Evolutionary Approaches to Neural Control of Rolling, Walking, Swimming and Flying Animats or Robots

Jean-Arcady Meyer, Stéphane Doncieux, David Filliat and Agnès Guillot

AnimatLab – Lip6

8 rue du Capitaine Scott

75015 Paris

France

e mail: jean-arcady.meyer@lip6.fr

telephone: +33 1 44 27 88 09

fax: +33 1 44 27 70 00

Abstract

This article describes past and current research efforts in evolutionary robotics that have been carried out at the AnimatLab, Paris. Such approaches entail using an artificial selection process to automatically generate developmental programs for neural networks that control rolling, walking, swimming and flying animats or robots. Basically, they complement the underlying evolutionary process with a developmental procedure – in order hopefully to reduce the size of the genotypic space that is explored – and they occasionally call on an incremental approach, in order to capitalize upon solutions to simpler problems so as to devise solutions to more complex problems. This article successively outlines the historical background of our research, the evolutionary paradigm on which it relies, and the various results obtained so far. It also discusses the potentialities and limitations of the approach and indicates directions for future work.

1. Introduction

From a recent review (Meyer 1998) of evolutionary approaches to neural control in mobile robots, it appears that the corresponding research efforts usually call upon a direct coding scheme, where the *phenotype* of a given robot – i.e., its neural controller and, occasionally, its body plan – is directly coded into its *genotype*. However, it has often been argued (e.g., Husbands et al. 1994; Kodjabachian and Meyer 1995) that indirect coding schemes – where the genotype actually specifies developmental rules according to which complex neural networks and morphologies can be derived from simple programs - are more likely to scale up with the complexity of the control problems to be solved. This should be the case, if only because the size of the genotypic space to be explored may be much smaller than the space of the resultant phenotypes, although it is true that opposite opinions have been expressed (Conrad 1990; Harvey 1992; Shipman 1999; Bongard and Paul 2001).

The feasibility of such indirect approaches, which combine the processes of evolution and development, has been demonstrated through several simulations (Boers and Kuiper 1992; Cangelosi et al. 1995; Dellaert and Beer 1994; Gruau 1995; Kodjabachian and Meyer 1998a, b; Nolfi et al. 1994; Sims 1994; Vaario 1993; Vaario et al. 1997) and a few applications involving real robots (Eggenberger 1996; Jakobi 1997a,b; Gruau and Quatramaran 1997; Michel 1995; Michel and Collard 1996). However, the fact that the great majority of controllers and behaviors thus generated are exceedingly simple, and the difficulties

encountered when more complex controllers and behaviors were sought (Gruau and Quatramaran 1997; Kodjabachian and Meyer 1998b), led us to suspect that so-called *incremental approaches* (Harvey et al. 1994; Lee et al. 1997) should necessarily be used in conjunction with indirect encoding schemes in more realistic applications. In other words, according to such a strategy, appropriate controllers and behaviors should be evolved and developed through successive stages, in which good solutions to a simpler version of a given problem are used iteratively to seed the initial population of solutions likely to solve a harder version of this problem.

As shown in Kodjabachian and Meyer (1995), indirect encoding schemes currently rely on four different paradigms for modeling development: rewriting rules (Boers and Kuiper 1992; Gruau 1994), axonal growth processes (Cangelosi et al. 1995; Vaario 1993; Vaario et al. 1997), genetic regulatory networks (Dellaert and Beer 1994; Eggenberger 1997), and nested directed graphs (Sims 1994). According to this typology, the SGOCE evolutionary paradigm that we are using at AnimatLab – which entails adding an axonal growth process to the cellular encoding scheme described by Gruau (1994) – turns out to be a combination of the first two above-mentioned approaches used to evolve developmental rules. Such rules, in turn, are used to develop a neural network module that is connected into an animat's sensors and actuators in order to control a given behavior and to confer a given competency. When needed, additional behaviors or competencies can be dealt with successively by evolving new developmental programs, which not only create new modules with new sensori-motor connections, but also generate inter-modular connections that secure an adapted and coherent overall functioning. Eventually, such an incremental methodology generates control architectures that are strongly reminiscent of Brook's subsumption architectures (Brooks, 1986).

After a description of the SGOCE paradigm, this article will successively describe how it has been used to evolve neural controllers for rolling, walking, swimming, and flying simulated animats, and how the best controllers thus obtained have occasionally been used to control actual robots. The advantages and difficulties of this approach will then be discussed and directions for future work will be proposed.

2. The SGOCE evolutionary paradigm

This paradigm is characterized by an encoding scheme that relates the animat's genotype and phenotype, by syntactic constraints that limit the complexity of the developmental programs generated, by an evolutionary algorithm that generates the developmental programs, and by an incremental strategy that helps produce neural control architectures likely to exhibit increasing adaptive capacities.

A. The encoding scheme

SGOCE is a simple geometry-oriented variation of Gruau's cellular encoding scheme (Gruau 1994, 1996) that is used to evolve simple developmental programs capable of generating neural networks of arbitrary complexity. According to this scheme, each cell in a developing network occupies a given position in a 2D metric substrate and can connect with other cells through efferent or afferent connections. In particular, such cells can be connected to sensory or motor neurons that have been positioned by the experimenter at initialization time at specific locations within the substrate. Moreover, during development, such cells may divide and produce new cells and new connections that expand the network. Ultimately, they may become fully functional neurons that participate in the behavioral control of a given animat, although they also may occasionally die and reduce the network's size.

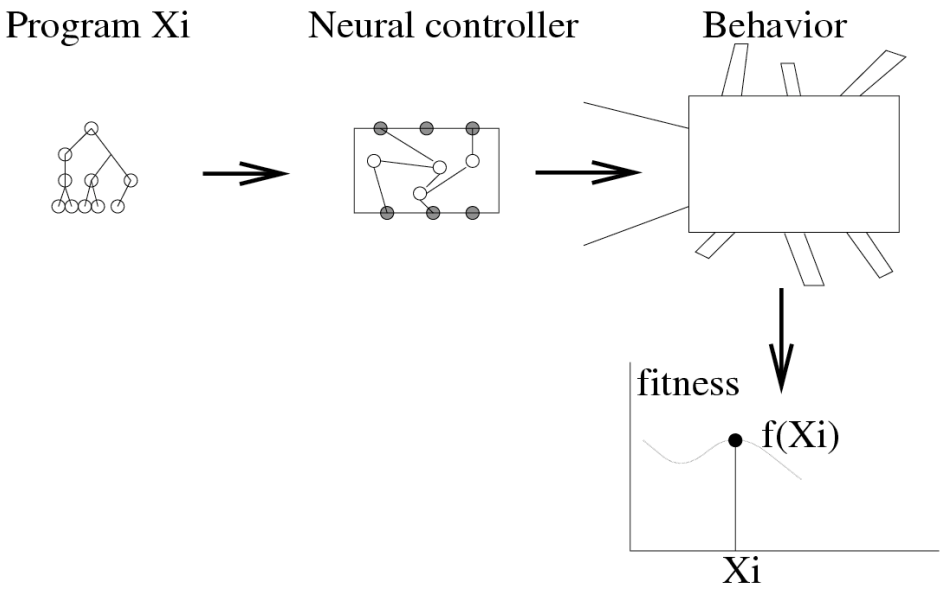


Figure 1. The three stages of the fitness evaluation procedure of a developmental program (X_i). First, the program is executed to yield an artificial neural network. Then the neural network is used to control the behavior of a simulated animat or a real robot that has to solve a given task in an environment. Finally, the fitness of Program X_i is assessed according to how well the task has been carried out.

SGOCE developmental programs call upon subprograms that have a tree-like structure like those of genetic programming (Koza 1992, 1994; Koza et al. 1999). Therefore, a population of such subprograms can evolve from generation to generation, a process during which individual instructions can be mutated within a given subprogram, and sub-trees belonging to two different subprograms can be exchanged.

At each generation, the fitness of each developmental program is assessed (Figure 1). To this end, the experimenter must provide and position within the substrate a set of precursor cells, each characterized by a local frame that will be inherited by each neuron of its lineage and according to which the geometrical specifications of the developmental instructions will be interpreted.

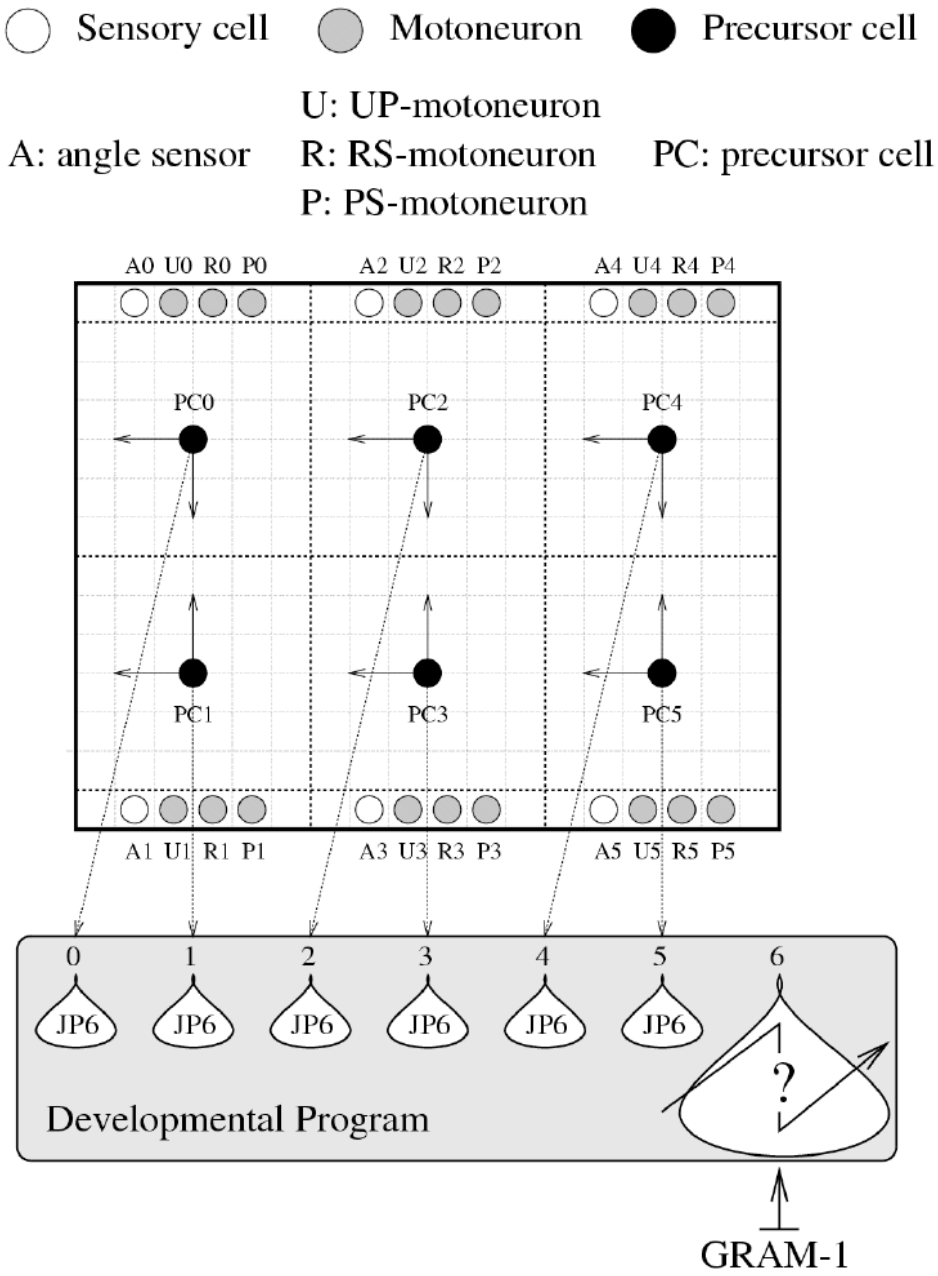


Figure 2. Setup for the evolution of a neural network that will be used in section 4 to control locomotion in a six-legged animat. The figure shows the initial positions of the sensory cells, motoneurons and precursor cells within the substrate, as well as the structure of the developmental programs 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. Its structure is constrained by the GRAM-1 tree-grammar (to be described in section 2.B). Additional details are to be found in the text.

