Evolution and Development of Modular Control Architectures for 1-D Locomotion in Six-Legged Animats

Jérôme Kodjabachian and Jean-Arcady Meyer
AnimatLab. Ecole Normale Supérieure. France.

Abstract— **An evolutionary approach is used to design neural control architectures for six-legged animats. Using a geometry-oriented variation of the cellular encoding scheme and syntactic constraints that reduce the size of the genetic search space, the developmental programs of straight locomotion controllers are first evolved. One such controller is then included as the first module in a larger architecture, in which a second neural module is evolved and develops connections to the first one, so as to set locomotion on or off according to tonic or phasic external control signals. Such an incremental approach should prove useful to the automatic design of relatively complex control architectures that might, in particular, implement some cognitive abilities over and above mere stimulus-response mechanisms.**

Keywords— **Evolution, Development, Dynamical Neural Networks, SGOCE, Hexapod Locomotion.**

I. Introduction

For a human, the design of the control architecture of an animat able to survive in a possibly changing environment is a highly challenging task because it is almost impossible to foresee each problem that the animat will have to solve and because there are — as of today at least — no basic principles upon which such design might rely [1]. To overcome these difficulties, many research efforts are directed towards the automatic design of control architectures, by means of a variety of evolutionary approaches that mimic the process of natural selection and that improve over successive generations the adaptive capacities of a population of animats. However, such an endeavor does not go without raising specific problems (see [2] for a review), notably that of choosing how an animat's genotype relates to its phenotype.

We have argued elsewhere [3] that it might be wise to tackle these problems the same way nature does, i.e., by evolving the developmental process according to which a neural network grows and ultimately controls an animat's behavior. We have also shown that four different paradigms have been used in the past for such a purpose: rewriting rules [4], [5], axonal growth modeling [6], [7], genetic regulatory networks [8], and nested directed graphs [9]. Finally, although we concluded that it was premature to speculate about the relative merits of these paradigms — which all proved able to solve simple problems by generating control architectures implementing mere stimulus-response mechanisms — we suggested that it would be extremely helpful to test whether they would be capable of solving more complex problems by generating more cognitive architectures — which would implement, for instance, some memory or planning abilities.

This paper describes some progress we have made in this direction starting from a *de facto* benchmark, i.e., the evolution of the locomotion controller of a simulated insect. We propose an incremental methodology, according to which the developmental rules of a neural 1-D locomotion controller are first evolved using a combination of the rewriting rules and axonal growth modeling paradigms. Then, this controller is included as a module in a larger architecture, where its behavior is modulated by a second neural module that is evolved and developed to perform a higher level task, i.e., that of setting the locomotion behavior on or off in response to external tonic or phasic stimuli. In a companion paper [10], this methodology is extended to 2-D locomotion and is used to allow a simulated insect to follow up an odor gradient while avoiding obstacles.

In the following, we first review previous research efforts that have aimed at evolving walking behaviors in animats. Then, we describe our methodology and we report on experiments in which modular controllers have been evolved. The paper concludes with a discussion of the results and proposes directions for future work.

II. Previous evolutionary approaches to walking

Walking is a basic aptitude that is involved in many higher-level behaviors — like food-seeking or predator-avoidance — and that contributes to the survival of numerous animals. As it is certainly likely to contribute to the survival of numerous animats as well, it is not a surprise that several research efforts have recently aimed at evolving locomotion controllers. Most of these efforts have been based on artificial neural networks, but a few of them relied upon other paradigms like Augmented Finite State Machines, Lisp-like programs or Classifier Systems.

Although Brooks [11] did not use an evolutionary algorithm, his approach was inspired by the process of biological evolution because it consisted of incrementally adding new behavioral capabilities to an already functional architecture. In this work, a real six-legged robot was equipped with a control architecture made up of a number of Augmented Finite State Machines (AFMS). Two AFSMs per leg allowed the robot to stand while a second level of AFSMs permitted walking. Still additional levels allowed for force balancing, leg lifting, etc.

Following this work, several researchers used simulated evolution to fine tune the parameters of hand-designed controllers. De Garis [12] used a genetic algorithm [13] to evolve the weights of a fully connected neural network controlling the locomotion of a simulated biped. Using a sequence of evolutionary stages, each characterized by a different fitness function — a process he called *Sequential Evolution* —, he was able to generate realistic walking behaviors. The same methodology was then used to evolve different motion controllers for a simulated

quadruped robot — among which were controllers for straight locomotion, clockwise and anticlockwise rotation — but did not lead to clearly successful results. Nevertheless, this work was one of the first attempts to evolve a neural architecture likely to produce a number of different behaviors.

Likewise, Beer and Gallagher [14] used a genetic algorithm to evolve the parameters of a dynamic neural network that controlled the locomotion of a simulated insect. Making hypotheses on the symmetries of the controller, they carefully chose the architecture of the network so as to reduce the number of parameters to 50. All legs were driven by identical sub-networks defined by a unique set of 40 parameters. The ten remaining parameters described ipsilateral and contralateral connections between adjacent legs. Non-adjacent legs were not connected. Thus, Beer and Gallagher succeeded in evolving controllers exhibiting a fast-walking *tripod gait*, according to which the front and back legs on each side of the body swung in phase with the middle leg on the opposite side and out of phase with the other tripod.

Lewis et al. [15] combined the two previous approaches to evolve a neural controller for a real six-legged robot. In this work, fitness was assessed by the user watching to the robot's behavior, and the number of parameters was reduced to eight. In a first stage, four parameters were evolved to allow a couple of neurons to oscillate, producing a succession of power and return strikes. Then these parameters were further evolved under a different fitness function, together with four additional parameters that described the connections between adjacent legs. The authors reported that tripod gaits consistently evolved after seven to 17 generations for the first stage, plus ten to 35 generations for the second stage.

More recently some research efforts have aimed at evolving both the architecture and the parameters of locomotion controllers. Spencer [16] used genetic programming [17] to evolve Lisp-like programs (S-expressions). The task was to control a simulated six-legged animat inspired from that of Beer and Gallagher. Although Spencer claimed not to use domain knowledge, in all his experiments either an oscillator function was given, or a leg-reversal mechanism was used. However, the overall architecture of the controller — implicitly defined by the program's structure — was not given in advance but was discovered by the evolutionary algorithm.

Gruau [18] applied *cellular encoding* — i.e., an efficient instance of the rewriting rule paradigm [5] — to evolve the developmental program of an artificial neural network that controlled the locomotion of a simulated animat also inspired from that of Beer and Gallagher. Using Automatically Defined Sub-Networks, a variant of Automatically Defined Functions used in genetic programming [19], he generated a modular architecture able to control the animat. However, it took a 32-processor parallel machine and over 1,000,000 evaluations to obtain this result. In a recent report, he explored how to help the evolutionary algorithm by assessing a controller's fitness by visual inspection and by providing syntactic constraints that restricted the variety of the developmental programs generated. He thus succeeded in reducing the number of evaluations to a few hundred and in generating a locomotion controller for a real 8-legged robot in a couple of days [20].

Bull et al. [21] used Pittsburg-style classifier systems (CS) to control each leg in a four-legged simulated robot. A flexible communication protocol allowed the different controllers to exchange messages, and a complete controller was thus made of four communicating CS's. Several strategies for evolving well-performing groups of CS's were compared. A coevolutionary strategy, in which a population was evolved for each of the four types of leg controllers, appeared to be superior to a strategy where all leg controllers were mixed in a single population, or to a single-agent approach where a chromosome encoded four CS's. In this work also, the architecture was largely unspecified by the programmers, as the connections between the different controllers could be modified by the evolutionary algorithm.

From this survey of the relevant literature, it is clear that, while much research has been targeted at evolving 1-D locomotion controllers, problems like speed control, direction control, or rough terrain locomotion remain largely unsolved by evolutionary methods, not to mention higher-level tasks like obstacle-avoidance, goal-seeking or pursuit-evasion. In this paper, we try to take current results one step further in these directions by first evolving a neural network that controls walking, and then by evolving another neural network that gets connected to the first one and modulates its inner workings, in such a way as to let the animat exhibit a higher-level behavior still involving locomotion. The next section describes our SGOCE evolutionary paradigm, a simple geometry-oriented variation of Gruau's cellular encoding scheme [5], [20], and the incremental approach that we use for such a purpose.

