented each CPG by using two neural nets, one in charge of the sensor and one in charge of the actuator. In a formal way, the implementation of a CPG does not require the use of sensor inputs, but the introduction of the sensor networks could provide the system with a reflex system that may be helpful in front of unpredicted circumstances [20]. Our architecture does integrate already the sensor's feedback into the CPG, but its benefits have not been studied yet and is part of our future work. In particular, this reflex system would be integrated into the own CPG walking structure, not being a separated system, and could benefit the walking style in front of irregular terrain with small obstacles, allowing the robot to adapt to them and keep walking.

When developing the sequence of actions that would lead us to obtain a walking controller, we found that it was impossible to obtain a walking controller if the architecture was directly applied and all the nets (24) were evolved at the same time. The evolutionary algorithm always found an easier and useless solution other than walk in order to go forward. This is due to the complexity of the search space, that makes useless to perform a brute force search. This is the reason why an evolution by stages was required. But the evolution by stages has the drawback that a previous knowledge of the situation is required by the engineer in order to find the best way to implement the stages and find the good fitness functions, and this is one of the main criticisms against the evolutionary robotics methods. That is why, neuro evolutionary roboticists try to avoid as much as possible to introduce their knowledge of the situation, allowing the robot to find their own solution and not biasing the search of it. However, we do bet for the use of the engineer knowledge in the application of the evolutionary process, in order to reduce the search space. We do think it is necessary because, at difference at how real evolution did, we do have to evolve the robot controller on an already made robot, meanwhile real evolution evolved at the same time the structure of the living system and its controller. This puts us on disadvantage when compared to evolution, and that disadvantage needs to be overcome by our analysis and knowledge of the situation. This analysis of the situation should lead to an engineered evolutionary process with some engineer defined fitness functions, resulting on an equilibred evolutionary process that should restrict the search space enough but, at the same time, give space enough to the evolutionary process to explore for the solution. Other approaches are also

possible, like for example, to imitate real evolution and evolve at the same time body and controller as some recent works try to implement[21].

Acknowledgements

The authors would like to specially thank Professor Auke J. Ijspeert for insightful comments on the necessity of division of the search space in order to be able to find a good solution, and Dr. Olivier Michel for his support on the use of the simulator.

References

1. Grillner, S.: Neurobiological bases of rhythmic motor acts in vertebrates. Science **228** (1985) 143–149
2. Hallam, J., Ijspeert, A.: 4. In: Using evolutionary methods to parametrize neural models: a study of the lamprey central pattern generator. Physica-Verlag (2003) 119–142
3. Ijspeert, A.J.: A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. Biological Cybernetics **84** (2001) 331–348
4. Hiroshi Kimura, S.A., Sakurama, K.: Realization of dinamic walking and running of the quadruped using neural oscillator. Autonomous Robots **7**(3) (1999) 247–258
5. Hiroshi Kimura, Y.F., Konaga, K.: Adaptive dynamic walking of a quadruped robot by using neural system model. Advanced Robot **15** (2001) 859–876
6. Collins, J., Richmond, S.: Hard-wired central pattern generators for quadrupedal locomotion. Biological Cybernetics **71** (1994) 375–385
7. Téllez, R., Angulo, C.: Evolving cooperation of simple agents for the control of an autonomous robot. In: Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles. (2004)
8. Téllez, R., Angulo, C., Pardo, D.: Highly modular architecture for the general control of autonomous robots. In: Proceedings of the 8th International Work-Conference on Artificial Neural Networks. (2005)
9. Seys, C.W., Beer, R.D.: Evolving walking: the anatomy of an evolutionary search. In: From Animals to Animats: Proceedings of the eighth international conference on simulation of adaptive behavior. Volume 8. (2004)
10. Mathayomchan, B., Beer, R.: Center-crossing recurrent neural networks for the evolution of rhythmic behavior. Neural Computation **14** (2002) 2043–2051
11. Yong, H., Miikkulainen, R.: Cooperative coevolution of multiagent systems. Technical Report AI01-287, Department of computer sciences, University of Texas (2001)
12. Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems **1**(1) (2004) 39–42
13. Minsky, M.: The Society of Mind. Touchtone Books (1988)
14. Carruthers, P.: The case for massively modular models of mind. In: Contemporary Debates in Cognitive Science. Blackwell (2005)
15. Gómez, F., Miikkulainen, R.: Solving non-markovian control tasks with neuroevolution. In: Proceedings of the IJCAI99. (1999)
16. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Technical Report AI96-248, University of Texas (1996)

17. Hopfield, J.: Neurons with graded response properties have collective computational properties like those of two-state neurons. In: Proc. National Academy of Sciences USA. Volume 81. (1984) 3088–3092
18. Reeve, R.: Generating walking behaviours in legged robots. PhD thesis, University of Edinburgh (1999)
19. Nunamaker, D.M., Blauner, P.: Normal and abnormal gait. In: Textbook of small animal orthopaedics. International veterinary information service, USA (1985)
20. Ijspeert, A.: Locomotion, vertebrate. The handbook of brain theory and neural networks, second edition (2002) 649–654
21. Pollack, J. B., H.G.S.L.H., Funes, P.: Computer creativity in the automatic design of robots. LEONARDO **36**(2) (2003) 115–121