Likewise, the experimenter must provide and position a set of sensory cells and motoneurons that will be used by the animat to interact with its environment. Lastly, an overall description of the structure of the developmental program that will generate the animat's control architecture, together with a specification of a grammar that will constrain its evolvable subprograms as described further, must be supplied (Figure 2).

Each cell within the animat's control architecture is assumed to hold a copy of this developmental program. Therefore, the program's evaluation starts with the sequential execution of its instructions by each precursor cell and by any new cell that may have been created during the course of development (Figure 3). At the end of this stage, a complete neural network is obtained whose architecture will reflect the geometry and symmetries initially imposed by the experimenter, to a degree depending on the side effects of the developmental instructions that have been executed. Through its sensory cells and its motoneurons, this neural network is then connected to the sensors and actuators of a model of the animat. This, together with the use of an appropriate fitness function, makes it possible to assess the

network's capacity to generate a specific behavior.

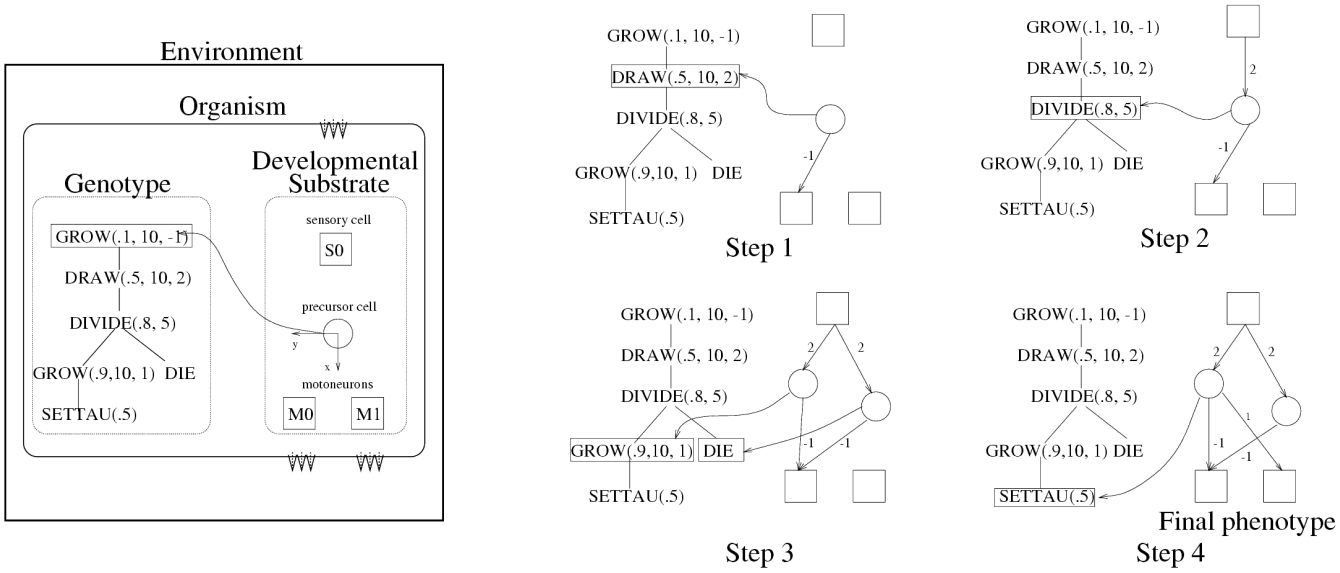


Figure 3. The developmental encoding scheme of SGOCE. The genotype that specifies the animat's nervous system is encoded as a grammar tree whose nodes are specific developmental instructions. Within such chromosomes, mutations change one branch into another, and crossovers swap branches. Each cell in the developing network reads the chromosome at a different position. The number of developmental steps required to generate a phenotype varies depending upon the length of the corresponding genotype.

Thus, from generation to generation, the reproduction of good controllers – and hence of good developmental programs – can be favored to the detriment of the reproduction of bad controllers and bad programs, according to standard genetic algorithm practice (Goldberg 1989).

DIVIDE a r create a new cell

GROW a r w create a connection to another cell

DRAW a r w create a connection from another cell

SETBIAS b modify the bias parameter

SETTAU t modify the time constant parameter

DIE trigger cellular death

Table 1. A set of developmental instructions.

In the various applications described herein, a small set of developmental instructions, like those listed in Table 1, needed to be included in evolvable subprograms.

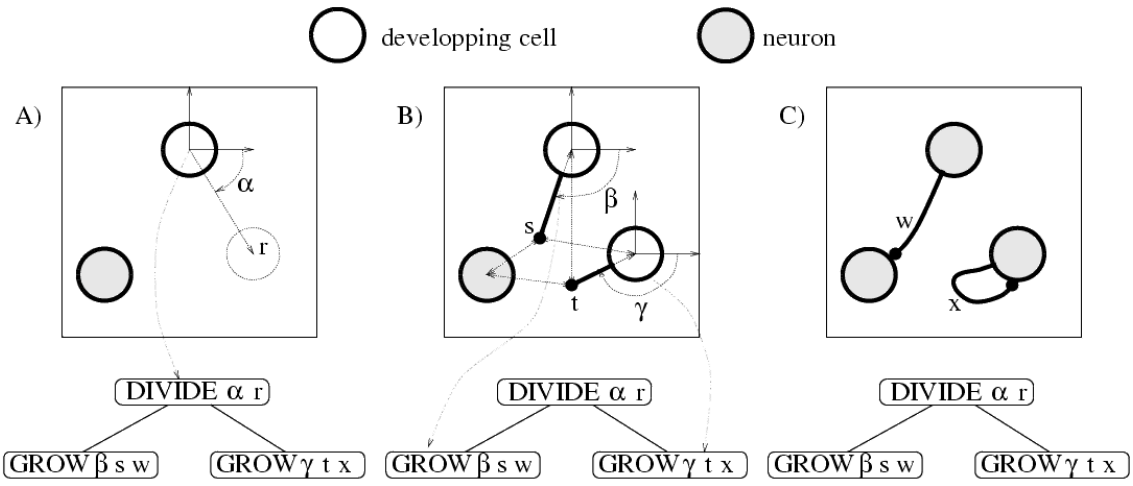


Figure 4. The effect of a sample developmental code. A) When the upper cell executes the *DIVIDE* instruction, it divides. The position of the daughter cell in the mother cell's local frame is given by the parameters a and r of the *DIVIDE* instruction, which respectively set the angle and the distance 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) Consequently, a connection is grown from each of the two cells. The two first parameters of a *GROW* instruction determine a target point in the local frame of the corresponding cell. The connection is formed 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.

A cell division instruction (*DIVIDE*) makes it possible for a given cell to generate a copy of itself. A direction parameter (a) 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 this cell's position and is oriented like the mother cell's frame (Figure 4).

Two instructions (*GROW* and *DRAW*) create respectively one new efferent and one new afferent connection. The cell that will be connected to the current cell is the one closest to a target position specified by the instruction parameters provided the target position lies within the substrate. No connection is created if the target is outside the substrate's limits. In some applications, another instruction, called *GROW2*, is also used which is similar to the *GROW* instruction except that it creates a connection from a cell in one neural module to another cell in another module. The synaptic weight of a new connection is given by the parameter w. Two additional instructions (*SETTAU* and *SETBIAS*) specify the values of a neuron's time constant t and bias b. Lastly, the *DIE* instruction causes a cell to die.

Neurons whose complexity is intermediate between abstract binary neurons and detailed compartmental models are used in the present applications. Contrary to neurons used in traditional PDP applications (McClelland and Rumelhart 1986; Rumelhart and McClelland 1986) such neurons exhibit internal dynamics. However, instead of simulating each activity spike of an actual neuron, the leaky-integrator model at work here only monitors each neuron's average firing frequency. According to this model, the mean membrane potential m_i of a neuron N_i is governed by equation 1:

$$\tau \cdot dm_i / dt = -m_i + \sum w_{i,j} x_j + I_i \quad (1)$$

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 firing threshold, and τ is a time constant associated with the passive properties of the neuron'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 between neuron N_j and neuron N_i. This model has already been used in several applications involving continuous-time recurrent neural network controllers (Beer and Gallagher 1992; Cliff et al. 1993; Gruau 1994; Cliff and Miller 1996). It has the advantage of being a universal dynamics approximator (Beer 1995), i.e., of being likely to approximate the trajectory of any smooth dynamic system.

B. Syntactic restrictions

In order to reduce the size of the genetic search-space and the complexity of the generated networks, a context-free tree grammar is often used to impose that each evolvable subprogram will have the structure of a well-formed tree. For instance, Figure 5 shows the GRAM-1 grammar used in Sections 4 and 6 below to constrain the structure of the subprograms that participate in the developmental process of neural networks that control walking and flying behaviors in animats.

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 ⟶ DIVIDE(Level1, Level1)
Level1 ⟶ DIVIDE(Level2, Level2)
Level2 ⟶ DIVIDE(Neuron, Neuron)
Neuron ⟶ SIMULT3(Bias, Tau, Connex) | DIE
Bias ⟶ SETBIAS | DEFBIAS
Tau ⟶ SETTAU | DEFTAU
Connex ⟶ SIMULT4(Link, Link, Link, Link)
Link ⟶ GROW | DRAW | NOLINK

Starting symbol

Start1.

Figure 5. The GRAM-1 grammar. This grammar defines a set of developmental programs, i.e., those that can be generated from it, starting with the Start1 symbol. When this grammar is used, a cell that executes such a program undergoes two division cycles, yielding four daughter cells, which can either die or modify internal parameters (e.g., time constant τ or bias B in equation 1) that will influence their behavior within the final neural controller. Finally, each surviving cell establishes a number of connections, either with another cell, or with the sensory cells or motoneurons that have been positioned by the experimenter in the

developmental substrate. According to this grammar, no more than three successive divisions can occur and the number of connections created by any cell is limited to four. Thus, the final number of interneurons and connections created by a program that is well formed in terms of this grammar cannot exceed 8 and 32, respectively.