III. The SGOCE evolutionary paradigm

In the present section we successively describe the method used to encode the developmental process of a neural network, the syntactic constraints that define the particular subset of genotypes we are considering, the evolutionary algorithm and the incremental methodology we are using.

A. Encoding scheme

Our encoding scheme is a combination of the cellular encoding and axonal growth paradigms. Here, each cell occupies a given position in a two-dimensional substrate and can make a connection with another cell either by sending an axon into, or by attracting an axon from, a given region of space. The sensors and actuators provided by the experimenter are also placed in the substrate, and are capable of connecting with any cell from the beginning of the developmental process[1].

Each animat possesses an artificial genotype, i.e., a program that describes the developmental process of an artificial neural network. Each developing cell in that animat is endowed with a copy of this program or *chromosome*, which is a set of subprograms each made of instructions that are executed by the cells during development. Such subprograms have the structure of trees with ordered branches, allowing the use of the same kind of mutation and recombination operator — by exchange of sub-

[1] Such a feature allows the generation of networks that are functional at every stage of their development. While this feature is not exploited in the current experiments, where ANNs are first developed and then evaluated, we could easily use it in future work to study how the animat's interactions with its environment can influence development.

trees — as in genetic programming. Each node in a tree is labeled by an instruction type and a variable number of parameters; it has a given number of sub-nodes that depends on its label's instruction type.

DIVIDE α r	create a new cell
GROW α r w	create a connection to another cell
DRAW α r w	create a connection from another cell
SETBIAS b	modify the bias parameter
SETTAU τ	modify the time constant parameter
DIE	trigger cellular death

TABLE I

The experimenter chooses the size of the substrate and positions the sensory cells and motoneurons that may be incorporated in the final neural network. He also positions a set of initial cells (or *precursor cells*), each of which is liable to execute a given subprogram. Finally, he associates with each precursor cell a local frame centered on that cell, according to which the geometrical specification contained in the subprograms will be interpreted. At the beginning of the developmental process, all precursor cells start executing their associated subprograms simultaneously.

The execution of a subprogram starts when a cell reads the node at its root. Whenever a cell reads a node, it executes the corresponding instruction and records in an appropriate event list that the sub-nodes of that node are to be read after a given time interval[2]. If the current instruction is a so-called *cellular division instruction*, a copy of the cell is created and, after the given interval, the daughter cell reads the right sub-node in the subprogram while the mother cell reads the left sub-node (nodes labeled by cellular division instructions have exactly two sub-nodes). A cell halts its development when it reads a node with no sub-node. From that moment it is called an interneuron.

We call *developmental instruction* an instruction that has the important side-effect of creating a new cell, of modifying a cell's parameters or of creating connections, either with another cell or with one element of a set of available sensors and actuators. The execution by a set of preexisting cells of a program containing such instructions leads to the formation of a complete ANN whose architecture may be arbitrarily complex and that may interact with the problem's environment.

As for neural dynamics, they are governed by a *leaky integrator* model that has already been used in several applications involving continuous-time recurrent neural network motion controllers [14], [22], [18], [23]. This model has the advantage of being a universal dynamics approximator [24], i.e., of being able to approximate the trajectory of any smooth dynamic system. Thus, the mean membrane potential m_i of a neuron N_i evolves according to:

$$\tau_i \cdot dm_i/dt = -m_i + \sum w_{i,j} x_j + I_i$$

where $x_j = (1+e^{-(m_j+B_j)})^{-1}$ is the neuron's short-term average firing frequency, B_j is a uniform random variable whose mean b_j is the neuron's bias, and τ_i is the time constant associated with the passive properties of N_i's membrane. I_i is the input that neuron N_i may receive from a given sensor, and $w_{i,j}$ is the synaptic weight of a connection from neuron N_j to neuron N_i.

In this paper we use a small set of general, low level, developmental instructions (Table I). A cellular division instruction (DIVIDE) makes it possible for a mother cell to generate a daughter cell. A direction parameter (α) and a distance parameter (r) associated with that instruction specify the position of the daughter cell to be created in the coordinates of the local frame attached to the mother cell. Then, the local frame associated with the daughter cell is centered on that cell and is oriented in the same way as the mother cell's frame (Figure 1). Two instructions (GROW and DRAW) respectively create one new efferent and one new afferent connection. The cell to be linked to the current one is the closest to a target position that is specified by the instruction parameters (Figure 1). However, no connection is created if the target is outside the substrate's limits. The synaptic weight of a new connection is given by the parameter w. Two additional instructions (SETTAU and SETBIAS) specify the values of a cell's time constant τ and bias b. Lastly, the instruction DIE causes a cell to die.

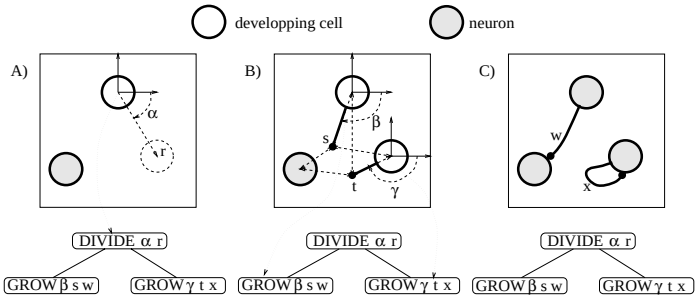

Fig. 1. The effect of a sample developmental code. A) When the upper cell executes the DIVIDE instruction, it divides. The position of the daughter cell in the mother cell's local frame is given by parameters α and r of the DIVIDE instruction, which set respectively the angle and the distance at which the daughter cell is positioned. B) Next, the mother cell reads the left sub-node of the DIVIDE instruction while the daughter cell reads the right sub-node. C) As a consequence, a connection is grown from each of both cells. The first two parameters of a GROW instruction determine a target point in the local frame of the corresponding cell. The connection is made with the cell closest to the target point — a developing cell, an interneuron, a motoneuron or a sensory cell — and its synaptic weight is given by the third parameter of the GROW instruction. Note that, in this specific example, the daughter cell being closest to its own target point, a recurrent connection is created on that cell. Finally, the two cells stop developing and become interneurons.

The DIVIDE instruction labels nodes that have exactly two sub-nodes. All other developmental instructions label nodes with no sub-node. Non-developmental instructions associated with nodes with different numbers of sub-nodes will be introduced in the next sub-section.

B. Syntactic restrictions

In order to reduce the size of the genetic search-space and the complexity of the generated networks, we constrain the structure of the programs in the population by requiring that all subprograms be well-formed trees according to a given context-free

[2] In the current implementation, all time intervals are of the same length.

Terminal symbols

DIVIDE, GROW, DRAW, SETBIAS, SETTAU, DIE, NOLINK, DEFBIAS, DEFTAU, SIMULT3, SIMULT4.

Variables

Start1, Level1, Level2, Neuron, Bias, Tau, Connex, Link.

Production rules

Start1 $\longrightarrow$ DIVIDE(Level1, Level1)

Level1 $\longrightarrow$ DIVIDE(Level2, Level2)

Level2 $\longrightarrow$ DIVIDE(Neuron, Neuron)

Neuron $\longrightarrow$ SIMULT3(Bias, Tau, Connex) | DIE

Bias $\longrightarrow$ SETBIAS | DEFBIAS

Tau $\longrightarrow$ SETTAU | DEFTAU

Connex $\longrightarrow$ SIMULT4(Link, Link, Link, Link)

Link $\longrightarrow$ GROW | DRAW | NOLINK

Starting symbol

Start1.

Fig. 2. The GRAM1 grammar. Figure 8 shows a subprogram recognized by GRAM1.

tree-grammar. Such syntactic restrictions have already been used by Koza and Rice to evolve neural networks [25]. However, while in their application such constraints were imposed to obtain valid descriptions of neural networks, here we use them to limit the sizes of the individual programs by concentrating the search on a restricted family of "interesting" programs. This approach is also similar to those of [26], [27] and is related to the notion of *strongly typed* genetic programming [28].