The set of terminal symbols consists of developmental instructions, like those listed in Table 1, 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 parameters b and t unchanged. These 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 enables 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.

Owing to 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.

C. Evolutionary algorithm

To slow down convergence and to promote the appearance of ecological niches, the SGOCE evolutionary paradigm resorts to a steady-state genetic algorithm that involves a population of N randomly generated, well-formed programs distributed over a circle and whose operation is outlined in Figure 6.

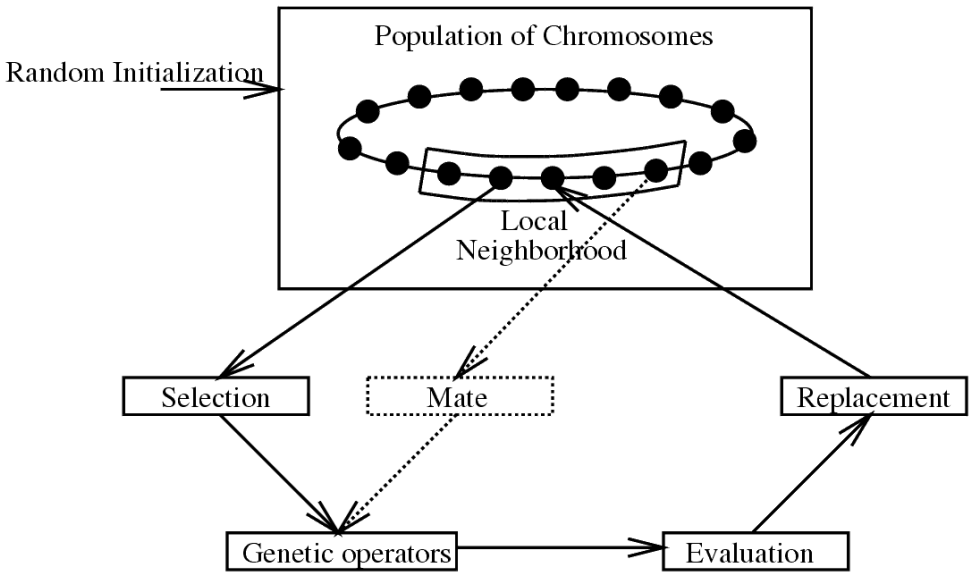


Figure 6. The evolutionary algorithm. See text for explanation.

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

1. A position P is chosen on the circle.

2. A two-tournament selection scheme is implemented whereby the better of two randomly selected programs from the neighborhood of P is kept.

3. The selected program is allowed to reproduce, and three genetic operators may modify it. The recombination operator is applied with probability p_c. It exchanges two compatible 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 probability p_m. It changes a randomly selected sub-tree into another compatible, randomly generated one. The second mutation operator is applied with probability 1. It modifies the values of a random number of parameters, implementing a constant perturbation strategy (Spencer 1994). The number of parameters to be modified is derived from a binomial distribution $B_{(n; p)}$.

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 two-tournament anti-selection scheme that selects the worse of two randomly chosen programs is used to decide which individual (in the neighborhood of P) will be replaced by the modified program.

In all the experiments reported in this article, $p_c = 0{:}6$, $p_m = 0{:}2$, $n = 6$ and $p = 0{:}5$.

D. Incremental methodology

In some applications, it has been found useful to let the SGOCE paradigm resort to an incremental approach that takes advantage of the geometrical nature of the developmental model. For instance, to control the behavior of a simulated insect as described in Kodjabachian and Meyer (1998a,b), locomotion controllers were evolved in a first evolutionary stage. At the end of that stage, the best such controller was selected to be the locomotion module used thereafter and the corresponding network was frozen, in the sense that the number of its neurons, their individual parameters, and their intra-modular connections were no longer allowed to evolve. During further evolutionary stages, other neural modules were evolved in which neurons were allowed to grow, not only intra-modular connections between themselves, but also inter-modular connections to neurons in the locomotion controller. Such modules could both influence the locomotion module and control higher-level behaviors. For instance, Figure 7 shows how the successive connection of two additional modules with a locomotion controller has been used to generate successively straight-line walking, gradient-following and obstacle-avoiding capacities in a simulated insect.

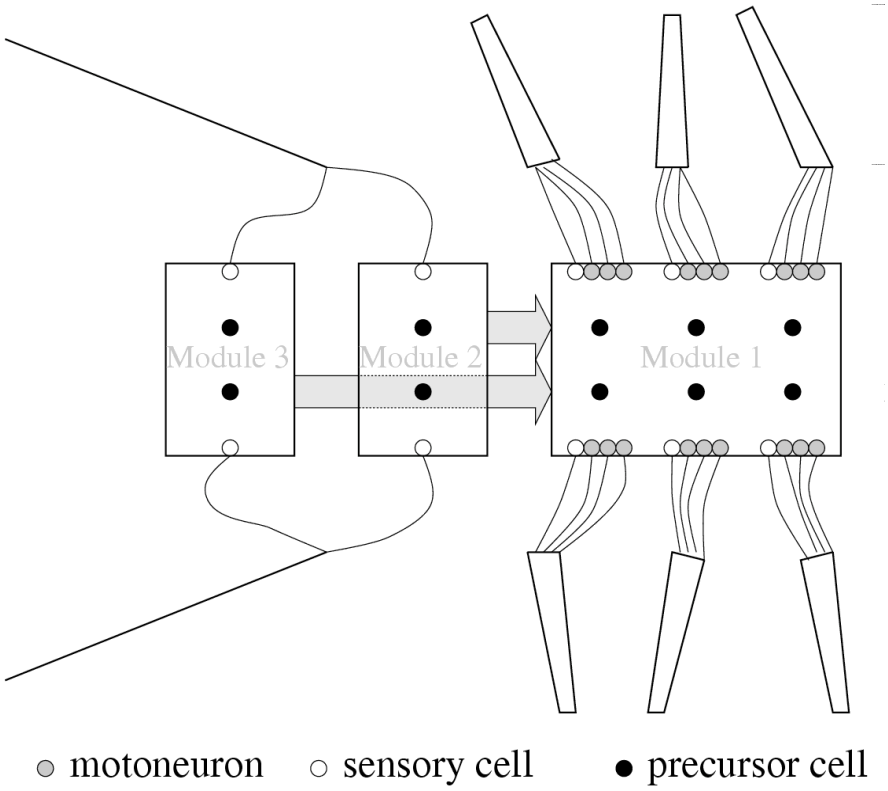


Figure 7. The SGOCE incremental approach. During a first evolutionary stage, Module 1 is evolved. This module receives proprioceptive information through sensory cells and influences actuators through motoneurons. In a second evolutionary stage, Module 2 is evolved. This module receives specific exteroceptive information through dedicated sensory cells and can influence the behavior of the animat by making connections with the neurons of the first module. Finally, in a third evolutionary stage, Module 3 is evolved. Like Module 2, it receives specific exteroceptive information and it influences Module 1 through inter-modular connections.

3. Obstacle-avoidance in a rolling animat

3.A. The problem

To evolve neural controllers for robust obstacle-avoidance in a Khepera robot (Chavas et al. 1998), a two stage approach was implemented in which controllers for obstacle-avoidance in medium-light conditions were first evolved, then improved to operate also under more challenging strong-light conditions, when a lighted lamp was added to the environment. Comparisons with results obtained under the alternative one-shot strategy did support the above-mentioned intuition about the usefulness of an incremental approach.

Khepera (Mondada et al. 1993) is a round miniature mobile robot – 55 mm across, 30 mm high, and weighing of 70 g – that rides on two wheels and two small Teflon balls. In its basic configuration, it is equipped with eight proximity sensors – six on the front, two on the back – that may also act as visible-light detectors. The wheels are controlled by two DC motors with incremental encoder that moves in both directions.

An infra-red light emitter and receiver are embedded in each proximity sensor of Khepera. This hardware allows two measurements to be made: that of normal ambient light – through receivers only – and that of light reflected by the obstacles – using both emitters and receivers. In medium-light conditions, this hardware makes it possible to detect a nearby obstacle – limited to about 6 cm – by successively sensing the infra-red light reflected by the obstacle and the infra-red contribution of the

ambient light, then by subtracting the corresponding values. However, under strong light conditions, the corresponding receptors tend to saturate and the subtraction yields meaningless values (Figure 8).

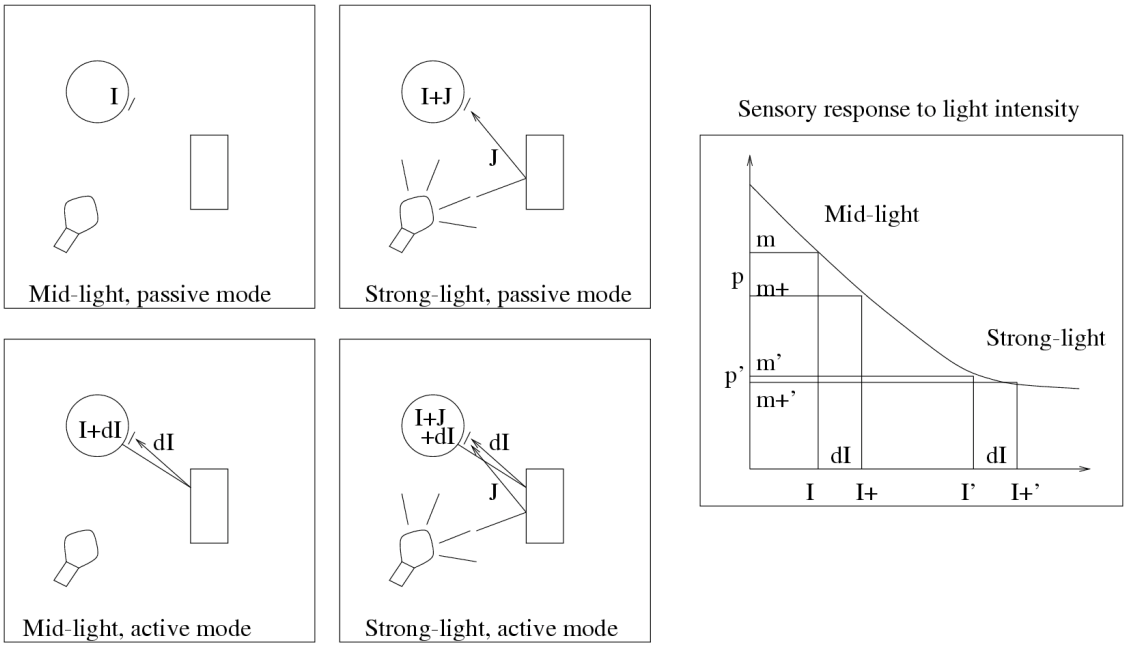


Figure 8. Left: This portion of the figure shows the different sources that contribute to the intensity received by a sensor. The geometrical configuration considered is the same in all four situations. The ambient light contributes an intensity I. When the lamp is lighted, the sensor receives an additional contribution J through direct and/or reflected rays. Finally, in active mode, the reflected IR-ray yields an intensity dI at the level of the sensor. The value of dI depends solely on the geometrical configuration considered. Intensities are summed, but the sensor response is non-linear. Right: In strong-light conditions, the response of a sensor can saturate. In this case, where $I' = I + J$, the same increase in intensity dI caused by the reflection of an IR-ray emitted by the robot causes a smaller decrease in the value measured than it does in the linear region of the response curve (medium-light conditions). In other words, although dI intensity increases can be sensed under medium-light conditions – and the corresponding obstacle configurations can thus be detected – this no longer holds true under strong-light conditions.

Therefore, an initial research effort aimed at automatically evolving a robust obstacle-avoidance controller capable of differentiating under both light conditions and of taking appropriate motor decisions. Such a controller was generated through simulations performed according to the SGOCE paradigm. Subsequently the corresponding network was downloaded on a Khepera robot and its ability to generate the requested behavior was ascertained.

3.B. Experimental approach.

The artificial evolution of robust controllers for obstacle-avoidance was carried out using a Khepera simulator to solve successively two problems of increasing difficulty. Basically, this entailed evolving a first neural controller that used its sensors in active mode to measure the proximity value $p = K (m - m^+)$ (see Figure 8) , in order to avoid obstacles successfully in medium-light conditions. Then a second neural controller was evolved that operated in passive mode under strong-light conditions and used measurements of the ambient light level m to modulate the normal function of the first controller. In other words, such an incremental approach relied upon the hypothesis that the second controller would be able to evaluate the local intensity of ambient light so as to change nothing in the correct obstacle-avoidance behavior secured by the first controller in medium-lighted regions, but to alter it – in whatever appropriate manner evolution would discover – when the robot traveled through strongly lighted regions liable to impair the proper operation of the first controller.

During Stage 1, to evolve the first controller and generate a classical obstacle-avoidance behavior under medium-light conditions, the following fitness function was used:

$$f_1 = \sum_t \left(0.5 + \frac{v_l(t) + v_r(t)}{4 \cdot V_{max}} \right) \cdot \left(1 - \frac{|v_l(t) - v_r(t)|}{2 \cdot V_{max}} \right) \cdot \left(1 - \frac{\sum_{front} p_i(t)}{4 \cdot P_{max}} \right) \quad (2)$$

where $v_l(t)$ and $v_r(t)$ were the velocities of the left and right wheels, respectively; V_{max} was the maximum absolute velocity; $p_i(t)$ was the proximity measurement returned by each sensor i of the four front sensors; P_{max} was the largest measured value that can be returned.

In the right-hand part of this equation, the first factor rewarded fast controllers, the second factor encouraged straight locomotion, and the third factor punished the robot each time it sensed an obstacle in front of it.

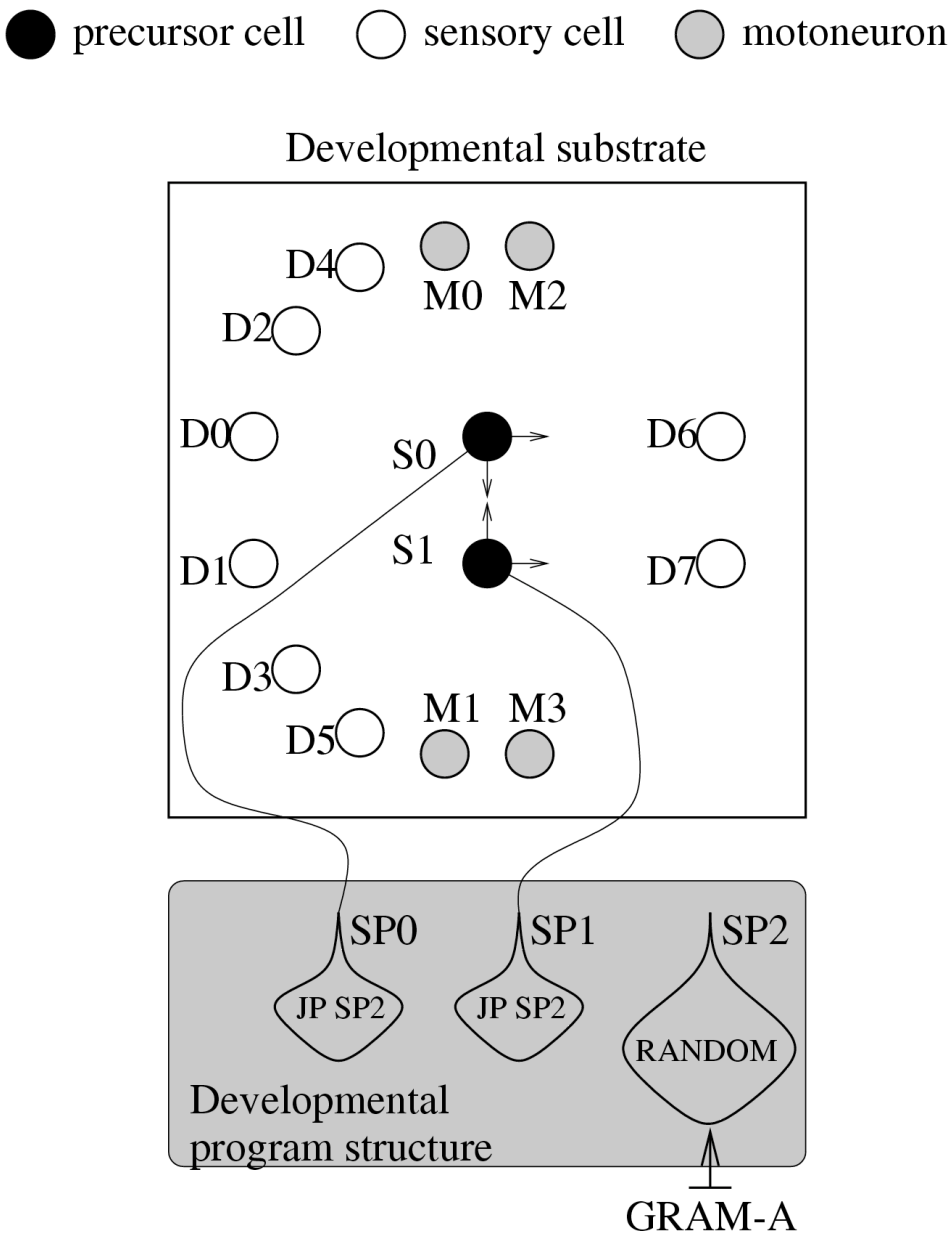


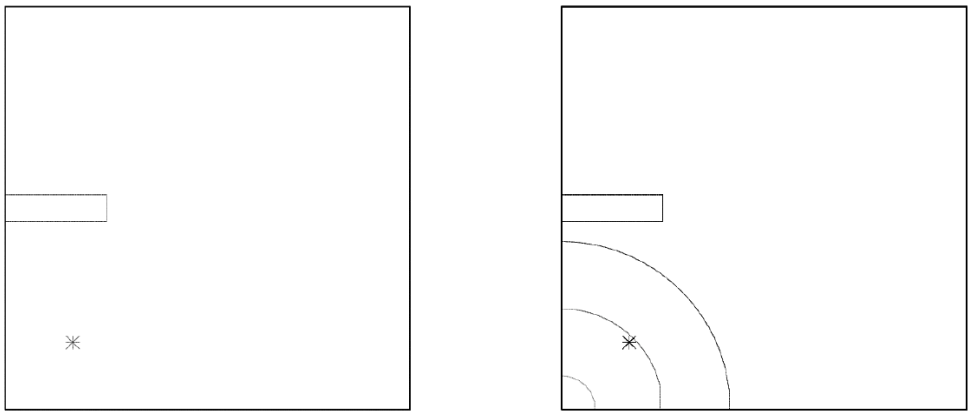
Figure 9. The substrate and the developmental program that were used to evolve obstacle-avoidance in a Khepera robot. Each of the two precursor cells carries out a developmental program that starts by a jump instruction to the evolving sub-program SP2. During evolution, the composition of sub-program SP2 can be modified within the limits defined by the constraints encoded in grammar GRAM-A described on Figure 10. D0 to D7 are sensory cells and M0 to M3 are motoneurons, which have been placed by the experimenter in specific positions within the substrate.

Using the substrate of Figure 9 and the grammar of Figure 10, controllers likely to include eight different sensory cells and four motoneurons were evolved after a random initialization of the population. The fitness of these controllers was assessed by letting them control the simulated robot over 500 time-steps in a square environment containing an obstacle (Figure 11-a).

Terminal symbols
DIVIDE, GROW, DRAW, SETBIAS, SETTAU, DIE,
NOLINK, DEFBIAS, DEFTAU, SIMULT3, SIMULT4.
Variables
Start1, Level1, Neuron, Bias, Tau, Connex, Link.
Production rules
Start1 $\longrightarrow$ DIVIDE(Level1, Level1)
Level1 $\longrightarrow$ 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.

Figure 10. The GRAM-A grammar, according to which a cell undergoes two division cycles yielding four daughter cells, which can either die or modify internal parameters (time-constant and bias) that will influence their future behavior as neurons. Each surviving cell establishes a maximum number of four connections, either with another cell or with sensory cells and motoneurons.

For this purpose, the sensory cells D0 to D7 in Figure 9 were connected to the robot's sensors such that the instantaneous activation value each cell propagated through the neural network to the motoneurons was set to the proximity measure p returned by the corresponding sensor. Likewise, the motoneurons were connected to the robot's actuators such that a pair of motoneurons was associated with each wheel, the difference between their inputs determining the speed and direction of rotation of the corresponding wheel.



(a) Medium-light environment. (b) Strong-light environment.

Figure 11. The environment that was used to evaluate the fitness of each controller. (a) Medium-light conditions of Stage 1. The star indicates the starting position of each run. (b) Strong-light conditions of Stage 2. A lighted lamp is positioned in the lower-left corner of the environment. Concentric arcs illustrate the corresponding intensity gradient.

Each controller was evaluated five times in the environment of Figure 11a, starting in the same position, but with five different orientations, its final fitness being the mean of these five evaluations.

After 10000 reproduction events, the controller with the highest fitness was used to seed an initial population that was subjected to a second evolutionary stage involving strong-light conditions. During Stage 2, the corresponding fitness function became:

$$f_2 = \sum_t \left(0.5 + \frac{v_l(t) + v_r(t)}{4 \cdot V_{max}} \right) \cdot \left(1 - \frac{|v_l(t) - v_r(t)|}{2 \cdot V_{max}} \right) \quad (3)$$

In this equation, the third term that was included in the right-hand part of equation 2 was eliminated because it referred to active-mode sensory inputs that could not be trusted in strong-light conditions. However, fast motion and straight locomotion were still encouraged.