Figure 2 shows the grammar GRAM1 used in Section IV-A to constrain the form of the evolved subprograms that participate in the developmental process of locomotion controllers.

The set of terminal symbols consists of the developmental instructions listed in Table I and of additional *structural instructions* that have no side-effect on the developmental process. NOLINK is a "no-operation" instruction. DEFBIAS and DEFTAU leave the default value of the parameter b and τ unchanged. Those instructions label nodes of arity 0. SIMULT3 and SIMULT4 are branching instructions that allow the sub-nodes of their corresponding nodes to be executed simultaneously. The introduction of such instructions makes it possible for the recombination operator to act upon whole interneuron descriptions or upon sets of grouped connections, and thus hopefully to exchange meaningful building blocks. Those instructions are associated with nodes of arity 3 and 4, respectively.

As a consequence of the use of syntactic constraints that predefine the overall structure of a developmental program, the timing of the corresponding developmental process is constrained. First divisions occur, then cells die or parameters are set, and finally connections are grown. No more than three successive divisions can occur and the number of connections created by any cell is limited to four. Thus, the final numbers of interneurons and connections created by a subprogram that is well-formed according to GRAM1 cannot exceed eight and 32 respectively.

C. Evolutionary algorithm

In order to evolve neuro-controllers, the experimenter must supply the initial conditions for the developmental process, i.e. the size of the substrate, the positions of sensors, motoneurons and precursor cells, and the number of subprograms. Each subprogram can either be pre-specified or evolved. In the latter case, a set of syntactic constraints must be given. Finally, the experimenter provides a fitness function that evaluates the programs (see Section IV).

To slow down convergence and to favor the apparition of ecological niches, we use a steady-state evolutionary algorithm that involves a population of N randomly generated well-formed programs distributed over a circle and whose mode of operation is outlined in Figure 3.

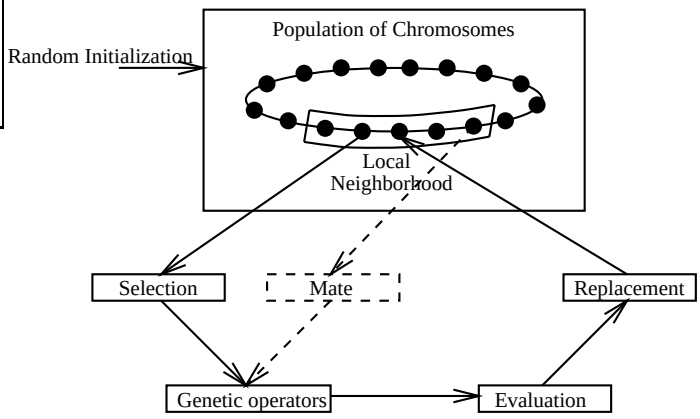

Fig. 3. The evolutionary algorithm.

The following procedure is repeated until a given number of individuals have been generated and tested:

1. A position P is chosen on the circle.
2. A 2-tournament selection scheme is applied in which the better of two programs randomly selected from the neighborhood of P is kept[3].
3. The selected program is allowed to reproduce and three genetic operators possibly modify it. The recombination operator is applied with a probability of p_c. It exchanges two *compatible*[4] sub-trees between the program to be modified and another program selected from the neighborhood of P. Two types of mutation are used. The first mutation operator is applied with a probability of p_m. It changes a randomly selected sub-tree into another compatible, randomly generated one. The second mutation operator is applied with a probability of 1. It modifies the values of a random number of parameters, implementing a *constant perturbation* strategy [16]. The number of parameters to be modified is drawn from a binomial distribution $\mathcal{B}(n, p)$.
4. The fitness of the new program is assessed by collecting statistics while the behavior of the animat controlled by the corresponding artificial neural network is simulated over a given period of time.
5. A 2-tournament anti-selection scheme, in which the less suitable of two randomly chosen programs is selected, is used to

[3] A program's probability p_s of being selected decreases with the distance d to P: $p_s(d) = max(R - d, 0)/R^2$, with $R = 4$. Programs for which d is greater than or equal to R cannot be selected ($p_s = 0$).

[4] Two sub-trees are compatible if they are derived from the same grammatical variable, like Start1, Level1, etc., in Figure 2.

decide which individual (in the neighborhood of P) will be replaced by the modified program.

In all the experiments reported on in this paper, $p_c = 0.6$, $p_m = 0.2$, $n = 6$ and $p = 0.5$.

D. Incremental methodology

We use an incremental approach that takes advantage of the geometrical nature of the developmental model.

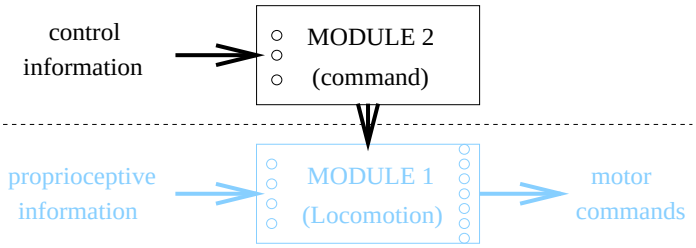

Fig. 4. During a first evolutionary stage, Module 1 is evolved. That module receives proprioceptive information through sensory cells and influence actuators through motoneurons. In a second evolutionary stage, Module 2 evolves. That module receives control information through special sensory cells called *control units* and can influence the behavior of the animat by making connections with the cells of the first module.

In a first evolutionary stage, locomotion controllers are evolved. At the end of that stage, the developmental program corresponding to the best evolved controller is selected to be the locomotion module used thereafter. During a second evolutionary stage, a second neural module is evolved. That module can influence the locomotion module by creating inter-modular connections. Figure 4 shows the information processed by each module.

IV. Experimental results

The experiments presented in this paper made use of a model of a six-legged animat called SWAN-1D [29].

Each the animat's leg was equipped with two pairs of muscles that allowed them to control the angular position of the leg and the height of the foot (Figure 5, Left). For three of those muscles, a corresponding motoneuron specified the value of the resting length parameter in a simple muscle model (Figure 5, Right). Furthermore, each leg was equipped with a sensor that measured the leg's angular position θ.

Thus the available motors and sensors corresponded to those of Beer and Gallagher's simulated insect [14]. However, one difference with Beer and Gallagher's scheme was that the foot status (up or down) was not determined solely by the state of the corresponding UP-motoneurons. More realistically, these positions were also influenced by the dynamics of the physical model of the animat. Our animat also differed from that of Gruau [18], who used anterior and posterior extreme position sensors instead of angle sensors.

A. Evolution of locomotion controllers

In the first evolutionary stage, locomotion controllers for simulated six-legged animats were evolved. In order to reduce the size of the search space, we sought controllers made of six sub-

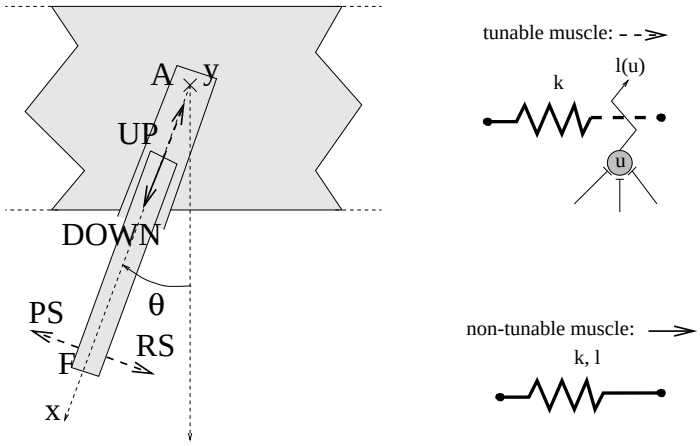

Fig. 5. Left: Each leg has two degrees of freedom. Thanks to the antagonistic PS- (Power Strike) and RS- (Return Strike) muscles, the leg can rotate around an axis (Ay) orthogonal to the plane of the figure. The UP- and DOWN-muscles allow the position of the foot F to be translated along the (Ax) axis. The resting lengths of the tunable PS-, RS- and UP-muscles depend on the activity levels of the corresponding motoneurons u. The resting length of the DOWN-muscle is supposed to be non-tunable. Right: The muscle model. A muscle is modeled as a spring of set stiffness k and of (possibly tunable) resting length l (after [30]).

networks grown according to the instructions of a unique developmental subprogram[5].