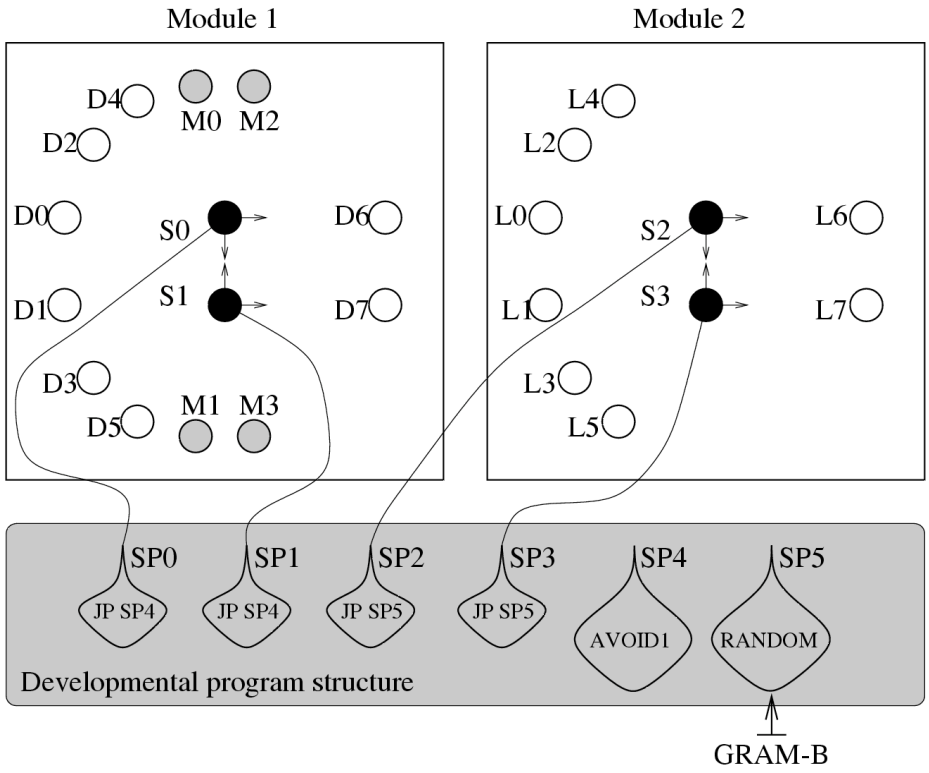


Figure 12. The initial configuration of the developmental substrate and the program structure used in Stage 2 for the evolution of obstacle-avoidance in Khepera. The same substrate was used for control experiments, in which both modules were evolved simultaneously. In these latter experiments, both SP4 and SP5 were initialized randomly and were submitted to evolution under constraints given by GRAM-A and GRAM-B, respectively.

Using the substrate of Figure 12 and the grammar of Figure 13, controllers likely to include 16 different sensors and four motoneurons were evolved during 10000 additional reproduction events.

Terminal symbols
DIVIDE, GROW, DRAW, GROW2, SETBIAS, SETTAU, DIE, NOLINK, DEFBIAS, DEFTAU, SIMULT3, SIMULT4.
Variables
Start1, Level1, Neuron, Bias, Tau, Connex, Link.
Production rules
Start1 $\longrightarrow$ DIVIDE(Level1, Level1)
Level1 $\longrightarrow$ 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 | GROW2 | NOLINK
Starting symbol
Start1.

Figure 13. The GRAM-B grammar. It is identical to GRAM-A except for the addition of one instruction (GROW2) that makes it possible to create a connection from the second to the first module.

This time, the activation values that the new sensory cells L0 to L7 in Figure 12 propagated through a given controller were each set to the ambient light measurement m returned by the robot's corresponding sensor. The fitness of the corresponding controller was assessed in the same square environment as the one used in Stage 1, but with a lighted lamp positioned in one of its corners (Figure 11b).

Again, the final fitness was the mean of five evaluations that corresponded to five trials of 500 time-steps, each starting in the same position, but with different orientations and light intensities. In particular, one such trial was performed in medium-light conditions when the lamp was switched off, two others were performed when the lamp was contributing a small amount of additional light, and the last two were performed in strong-light conditions, where the lamp contributed its maximum light intensity. At the end of Stage 2, the best neural network thus obtained was downloaded and tested on a Khepera for 50 seconds.

3.C. Experimental results

As explained in the original publication (Chavas et al. 1998), the comparison of strong-light fitnesses indicated that controllers selected at the end of Stage 1 were less efficient than those that were obtained at the end of Stage 2 when the lighted lamp was added to the environment. Thus, this second evolutionary stage contributed to improving the behavior of the robot under strong-light conditions. Likewise, the comparison of medium-light fitnesses indicated that controllers selected at the end of Stage 2, according to their capacities at coping with strong-light conditions, didn't lose the essential of their abilities to generate appropriate behavior under medium-light conditions. Indeed, although their fitnesses tended to be slightly lower than those of the best controllers of Stage 1, they still were of the same order of magnitude.

To assess the usefulness of the incremental approach that was used here, additional control runs were performed, each involving 10000 reproduction events and the same number of evaluations (100000) as were made in the incremental runs. Each such run directly started with the substrate of Figure 12 and called upon both GRAM-A and GRAM-B grammars, thus permitting the simultaneous evolution of both neural modules. The fitness of each individual was the mean of 10 evaluations: five in the conditions of

Stage 1 described above, and five in the conditions of Stage 2. A quantitative comparison of incremental and control runs indicates that the means of the medium-light and strong-light fitnesses obtained at the end of the incremental runs are statistically higher than the corresponding means obtained at the end of the control runs. Moreover, a qualitative comparison of the behaviors generated by these controllers indicates that the behaviors of the control runs are far less satisfactory than those of their incremental competitors. In fact, in almost every control run, the robot alternated between moving forward and backward and never turned.

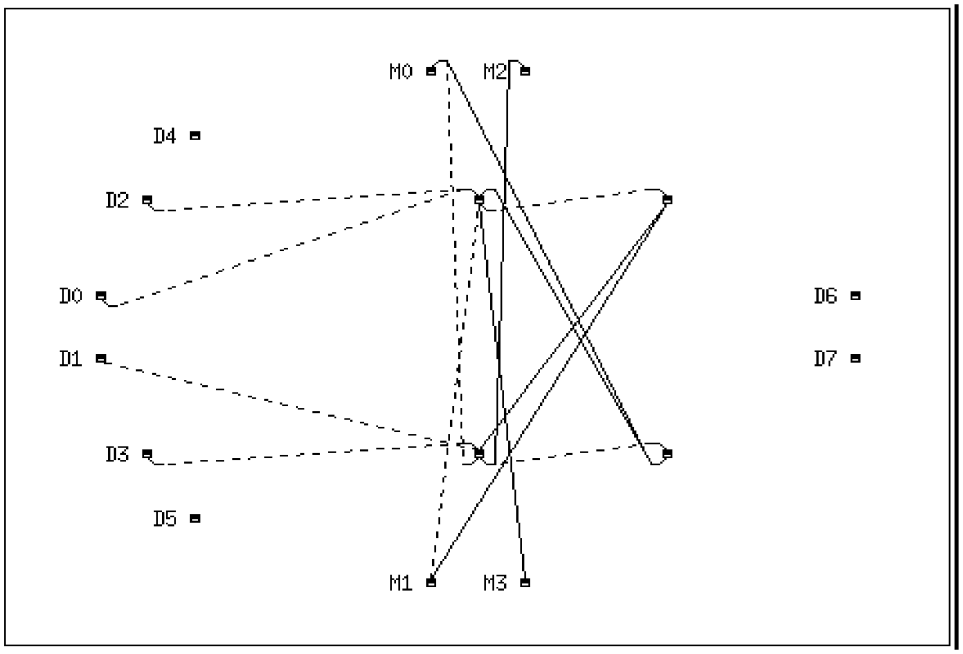


Figure 14. The best controller obtained after Stage 1 for the particular run that resulted in the controller of Figure 15. The outputs of the motoneurons M2 and M3 are interpreted as forward motion commands for the right and left wheels, respectively, while the output of the motoneurons M0 and M1 correspond to backward motion commands. Solid lines correspond to excitatory connections, while dotted lines indicate inhibitory links. This network contains four interneurons.

To understand how the controllers obtained during the incremental runs succeeded in generating satisfactory behaviors, the internal organization of the neural networks obtained at the end of Stage 1 (Figure 14) and Stage 2 (Figure 15) in one of these runs has been examined. The corresponding simulated behaviors are shown in Figures 16A-D respectively.

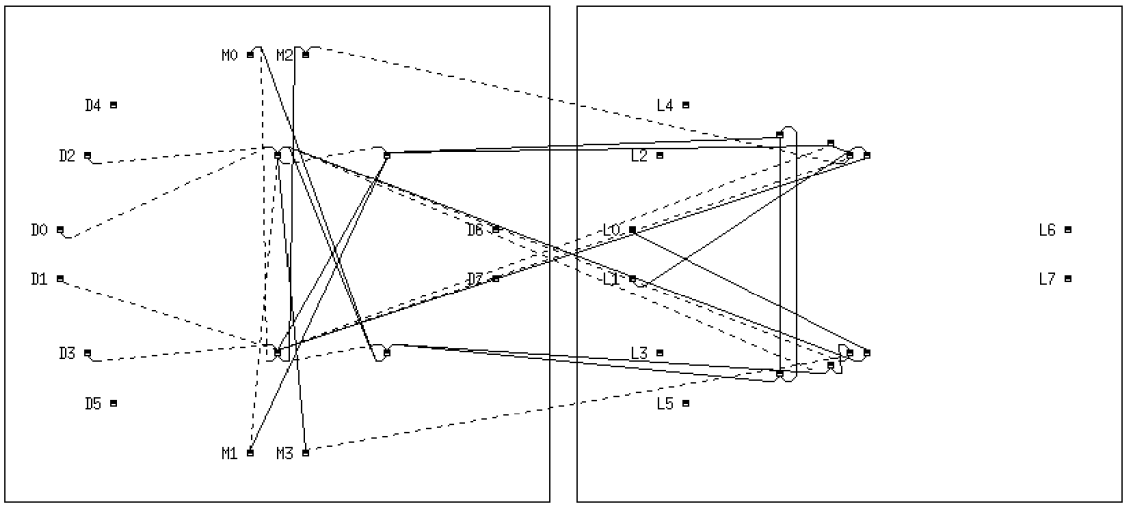
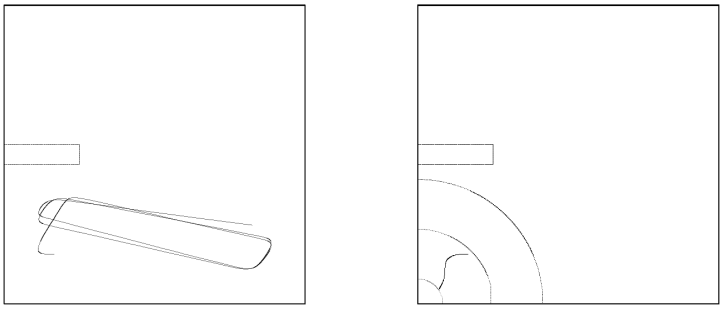
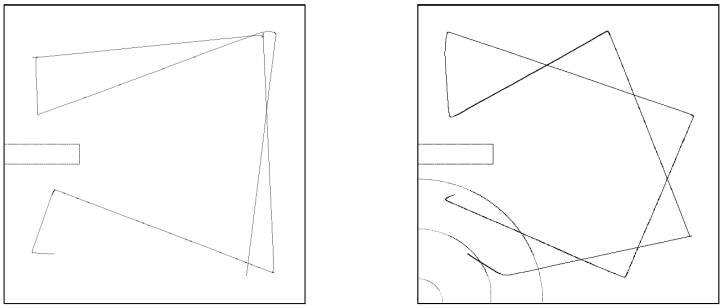


Figure 15. The best controller obtained after Stage 2. This networks contains four interneurons in Module 1 and eight interneurons in Module 2.

It thus appears that the single-module controller uses the four front sensors only. It drives the robot at maximal speed in open areas, and makes it possible to avoid obstacles in two different ways. If the obstacle is detected on one given side, then the opposite wheel slows down, allowing for direct avoidance. When the detection is as strong on both sides, then the robot slows down, reverses its direction, and turns slightly while backing up. After a short period of time, forward locomotion resumes. However, when placed in strong-light conditions, this controller is unable to detect obstacles and thus keeps bumping into walls. The two-module controller corrects this defect by avoiding strongly-lighted regions in the following way. In medium-light conditions, all L-sensors return high m values. Excitatory links connect each of the two frontal L-sensors, L0 and L1, to one interneuron of module 1 apiece. Each of these interneurons in turn sends a forward motion command to the motor on the opposite side. Consequently, whenever the value m returned by one of these two sensors decreases – an event that corresponds to the detection of a high light intensity – the corresponding interneuron in Module 1 becomes less activated and the wheel on the opposite side slows down. This results in a light avoidance behavior.



(A) Best of Stage 1, medium-light (sim.) (B) Best of Stage 1, strong-light (sim.)

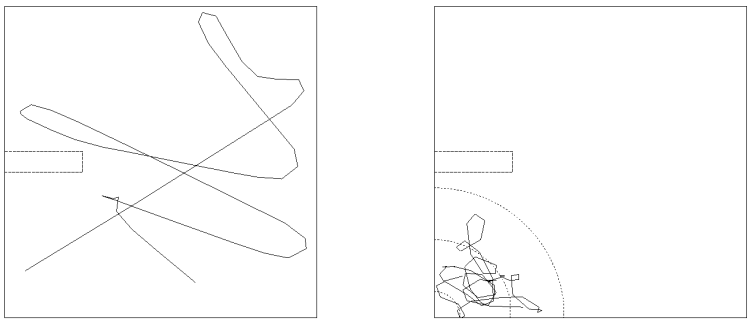


(C) Best of Stage 2, medium-light (sim.) (D) Best of Stage 2, strong-light (sim.)

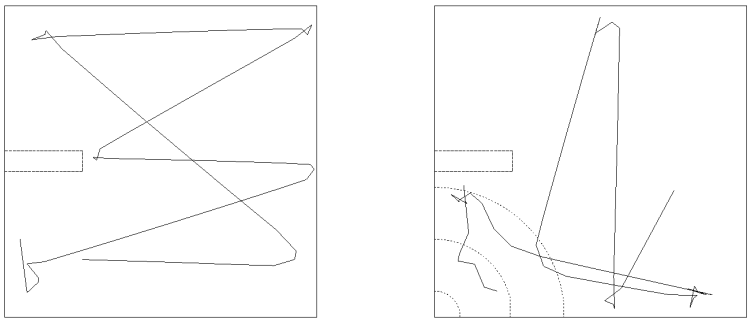
Figure 16. (Top) The simulated behavior of the single-module controller of Figure 14. When starting in front of the lamp, the corresponding robot gets stuck in the corner with the lamp. (Bottom) Simulated behavior of the two-module controller of Figure 15. Now, the robot avoids the lighted region of the environment.

Figures 17A-D show the behavior exhibited by Khepera when the networks described above are downloaded onto the robot and are allowed to control it for 50 seconds in a 60x60 cm square arena designed to scale the simulated environment. These figures were obtained through the on-line record of the successive orders sent to the robot's motors and through the off-line reconstruction of the corresponding trajectories. They demonstrate that the behavior actually exhibited by Khepera is qualitatively similar to the behavior obtained through simulation – in terms of the robot's ability to avoid obstacles and to move quickly along straight trajectories – and that it fulfills the experimenter's specifications. The main behavioral difference occurs when, at the end of the medium-light stage, controllers are tested in the presence of the additional lamp: the actual behavior of Khepera is more disrupted than the simulated behavior, probably because light that is reflected by the ground in the experimental arena is not adequately taken into account by the simulator (Figures 16B and 17B). Such

discrepancies do not occur at the end of the strong-light stage because the robot then avoids the region where the lamp is situated, and where such disturbing light reflections are the strongest.



(A) Best of Stage 1, medium-light (real) (B) Best of Stage 1, strong-light (real)



(C) Best of Stage 2, medium-light (real) (D) Best of Stage 2, strong-light (real)

Figure 17. The paths actually traveled by the Khepera robot, which have been reconstructed off-line using motor orders recorded during the trial. Top: Actual behavior of the single-module controller of Figure 14. Due to reflections present in the real world, but not in the simulation, the behavior under strong-light conditions is different from that of Figure 16. Bottom: Actual behavior of the two-module controller of Figure 15. Now the actual behavior is qualitatively similar to the simulated behavior shown in Figure 16.

4. Obstacle-avoidance and gradient-following in a walking animat

4.A. The problem

The SGOCE paradigm was used to generate neural controllers for locomotion and obstacle-avoidance in a real 6-legged robot (SECT robot manufactured by Applied AI Systems) in two stages. In a first stage, a recurrent neural network controlling straight locomotion in the SECT robot was generated. In a second stage, an additional controller able to modulate the leg movements secured by the first controller was generated, which made it possible for the robot to turn in the presence of an obstacle and to avoid it.

In this application, the main difficulty was to devise a legged robot simulation that would handle instances where some leg slippage occurs. However, as Jakobi (1997, 1998) points out, because such events are only involved in poorly efficient behaviors, they are not expected to occur with a fast-walking

gait. As a consequence, controllers producing leg slippage would never be used by an efficient real robot, and therefore leg slippage did not need to be accurately simulated, thereby tremendously simplifying the simulation problem.

In view of these considerations, our simulation assumed that all the legs were characterized by the same constant slippage factor, and simply calculated the movement of the robot body that minimized the slippage of any leg touching the floor between two time steps. Consequently, if the actual movement did not involve any slippage, the calculated movement was accurate and, conversely, if the actual movement did involve slippage, the calculated movement was a good approximation of the real one.

- Set initial robot leg and body positions.

- Iterate N times :

    1. Update sensor values using robot position.
    2. Update neural network outputs M times using sensor values.
    3. Update leg positions relative to robot body using network outputs.
    4. Find a set S of 3 legs Li, Lj, Lk, the extremities xi, xj, xk of which form a support triangle.
    5. Add into S all legs lying within a distance d of the plane P generated by xi, xj and xk.
    6. Find the elementary translation and rotation of the body that minimize the displacement of the extremities of the legs in S.
    7. Update body position, applying the optimal rotation and translation.

Figure 18. The algorithm used for the simulation of the 6-legged robot.

4.B. Experimental approach

The algorithm used for the simulation is given in Figure 18. The simulation of the sensors of the robot (step 1 in Figure 18) was straightforward, as only two binary obstacle detectors placed on either side of the head were used. It simply entailed calculating the distance from each robot sensor to the closest obstacles it could detect and, if the distance fell beneath a certain threshold, setting the activity of the corresponding sensory neuron to 1. Otherwise, the activity was set to 0.

At every iteration of the algorithm, the neural network was updated M times (M=1 in the present work) using the updated sensory activities to obtain new motoneuron outputs (step 2 in Figure 18). The position of each leg relative to the robot body (step 3 in Figure 18) was obtained through a simple geometric computation because leg positions were determined by neural network outputs only.

The determination of the legs in contact with the ground involved first finding three legs whose extremities formed a support triangle (step 4 in Figure 18). This triangle had to be such that all other leg extremities were above its plane, and that the vertical projection of the robot's center of mass fell within it. Then all leg extremities ending close enough to the plane of the triangle were also considered as being in contact with the ground (step 5 in Figure 18). The aim of this step was to smooth the effect of differences in leg lengths and positions in real and simulated robots. This resulted in having more legs considered to be in contact with the ground in the simulation than in the real case, which implied that the fitnesses of the controllers were not overestimated because controllers that did not lift the legs above a given threshold were penalized in the simulation.

The computation of the movements of the robot body was based on the assumption that the actual slippage of the legs on the ground minimized the energy dissipated by friction. In the simulation, it was assumed that the slippage factor was constant and identical for all legs. The energy dissipated by friction was consequently proportional to the sum of the distances covered by all leg extremities in contact with the ground. Hence, to determine the body movement between two time steps, the elementary horizontal rotation and translation of the robot that minimized the displacement of the legs on the ground were sought (step 6 in Figure 18). To achieve this, the distance covered by the legs in contact with the ground was computed as a function of three parameters: the two coordinates of the body translation and the angle of the body rotation. Minimizing this function, with the assumption that the body rotation remained small, led to a simple linear system with three equations and three unknowns, which was computationally easy to solve.

The 2D substrate that was used to evolve locomotion controllers is shown in Figure 19. It contained 12 motoneurons that were connected to the 12 motors of the robot, in the sense that the activity level of a given motoneuron determined the target angular position that was sent to the corresponding servo-motor. The six precursor cells executed the same evolved developmental program in order to impose symmetrical constraints on the growing neural network. Developmental programs were generated according to the GRAM-1 grammar shown in Figure 5.

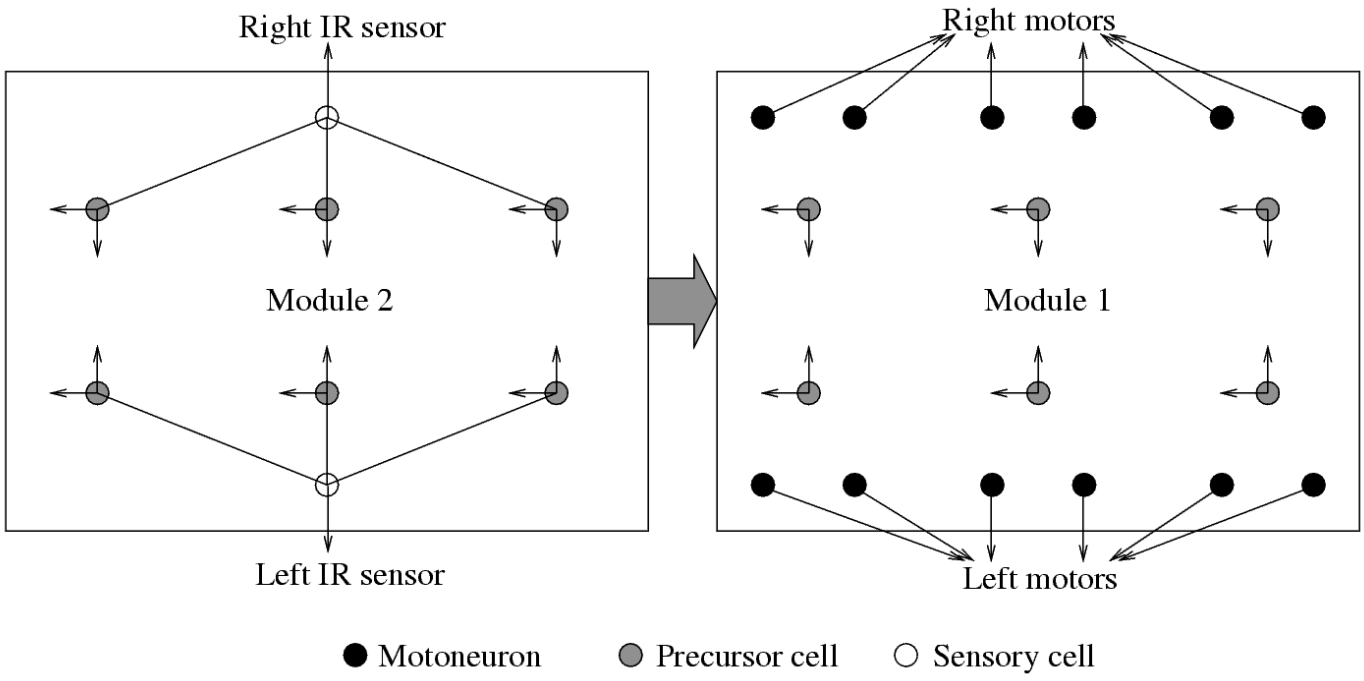


Figure 19. The substrates and the modular approach that have been used to evolve neural controllers for a walking robot. Module 1 produces straight walking, while Module 2 modifies the behavior of Module 1 in order to avoid obstacles.