Figure 6 shows the setting of the two-dimensional substrate when the developmental process was initialized. Six precursor cells called six associated subprograms (dotted lines) that, in turn, each called subprogram 6. The positions and the local frames of the different precursor cells reflected the assumed bilateral symmetry of the animat's morphology. The motoneurons and sensory cells of each leg had specific coordinates in the local frame associated with the corresponding precursor cell. The execution of the whole developmental program resulted in the creation of a neuro-controller made of six interconnected subnetworks. According to such a logic, only subprogram 6 had to be evolved.

The fitness function was the distance covered during the evaluation increased by a term encouraging any leg motion:

$$f = x(T_{max}) + \int_{t=0}^{T_{max}} (\sum_p |\frac{d\theta_p}{dt}(t)| + \sum_p |\frac{dh_p}{dt}(t)|)dt$$

where $x(t)$ is the position of the animat's center of mass at time t, T_{max} is the evaluation time, and $\theta_p(t)$ and $h_p(t)$ are the angular position and the height of leg p at time t [29]. We did not introduce explicit selection pressure for not falling. However, falls were implicitly penalized because they slowed down locomotion.

We performed a series of five experiments. In each experiment, 100,000 replacements were made in a population of 200 programs with different, randomly-generated subprograms 6, well-formed according to GRAM1.

Several kinds of walking strategies were obtained. In four experiments, symmetrical gaits — in which both sides were moved

[5] In this approach, although the same developmental subprogram was called six times, the corresponding sub-networks might differ due to side-effects. In Beer and Gallagher's experiment, a more stringent constraint imposed the symmetry of the overall architecture.

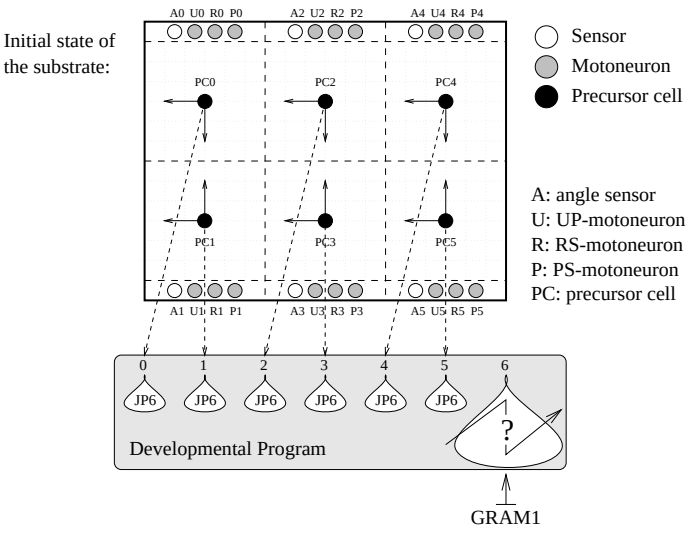

Fig. 6. Setup for the evolution of a straight locomotion controller for a six-legged animat. The figure shows the initial positions of the sensors, motoneurons and precursor cells, as well as the structure of the developmental programs that call upon seven subprograms. JP is a call instruction that forces a cell to start reading a new subprogram. Only subprogram 6 needs to be evolved. It's organization is constrained by the GRAM1 grammar. Additional details are to be found in the text.

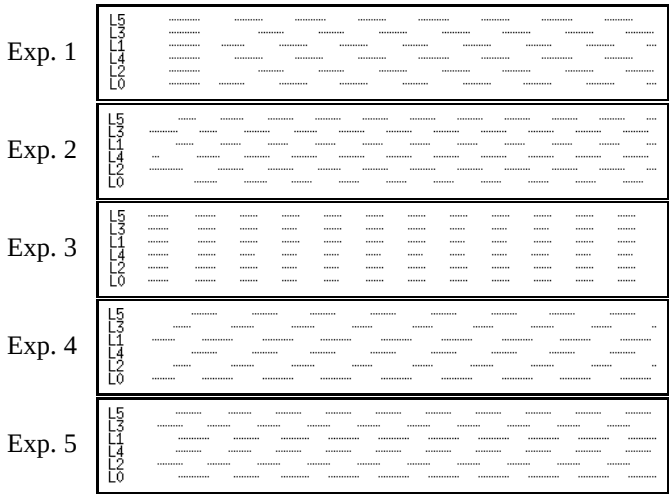

Fig. 7. Best gait in the final population for each of the five experiments. The horizontal axis represents time. A dot is plotted when the corresponding leg is raised. Legs are numbered as in Figure 6. Only the results of Experiment 2 correspond to a stable, tripod gait. All other experiments in the series led to unstable, leaping behaviors.

synchronously — were generated. The corresponding behaviors consisted in making a succession of leaps, using groups of two, four or six legs together. Such solutions were the most likely to evolve both in the series of experiment reported here and in others as well. None of those gaits was stable because of the large mass of the modeled body. However, in the course of Experiment 2, a stable, non-symmetric tripod gait was obtained (Figure 7). This solution allowed the longest distance to be covered during the given evaluation time.

External feedback provided by the sensors was used only by the controllers that evolved during Experiment 5. Besides the possible incidence of initial geometrical constraints that would lessen the chances of such sensors being incorporated into the control architecture, this is due to the fact that the simple environment used (flat ground, no obstacles) did not really necessitate the use of sensors. Furthermore, the controllers of Experiment 5 appeared to be the most sensitive to starting conditions: some initial leg positions could not trigger subsequent periodic activity in the network, presumably because specific sensor values were needed. In the other experiments, whatever the initial leg positions, intrinsic periodic activity was produced by *central pattern generators* that are known to exist in arthropods and that contribute to the generation of rhythmic locomotion movements [31].

Figure 8 shows the subprogram 6 and the architecture of the corresponding network for the best individual found in Experiment 2. A part of the circuitry that is responsible for the control of foot positions in that network is shown in Figure 9 and helps to understand its inner workings. Two pairs of oscillators, associated with the hind- and middle-legs, are coupled together.

Simulating either pair in isolation with the corresponding coupling connections (from U_i to $b_{opp(i)}$ and from a_i to $U_{opp(i)}$ where i is one oscillator and $opp(i)$ is the other oscillator of the same pair) results in the two corresponding legs oscillating out of phase with each other.

Adding the coupling connections between the two pairs (from U_i to $b_{fopp(i)}$, where i is a hind-leg oscillator and $fopp(i)$ is the contralateral middle-leg oscillator) makes adjacent legs oscillate out of phase with each other.

Finally, reintroducing the connections from c_4 and c_5 to U_0 and U_1 respectively synchronizes the front-legs with the hind-legs, producing a tripod pattern for the UP-motoneurons. It appears that the periodic activity of the other motoneurons is also produced by the same four oscillators, and additional details on how that particular network functions can be found in [32].

This network was selected to be Module 1 in the second evolutionary stage where two kinds of Module 2 were evolved. In Section IV-B, we report on experiments in which we looked for a control mechanism such that the animat walked as long as a boolean control unit received the value False and stopped whenever that unit received the value True. In Section IV-C, we describe another experiment where two control units were considered. The animat had to stop walking whenever the first unit was briefly stimulated and had to resume walking whenever the second unit was briefly stimulated.

B. Evolution of a command module

In this section, we let a two-module neural network to evolve that is able to generate walking or resting according to the value of a tonic boolean command input. The input value is set by the experimenter and is communicated to the system through a specific control unit.

Assuming that a simple architecture would solve the task, the precursor cells of the second module were connected by default to the control unit at the beginning of the developmental process by *ad hoc* DRAW instructions. These cells were not allowed to divide, to create other intra-modular connections or to modify their bias parameters. Thus, only inter-modular connections toward Module 1 were allowed. Under such conditions, the only task of the evolutionary process was to find a set of connections

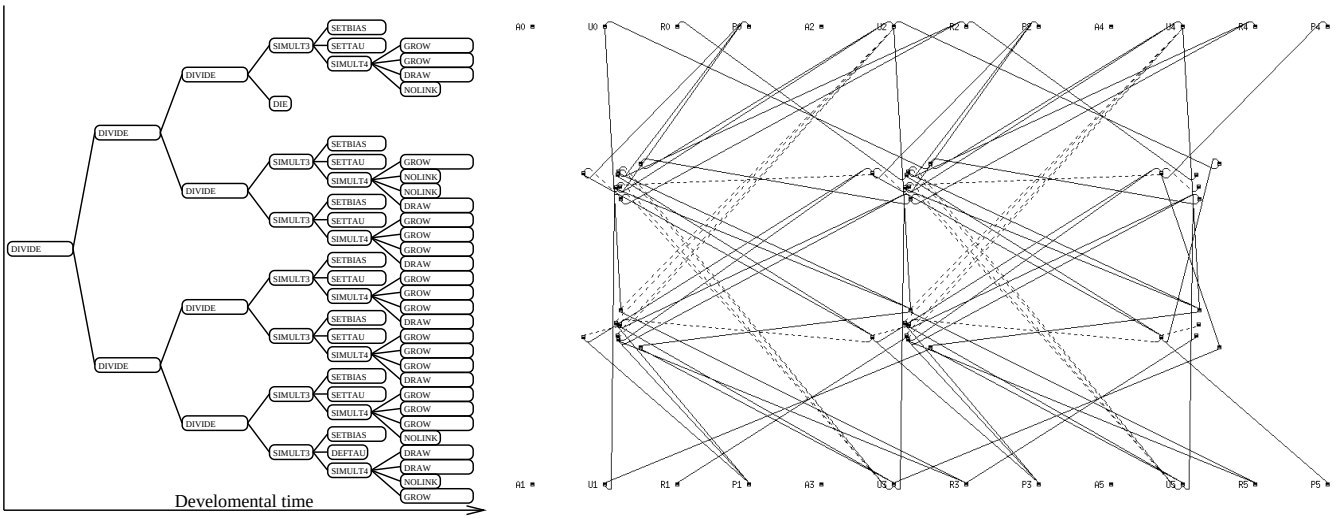

Fig. 8. Left: The best subprogram 6 found in Experiment 2 (parameter values are not shown). This subprogram generates a tripod gait and is called LOCO1 thereafter. Right: The corresponding artificial neural network after useless interneurons and connections have been pruned. Solid lines are excitatory connections, dotted lines are inhibitory connections. Fan-in connections arrive at the top and fan-out connections depart from the bottom of each neuron. The network contains 38 interneurons and 100 connections.

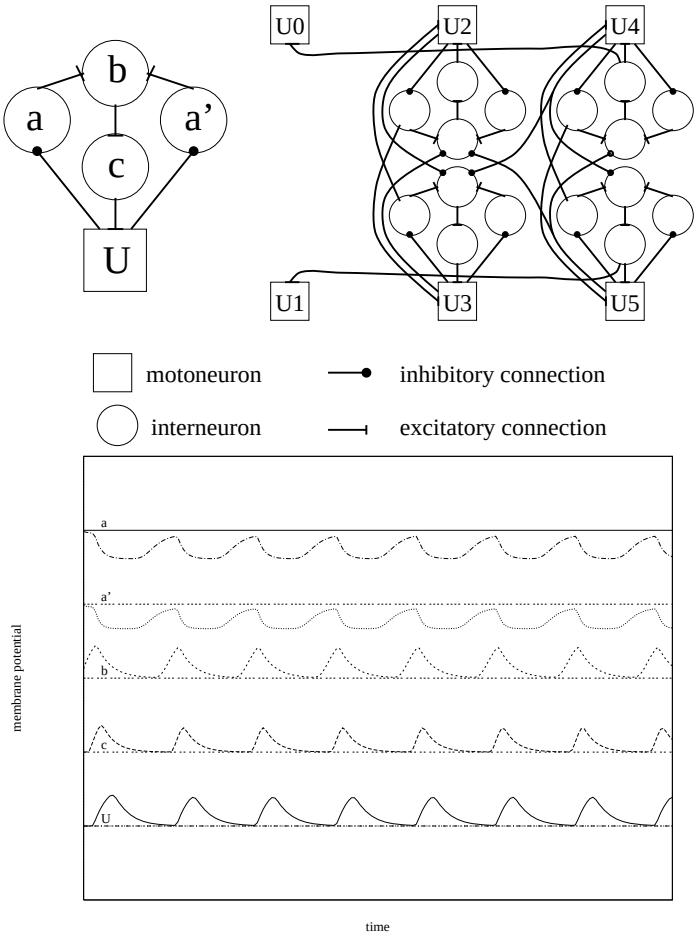

Fig. 9. Part of the circuitry that is responsible for foot-position control. A basic oscillator (Upper, Left) is copied four times within the control architecture. These oscillators are connected in such a way that the feet of adjacent legs are raised out of phase with each other (Upper, Right). When the basic oscillator is simulated in isolation, the membrane potentials of its different neurons oscillate as shown at the bottom.

Terminal symbols

GROW2, NOLINK, SIMULT8.

Variables

Start2, Link2.

Production rules

Start2 $\longrightarrow$ SIMULT8(Link2, Link2, Link2, Link2, Link2, Link2, Link2, Link2)

Link2 $\longrightarrow$ GROW2 | NOLINK

Starting symbol

Start2.

Fig. 10. The GRAM2 grammar.

able to inhibit locomotion behavior when the value of the command input was maximal (True). Whenever the command input value was zero (False), the neurons of the second module were not activated — because of the specific default value of their bias parameter — and the first module generated the default locomotion behavior.

A new developmental instruction (GROW2) was used to create a connection between a cell in the second module and a cell in the locomotion module. This instruction works like the GROW instruction except that the geometric parameters are interpreted in the local frame's orthogonal projection into the locomotion module[6]. The GRAM2 grammar (Figure 10) defined a set of well-formed subprograms liable to create a number of connections (maximally 8) from a precursor cell of the second module into the locomotion module.

Figure 11 shows the initial conditions for the developmental process and the general structure of the programs.

During an evaluation, the value of the command input was successively set to False, True, False, True and False. The fitness function rewarded individuals for not moving and for standing when the command was True:

[6] The second module has the same dimensions as the first and is considered to be positioned above it.

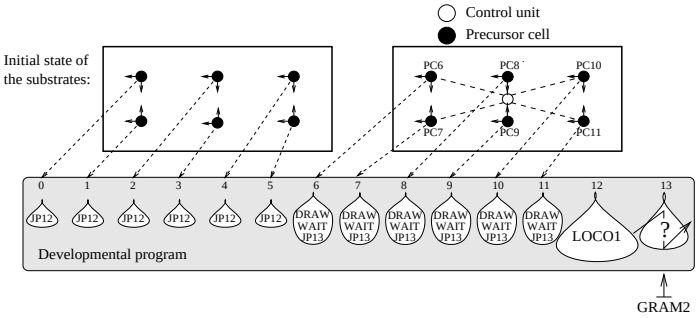

Fig. 11. Setup for the evolution of a command network. The figure shows the initial positions of the control unit and the precursor cells, as well as the structure of the developmental programs that calls upon 14 subprograms. DRAW instructions (followed by appropriate parameters) are added to create excitatory connections (dashed lines) from the control unit to precursor cells 6 to 11. WAIT is a no-operation instruction used to delay the call of subprogram 13. This delay allows time for the three successive divisions of precursor cells 0 to 5 to occur before instruction GROW2 can be executed by precursor cells 6 to 11. Sub-program 12 has been evolved in the previous experiments; only subprogram 13 needs to be evolved.

$$f = \int_{t=0}^{T_{max}} r(t) \cdot dt$$

$r(t) = (k \cdot s(t) - |v(t)|)$ if True and $r(t) = 0$ otherwise;

where $v(t)$ is the speed of the animat's center of mass at time t, k is a weighting coefficient set to 0.01 in the experiments described herein, and $s(t)$ is 1 if the animat is stable at time t and 0 otherwise. No reward was granted while the command was False.