The fitness of each developmental program was estimated by the mean distance covered in three obstacle-free environments during a fixed amount of time, increased by a slight bonus encouraging any

leg motion (Kodjabachian and Meyer 1998a, 1998b). It should also be noted that the simulation implicitly rewarded controllers that raised legs above a small threshold as mentioned above. Finally, the population size was 100 individuals evolving over 500 generations.

4.C. Experimental results

As indicated in Filliat et al. (1999a,b), the evolved controllers thus obtained were as efficient in reality as they were in simulation. They could be classified into two categories: those generating tripod gaits and those generating symmetrical gaits (i.e., moving the corresponding legs on both sides of the robot simultaneously).

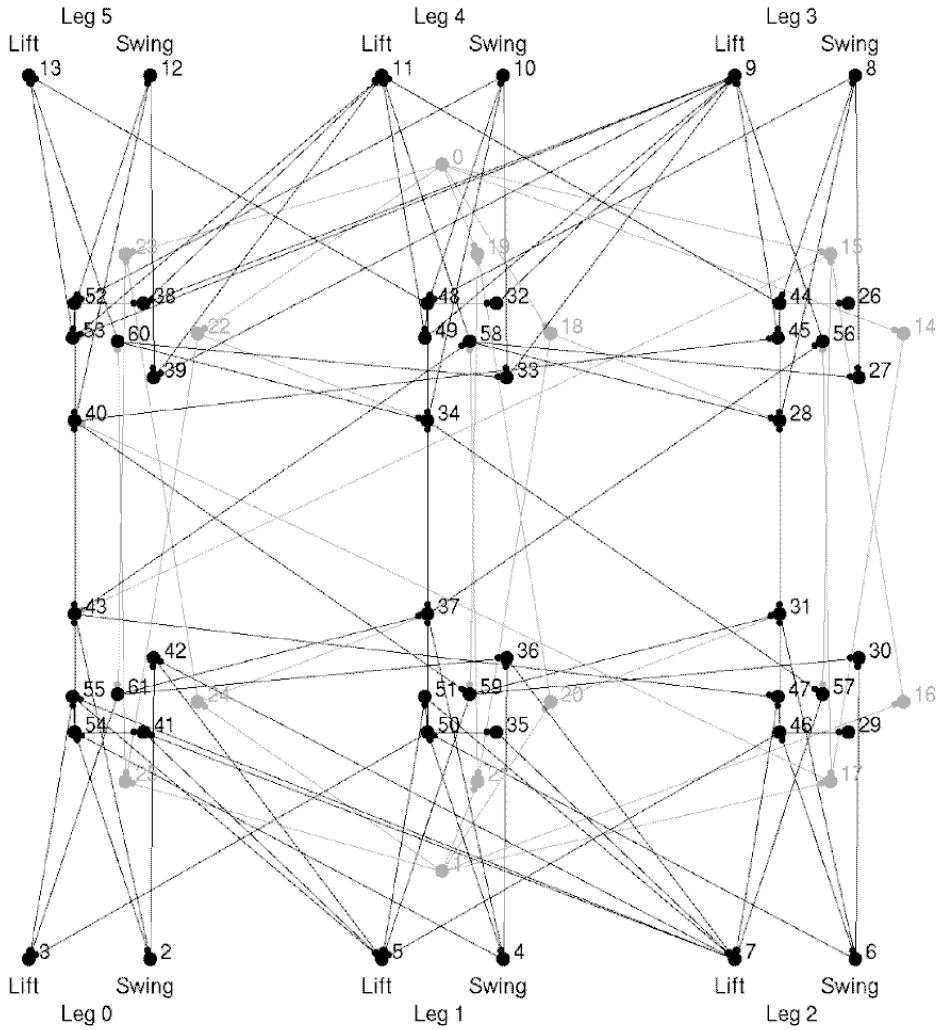


Figure 20. An example of an evolved controller for a six-legged walking robot. Black neurons belong to the module controlling locomotion, gray neurons belong to the module controlling obstacle-avoidance. Neurons 0 and 1 are input neurons connected to the IR sensors, neurons 2 to 13 are output neurons connected to the robot motors. The first controller contains 48 neurons and 82 connections; the second contains 14 neurons and 36 connections.

Figure 20 shows the best tripod-gait controller that was obtained. We analyzed its behavior in order to understand how this gait was generated. The mechanism responsible for leg lift is straightforward. It calls upon six central pattern generators, one for each leg, made up of only three neurons (Figure 21). These pattern generators are synchronized by two connections between the legs of the same side of the body, and by two connections linking two symmetrical legs on each side of the body – thus closely approximating the biologically plausible model of Cruse et al. (1995). The mechanism producing leg swing is far more intricate because it is extremely distributed. In fact, the activation of a given leg's swing neuron depends on neurons involved in the control of all the other legs, and it cannot be broken

down into six similar units as is the case of the lift neurons.

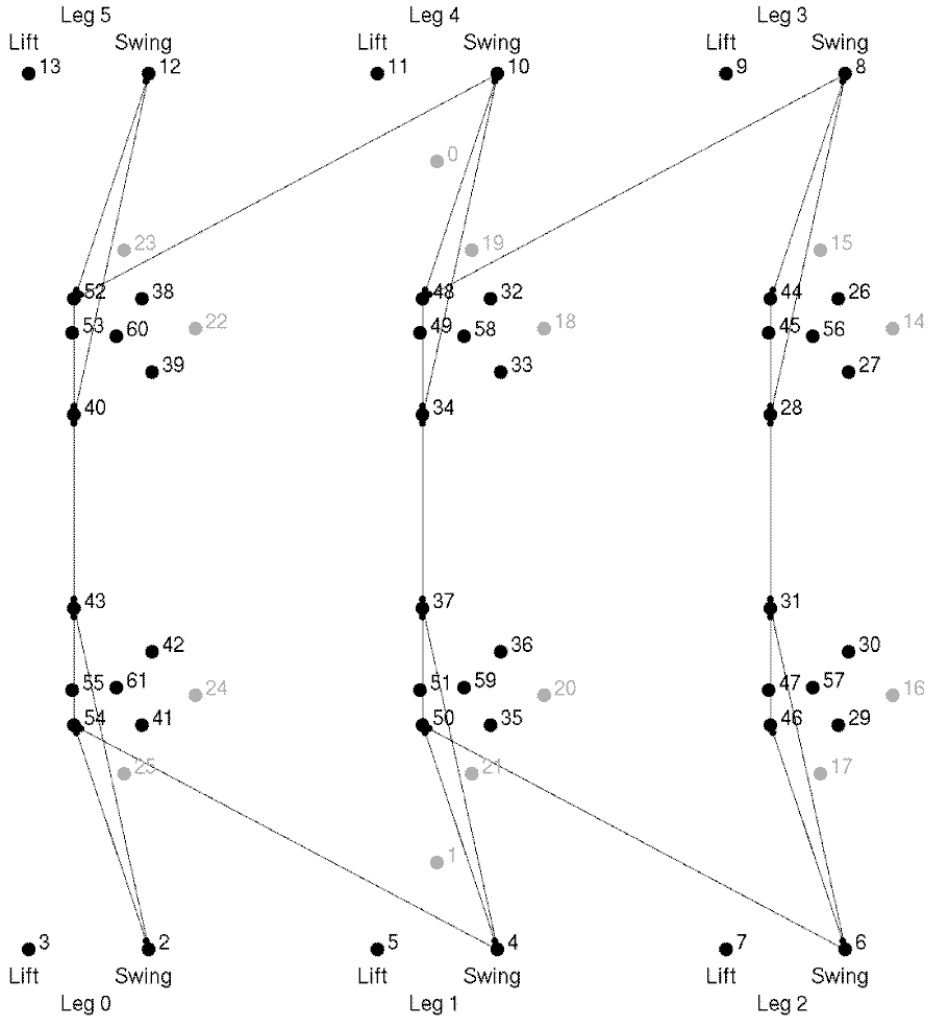


Figure 21. The central pattern generator that controls the swing motors of the robot. A group of three neurons (2, 43, 54 and their five symmetrical counterparts) participates in six identical and independent oscillators whose phase differences are controlled by the connections which link them (54-4, 40-43 and their symmetrical counterparts). The obstacle-avoidance module is connected to this sub-network alone. Its inner working is simple: whenever an IR neuron is activated, the connections between neurons 18-28,20-31,22-34,24-37, 17-40 and 15-43 (not shown) stop the oscillators on the opposite side of the detected obstacle, hence blocking the leg swing movement and making the robot turn to avoid the obstacle.

Obstacle-avoidance was evolved using a second module whose neurons could be connected to those of the tripod-gait controller of Figure 21. The substrate of this second module is shown on Figure 19. The grammar was identical to that of Figure 5, with two exceptions. The first one concerns the replacement of rule *Start1 → DIVIDE(Level1, Level1)* by *Start1 → DRAWI & DIVIDE(Level1, Level1)*, where *DRAWI* was a terminal symbol creating a connection from an input cell, and where character & meant that the two symbols it connected should be executed in sequence. This rule enforced connections between input neurons and precursor cells. The second exception concerned the addition of the instruction *GROW2* to the rule *Link → GROW|DRAW| NOLINK*, as was done in GRAM-B grammar of Figure 13. Again, this instruction allowed the precursor cell to grow an outgoing connection towards neurons in the first module. Each IR sensor was binary, i.e., it returned 0 when it detected no obstacle, and it returned 1 when an obstacle was detected within a 30 cm-range. The robot evolved in three different closed environments containing some obstacles, and the corresponding fitness was the distance covered by the robot until it touched an obstacle, or until the simulation time was over, averaged over the three environments. The population consisted of 100 individuals, and evolution lasted 200 generations.