We carried out five experiments in which 20,000 replacements were performed in a population of 200 individuals. In each experiment, highly rated controllers were found. Figure 12 illustrates the corresponding STOP and GO behaviors.

To check whether such controllers could generate intermediate speeds between fast-walking and resting, we subjected them to intermediate command values. This strategy can be compared to that used in [33] to evolve so-called *steerable GenNets*. However, instead of checking for interpolation in a controller previously evolved to exhibit two qualitatively identical behaviors (e.g. two locomotion behaviors characterized by two different speeds), we left the previously evolved controller unchanged and submitted it to continuous command values.

Results shown in Figure 13 indicate that, when the control unit is clamped to a value comprised between (about) 0.2 and 0.5, the animat walks at a reduced speed. Beyond 0.5, walking is inhibited. Observation of the behavior reveals that this result is due to a decrease in the animat's step size, and not to a change in the rhythm of its basic oscillators. Closer inspection of the inner workings of the controllers gives some insight into how step size is reduced. It thus turns out that the corresponding mechanisms involve inhibitory connections from Module 2 to the c neurons of the oscillators of Figure 9, in four experiments out of five, or to the PS- and RS- motoneurons directly, in the fifth experiment. It can easily be checked that the inhibition of these neurons within an isolated oscillator results in a reduction of its output amplitude, but in no significant frequency variation.

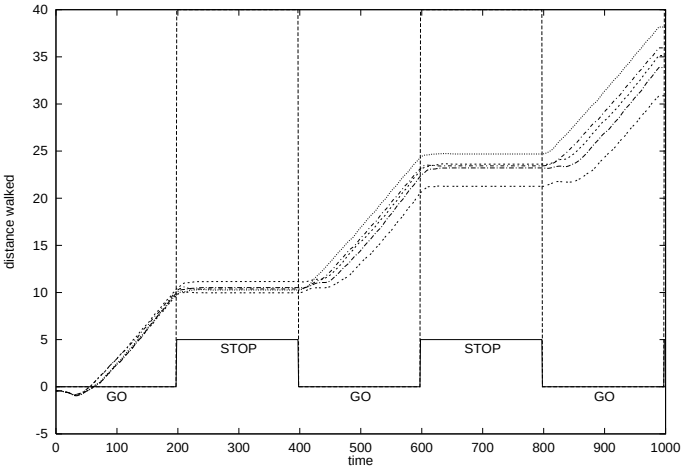

Fig. 12. Distance covered as a function of time by the best controllers in the final population for five different experiments. The boolean command input is True between cycles 200 and 400 and between cycles 600 and 800.

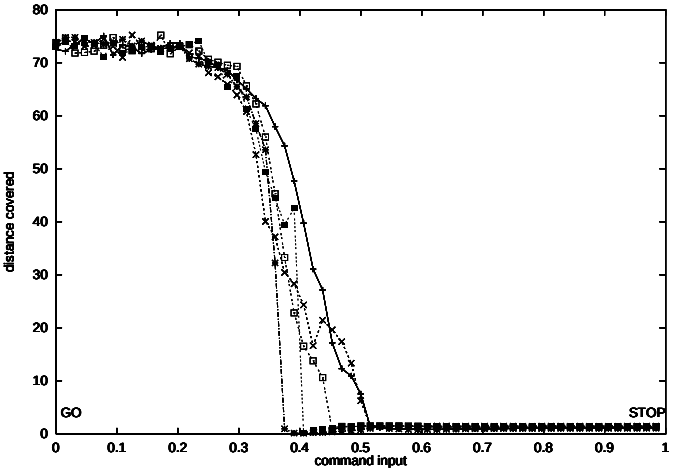

Fig. 13. The distance covered during 1000 cycles when a fixed continuous command value is applied. Each curve represents the mean distance covered in ten runs.

C. Evolution of a switching mechanism

In this section, again using the locomotion module evolved in Section IV-A, we evolve a new Module 2 that can respond to two phasic stimuli, $S1$ and $S2$, by switching either to a resting or to a walking behavior. These stimuli are delivered by the experimenter through two specific control units.

For this task, we allowed intra-modular divisions and connections inside Module 2. No ad-hoc connections were imposed and each bias was allowed to evolve. A new developmental instruction called DRAW2 was introduced. DRAW2 causes the creation of an afferent connection from a cell of Module 1 to the executing cell, and works like the DRAW instruction except that it is interpreted in the orthogonal projection of the local frame into Module 2. Furthermore only two precursor cells were placed in Module 2. Under such conditions, the walking behavior generated by Module 1 was liable to be perturbed even in the absence of control signals.

The GRAM3 grammar (Figure 14) defined the set of valid subprograms that described the developmental process of a pre-

Terminal symbols

DIVIDE, GROW, DRAW, GROW2, DRAW2, SETBIAS, SETTAU, DIE,
NOLINK, DEFBIAS, DEFTAU, SIMULT3, SIMULT4.

Variables

Start3, Levelb, Neuronb, Biasb, Taub, Connexb, Linkb.

Production rules

Start3 $\longrightarrow$ DIVIDE(Levelb, Levelb)
Levelb $\longrightarrow$ DIVIDE(Neuronb, Neuronb)
Neuronb $\longrightarrow$ SIMULT3(Biasb, Taub, Connexb) | DIE
Biasb $\longrightarrow$ SETBIAS | DEFBIAS
Taub $\longrightarrow$ SETTAU | DEFTAU
Connexb $\longrightarrow$ SIMULT4(Linkb, Linkb, Linkb, Linkb)
Linkb $\longrightarrow$ GROW | DRAW | GROW2 | DRAW2 | NOLINK

Starting symbol

Start3.

Fig. 14. The grammar GRAM3.

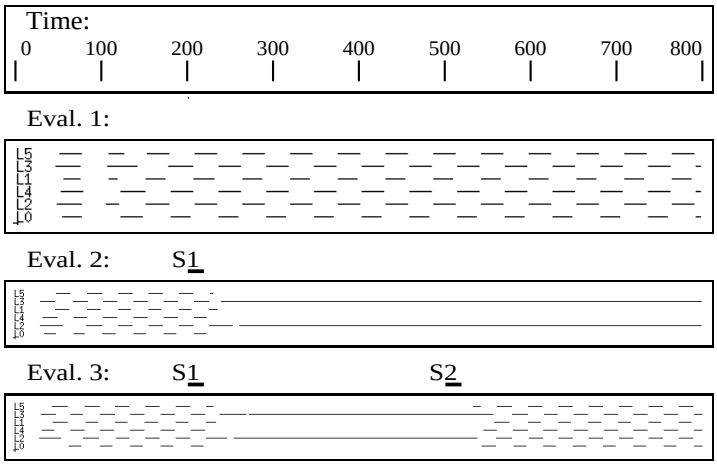

Fig. 16. Behavior of a good individual in the three phases of the evaluation. That individual responds correctly to both stimuli. The conditional protocol (described in the text) prevents the animats from just predicting the time of occurence of the stimuli.

cursor cell of Module 2. Such subprograms can create up to four neurons and 16 connections. Figure 15 depicts the initial setup for the developmental process.

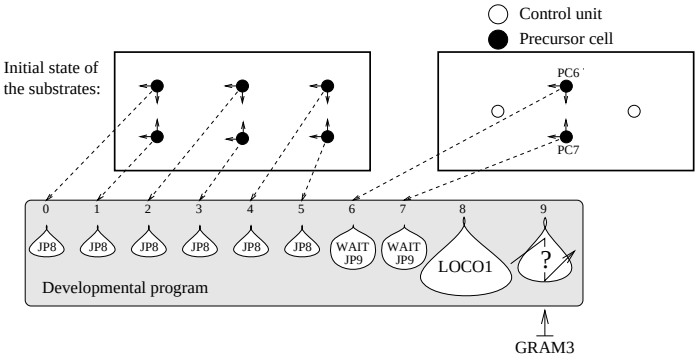

Fig. 15. Setup for the evolution of a switching mechanism. The figure shows the initial positions of the control units and the precursor cells, as well as the structure of the developmental programs that call upon ten subprograms. Sub-program 8 has been evolved in the previous experiments. Only subprogram 9 needs to be evolved.

In order to evolve a switching mechanism, we had to design a conditional evaluation procedure in which each individual could be evaluated up to three times in different conditions and with different fitness functions. This was necessary to prevent the networks from learning to predict the time of occurrence of the stimuli. An alternative solution to this problem would have been to present the stimuli at variable times. However such a solution would have made fitness comparisons less robust because different individuals would have been evaluated in different conditions.

In the first evaluation, we checked that the individual had not lost its walking ability:

$$f = \int_{t=0}^{T_{max}} v(t) \cdot dt = x(T_{max})$$

Provided the corresponding individual walked along a minimum distance, he was allowed to go through the next evaluations. In the second evaluation, stimulus $S1$ was presented on the first control unit, and the animat had to stop its progression. It was rewarded according to the previously used fitness function:

$$f = \int_{t=T_{S1}}^{T_{max}} (k \cdot s(t) - |v(t)|) \cdot dt$$

Finally, if the individual received a high enough rating, it was allowed to undergo the third evaluation. During this last evaluation, stimulus $S2$ was presented on the second control unit some time after stimulus $S1$ had been presented on the first control unit and the animat was thereafter rewarded for resuming walking:

$$f = \int_{t=T_{S2}}^{T_{max}} v(t) \cdot dt$$

We made several experiments involving populations of 200 programs. After 60.000 replacements, controllers able to respond correctly to both kinds of stimuli were obtained. Figure 16 shows the behavior of a network of the final population in the corresponding experiment. Some experiments, although leading to successfull results at evaluations 1 and 2, did not yield to animats able to react correctly to $S2$. We suspect that this is due to the fact that GRAM3 was too restrictive. Consequently, the allowed numbers of neurons and connections per neuron were too low and all of them were recruited to control the response to $S1$.

The examination of successful controllers indicates that they use the neurons of Module 2 to build switch mechanisms, which can be in one of two stable states, quiescent or excited. Such mechanisms can be implemented by a single neuron that has a sufficiently strong self-connection or by a small network of interconnected neurons, as demonstrated in [24]. According to the current switch state, the animat executes either a resting or a walking behavior. The presentation of $S1$ forces the switch into the excited, "resting" state, thanks to excitatory connections, while the presentation of $S2$ forces it back into the quiescent, "walking" state by making use of inhibitory connections (Figure 17).

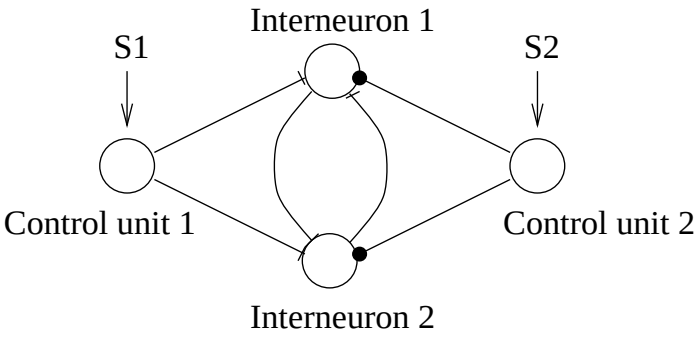

Fig. 17. A solution discovered by the evolutionary algorithm to implement a switch mechanism. Inhibitory and excitatory connections are represented as in Figure 9. In the absence of significant input, both interneurons have a low output value. Whenever $S1$ is presented, however, they are excited and the reciprocal excitatory connections keep their output values high. Finally, whenever $S2$ occurs, they are inhibited and return to their quiescent state.

V. Discussion

Results obtained here and elsewhere [10] demonstrate that an incremental methodology can be used to automatically design an animat's control system that merges low-level controllers into higher-level adaptive architectures. Such a methodology implies that low-level controllers be first evolved under the effect of appropriate fitness functions, and then that these controllers be fixed and protected against drastic modifications that mutations or other genetic operators might generate. Higher-level architectures can later be evolved and modulate the inner workings of the low-level controllers, thus making it possible to cope with the constraints of new fitness functions. Clearly, such a methodology bears a strong resemblance to that of Brooks [11] for the hand-design of so-called *subsumption architectures*. It is likely to be efficient for the sort of reasons put forth by Dawkins [34] in his comparison of the working strategies of the blind watchmakers: without a convenient means of protecting useful inner-mechanisms from deleterious mutations, there is no chance of having as complex an integrated whole as a clock evolve from scratch. Such a methodology might also be well adapted to curing the unfortunate consequences of the well-known opportunistic capacities of any evolving process: the fitness function likely to select a complex behavior from scratch may be extremely hard for a human to design and may offer numerous opportunities for the evolutionary process to follow unexpected trajectories. By decomposing the overall fitness function into successive components, each easier to design, one can hope to channel the evolutionary path. Finally, this methodology also bears a strong resemblance to the strategy of incremental evolution advocated by de Garis [12] and Harvey et al. [35], which suggests that, in order to evolve controllers to achieve some challenging task, it is better to start from a population that has already been selected for a similar but less challenging task, rather than starting with a population of random genotypes.

Be that as it may, the SGOCE paradigm appears to be well suited to the incremental methodology advocated here. Associating an indirect encoding scheme with the evolutionary process clearly reduces the size of the genotype space explored by the genetic algorithm, while leaving opportunities for the generation of complex phenotypes. Resorting to syntactic constraints on the structure of the genotypes allows us to limit the size of the search space. Although such constraints have been set arbitrarily by the experimenter in the present work, they could also be coded in a second chromosome that would evolve in parallel with the chromosome coding for the developmental program. However, such metarules would probably be long to evolve. Finally, besides having functional advantages already stressed by several authors [14], [36], [37], [24], dynamic neural networks appear to be well suited to the use of low-level developmental instructions, such as cell division and axonal growth processes, that facilitate the automatic search for useful architectures. The very general set of developmental instructions we devised should make it easy to apply our methodology to other problems.

Concerning the specific results obtained here, it appears that the SGOCE paradigm made it possible to go further than any previous comparable attempt at automatically designing a 6-legged animat's control architecture. Not only has a tripod-gait controller been generated, but this controller has been included in higher-level architectures capable of slowing down, stopping, or resuming walking. Moreover, to take into account information contributed by transient phenomena, the evolutionary process has been committed to inventing a switch mechanism, i.e., a rudimentary form of memory that somehow encodes knowledge about the world. One might argue this constitutes a first step towards the invention of representations and truly cognitive mechanisms, and that this step parallels other advances already made in this direction with artificial evolutionary processes [38].

It should be stressed that the organization of the tripod-gait controller that has been evolved and used in this work is certainly heavily dependent upon the specific implementation of the SGOCE paradigm that we used. In particular, although in previous attempts [39] we succeeded in evolving tripod-gait walking without using syntactic constraints, the corresponding neural networks commonly comprised several hundred neurons and connections, thus tremendously slowing down the simulations. Therefore, we chose to constrain the complexity of the neural networks generated, maybe with the risk of restricting their adaptive capacities, because complex neural networks are likely to exhibit many functional redundancies. In this perspective, it would be enlightening to compare the robustness of various locomotion controllers with respect to various accidents ranging from neuron or connection suppressions to whole leg amputations. A study of the effect of neuron losses on the controller LOCO1 evolved in Section IV-A has revealed such redundancies [32].

Likewise, the solutions described herein were certainly restricted by the corresponding initial setups. In particular, according to their respective positions in the substrate, some cells had a better chance of getting connected to each other than did others. This, in particular, was the case with sensors, motoneurons and precursor cells whose initial positions were set by the experimenter and that were, therefore, more or less likely to be incorporated into the final, developed neural network. A possible way of combating the negative consequences of an experimenter's arbitrary choice is to let some aspects of an animat's morphology evolve in parallel with its control architecture, an approach already explored in [35], [40], [41], [9].

Finally, at this stage of our work, it is hard to draw any conclusion about the efficiency of the evolutionary algorithm used here. We happened on these specific settings after numerous trials and errors, which aimed at preserving over generations the diversity of the fitness distribution in the population of chromosomes. Whether or not the results we obtained were optimal in this respect will have to be ascertained through systematic comparisons that we haven't yet had the opportunity to perform. It seems, however, that an important implementation decision has been the addition of a sort of constant disturbance strategy [16], according to which several parameters were mutated each time a developmental program was reproduced. Indeed, such a strategy allowed a better exploration of the parameter space in the absence of a learning algorithm.

VI. Conclusion

It has been shown here that the current implementation of the SGOCE evolutionary paradigm makes it possible to automatically design the control architecture of a six-legged animat capable not only of straight walking according to a tripod gait, but also of slowing-down, stopping or resuming walking when it receives appropriate tonic or phasic commands. It is also shown elsewhere [10] that such an approach can be extended to 2-D locomotion and that it is likely to automatically generate the control architecture of an animat capable of following up an odor gradient and avoiding obstacles. We argue that such results provide marked improvements over current state-of-the-art in the automatic design of locomotion controllers. They rely upon specific mechanisms implementing the developmental process of a recurrent dynamic neural network and upon an incremental strategy that amounts to setting the architecture of functional sub-networks in a still evolving, higher-level control system. There are numerous ways of improving the corresponding mechanisms, in particular by letting several characteristics evolve that were arbitrarily set here by the experimenter. There is also good reason to believe that the SGOCE paradigm will prove capable of automatically generating control architectures that implement more than mere stimulus-response pathways or central pattern generators and that exhibit genuinely cognitive abilities.

References

[1] J.-A. Meyer and A. Guillot, "From SAB90 to SAB94: Four years of animat research," in *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, eds.), The MIT Press/Bradford Books, Cambridge, MA, 1994.

[2] M. Matarić and D. Cliff, "Challenges in evolving controllers for physical robots," *Robotics and Autonomous Systems*, vol. 19, pp. 67–83, 1996.

[3] J. Kodjabachian and J.-A. Meyer, "Evolution and development of control architectures in animats," *Robotics and Autonomous Systems*, vol. 16, pp. 161–182, December 1995.

[4] E. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," Master's thesis, Dept. of CS and Exp. and The. Psy., Leiden University, August 1992.

[5] F. Gruau, *Synthèse de Réseaux de Neurones par Codage Cellulaire et Algorithmes Génétiques*. Thèse d'université, ENS Lyon, Université Lyon I, January 1994.

[6] J. Vaario, *An Emergent Modeling Method for Artificial Neurol Networks*. PhD thesis, University of Tokyo, August 1993.

[7] A. Cangelosi, D. Parisi, and S. Nolfi, "Cell division and migration in a 'genotype' for neural networks," *Network: computation in neural systems*, 1995.

[8] F. Dellaert and R. Beer, "Toward an evolvable model of development for autonomous agent synthesis," in *Proceedings of the Fourth International Workshop on Artificial Life* (R. A. Brooks and P. Maes, eds.), The MIT Press/Bradford Books, Cambridge, MA, 1994.

[9] K. Sims, "Evolving 3D morphology and behavior by competition," in *Proceedings of the Fourth International Workshop on Artificial Life* (R. A. Brooks and P. Maes, eds.), The MIT Press/Bradford Books, Cambridge, MA, 1994.

[10] J. Kodjabachian and J.-A. Meyer, "Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects," 1997. Submitted for publication.

[11] R. A. Brooks, "A robot that walk: Emergent behavior form a carefully evolved network," *Neural Computation*, vol. 1, no. 2, pp. 253–262, 1989.

[12] H. de Garis, *Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos*. PhD thesis, Université Libre de Bruxelles, Belgium, 1991.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[14] R. Beer and J. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adaptive Behavior*, vol. 1, no. 1, pp. 91–122, 1992.

[15] M. A. Lewis, A. H. Fagg, and A. Solidum, "Genetic programming approach to the construction of a neural network for control of a walking robot," in *IEEE International Conference on Robotics and Automation*, (Nice, France), pp. 2618–2623, 1992.

[16] G. Spencer, "Automatic generation of programs for crawling and walking," in *Advances in Genetic Programming* (K. E. K. Jr., ed.), pp. 335–353, The MIT Press / Bradford Books, Cambridge, MA, 1994.

[17] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[18] F. Gruau, "Automatic definition of modular neural networks," *Adaptive Behavior*, vol. 3, no. 2, pp. 151–184, 1994.

[19] J. Koza, *Genetic Programming II: Automatic Discovery of Reusable Subprograms*. The MIT Press, 1994.

[20] F. Gruau and K. Quatramaran, "Cellular encoding for interactive evolutionary robotics," tech. rep., University of Sussex, School of Cognitive Sciences, EASY Group, Brighton, UK, 1996.

[21] L. Bull, T. C. Fogarty, and M. Snaith, "Evolution in multi-agent systems: Evolving communicating classifier systems for gait in a quadrupedal robot," in *Proceedings of the Sixth International Conference on Genetic Algorithms* (L. J. Eshelman, ed.), pp. 382–388, Morgan Kaufmann, San Mateo, CA, 1995.

[22] D. Cliff, I. Harvey, and P. Husbands, "Explorations in evolutionary robotics," *Adaptive Behavior*, vol. 2, no. 1, pp. 73–110, 1993.

[23] D. Cliff and G. F. Miller, "Co-evolution of pursuit and evasion ii: Simulation methods and results," in *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, eds.), The MIT Press/Bradford Books, Cambridge, MA, 1996. Submitted.

[24] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–510, 1995.

[25] J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for neural networks," in *an IEEE International Joint Conference on Neural Networks*, pp. II–397–II–404, 1991.

[26] F. Gruau, "Artificial cellular development in optimization and compilation," in *Evolvable Hardware'95* (E. Sanchez and Tomassini, eds.), Lecture Notes in Computer Science, Springer Verlag, 1996.

[27] S. M. Lucas, "Evolving neural network learning behaviours with set-based chromosomes," in *ESANN'96*, 1996.

[28] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[29] J. Kodjabachian, "Simulating the dynamics of a six-legged animat," tech. rep., AnimatLab, ENS, Paris, 1996.

[30] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of Neural Science*, ch. 36: Muscles, Effectors of the Motor Systems. Prentice Hall International Inc., third ed., 1991.

[31] F. Delcomyn, "Factors regulating insect walking," *Annual Review of Entomology*, vol. 30, pp. 239–256, 1985.

[32] J. Kodjabachian, "Analysis of a neural locomotion controller found by simulated evolution," 1997. In preparation.

[33] H. de Garis, "Steerable GenNets: The genetic programming of steerable behaviors in GenNets," in *Toward a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life* (P. Bourgine and F. J. Varela, eds.), pp. 272–281, The MIT Press, Cambridge, MA, 1991.

[34] Dawkins, *The blind watchmaker*. Longman Scientific & Technical, Essex, England, 1986.

[35] I. Harvey, P. Husbands, and D. Cliff, "Seeing the light: Artificial evolution, real vision," in *From Animals to Animats 3. Proceedings of the Third*

International Conference on Simulation of Adaptive Behavior (D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, eds.), pp. 392–401, The MIT Press/Bradford Books, Cambridge, MA, 1994.

[36] P. Husbands, I. Harvey, and D. T. Cliff, "Analysing recurrent dynamical networks evolved for robot control," in *Proceedings of the Third IEE International Conference on Artificial Neural Networks*, IEE Press, London., 1993.

[37] B. Yamauchi and R. Beer, "Integrating reactive, sequential, and learning behavior using dynamical neural networks," in *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, eds.), pp. 382–391, The MIT Press/Bradford Books, Cambridge, MA, 1994.

[38] R. D. Beer, "Toward the evolution of dynamical neural networks for minimally cognitive behavior," in *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, eds.), pp. 421–429, The MIT Press/Bradford Books, Cambridge, MA, 1996.

[39] J.-A. Meyer, "From natural to artificial life: Biomimetic mechanisms in animat design," *Robotics and Autonomous Systems*, 1997. In press.

[40] C. W. Reynolds, "Evolution of corridor following behavior in a noisy world," in *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, eds.), pp. 402–410, The MIT Press/Bradford Books, Cambridge, MA, 1994.

[41] K. Sims, "Evolving virtual creatures," in *Computer Graphics Proceedings, Annual Conference Series*, pp. 15–23, 1994.