The robot's simulated and actual behaviors were simple and very similar: legs on the side opposite a detected obstacle were blocked, thus causing the robot to change its direction and to avoid the obstacle. Analyzing the inner working of the corresponding controller (Figure 21), it turned out that such behavior was possible due to a strong inhibitory connection between a given sensor and the swing neurons of the legs on the opposite side of the robot.

Figure 22 shows the robot's trajectories in three simulated environments and in reality.

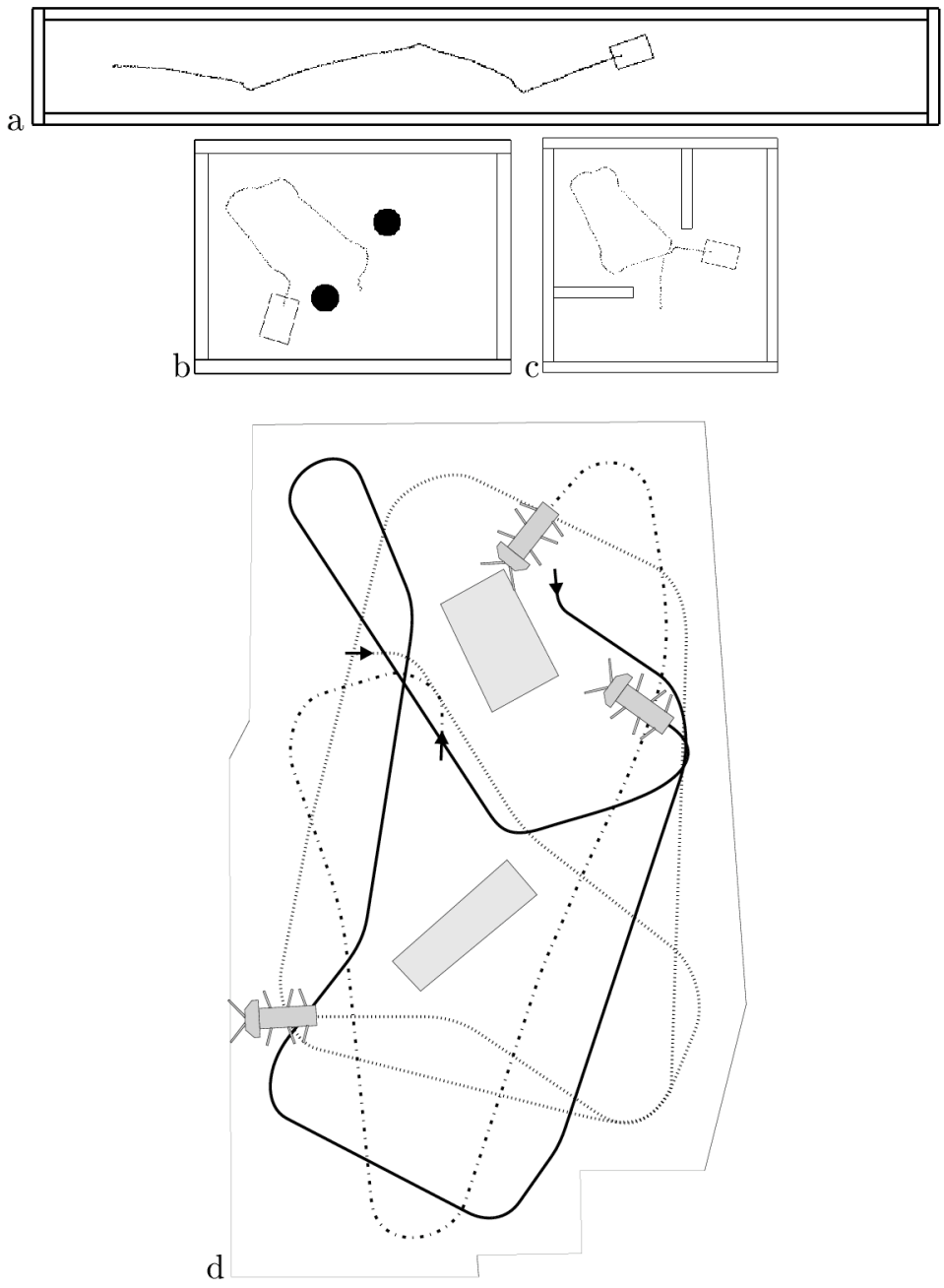


Figure 22. Robot behavior in simulated (a,b,c) and real (d) environments. The three trajectories shown in d have been reconstructed from video recordings of actual experiments. They illustrate the fact that the robot actually avoids obstacles most of the time, but that it may occasionally fail because of the dead-angle between its front IR sensors.

5. A swimming animat

5.A. The problem

In this application (Ijspeert and Kodjabachian 1999), neural networks were directly evolved – i.e., in a one-stage approach – for swimming control in a simulated lamprey. A lamprey is a fish which projects itself in water without the use of fins, through undulations of its body that call upon traveling waves propagating from head to tail. The objective was to assess the possibilities of applying two input signals to a neural network thus obtained in order to initiate swimming and to modulate the speed and the direction of motion by simply varying the amplitude of these signals. Hopefully, such controllers could easily be used by higher control centers for sensori-motor coordination experiments with a swimming robot.

5.B. Experimental approach

The two-dimensional simulation used in this work is based on that developed by Ekeberg (1993). The body of the lamprey is simulated as 10 rigid links connected by one-degree-of-freedom joints. Two muscles are connected in parallel to each joint and are modeled as a combination of springs and dampers. Muscles can be contracted when they receive an input; their spring constant is then increased proportionally to the input, thus reducing the resting length. The motion of the body is calculated from the accelerations of the links. These accelerations result from three forces: the torques due to the muscles, the inner forces linked with the mechanical constraints keeping the links together, and the forces due to the water (Figure 23).

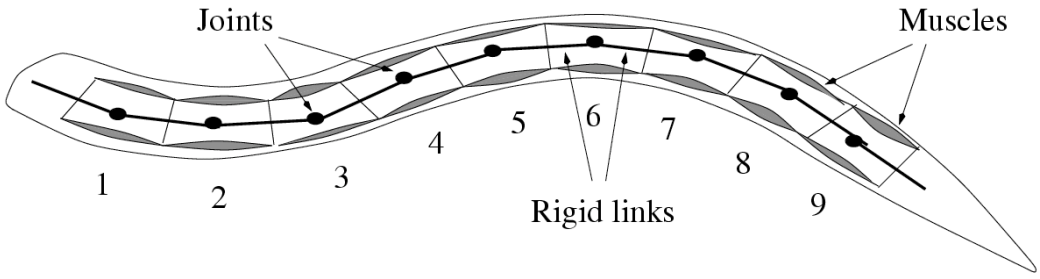


Figure 23. Schema of the mechanical simulation as developed in Ekeberg (1993). The body is composed of ten rigid links connected through nine one-degree-of-freedom joints. The torques of each joint are determined by the contraction of two muscles attached in parallel.

Figure 24 shows that the substrate used was made of nine segments, with two output nodes and two precursor cells each, corresponding to the nine joints of the mechanical simulation. Each of the 18 precursor cells holds a copy of the same developmental program. However, the local frame attached to each cell did not need to have the same orientation, and in particular the precursor cells on each side (left and right) of the body were given opposite orientations, which forced the developed controllers to be symmetrical.

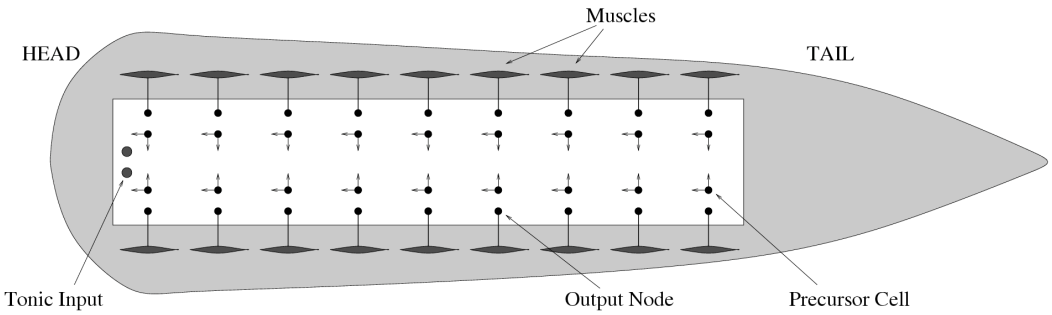


Figure 24. Substrate of the developmental process. The rectangular substrate extends over the length of the mechanical body. It contains 18 output nodes located symmetrically on both sides of the substrate which determine the contraction state of the muscles. It also contains two input nodes which provide the left and right tonic input of the CPG. Eighteen precursor cells are spread over the substrate. Left and right precursor cells have local frames which are oriented with a left-right symmetry. As all precursor cells develop using the same developmental code, this means that a left-right symmetry of the developed network is automatically created.

Figure 25 defines the grammar that was used, which limited the total number of neurons in a controller to 72.

Terminal symbols
DIVIDE, GROW, DRAW, SETBIAS, SETTAU, SETINPUTW, DIE, NOLINK, DEFBIAS, DEFTAU, DEFINPUTW, SIMULT4.
Variables
Start1, Level2, Neuron, Bias, Tau, InputW, Connex, Link.
Production rules
Start1⟶DIVIDE(Level2, Level2)
Level2⟶DIVIDE(Neuron, Neuron)
Neuron⟶SIMULT4(Bias, Tau, InputW, Connex) | DIE
Bias⟶SETBIAS | DEFBIAS
Tau⟶SETTAU | DEFTAU
InputW⟶SETINPUTW | DEFINPUTW
Connex⟶SIMULT4(Link, Link, Link, Link)
Link⟶GROW | DRAW | NOLINK
Starting symbol
Start1.

Figure 25. The grammar used to evolve neural controllers for a swimming lamprey. In addition to the connections created by the developmental program, each created neuron was automatically drawing an afferent connection from the input node of the side of the precursor cell it stemmed from. The default synaptic weights of the corresponding connections (DEFINPUTW) were occasionally modified by the SETINPUTW instructions.

The objective of this research was to develop controllers which could produce patterns of oscillations necessary for swimming when receiving tonic (i.e. non-oscillating) input, and which could modulate the speed and the direction of swimming when the amplitude of the left and right control signals were varied. Therefore, the corresponding fitness function was rewarding controllers which:

- produced contortions, where a contortion was defined as the passage of the center of the body from one side to the other of the line between head and tail;
- reached high swimming speeds;
- could modulate the speed of swimming when the tonic input was varied, with the speed increasing monotonically with the level of input;
- could change the direction of swimming when an asymmetrical tonic input (between the two sides of the body) was applied.

The evaluation of a developmental program consisted of a series of simulations of the corresponding developed controller with different command settings. Fixed-time simulations (6000 ms) were carried

out with different levels of tonic input in order to determine the range of speeds the controller can produce. Asymmetrical initial states for the neurons were created by applying an asymmetrical input during the first 50 ms of the simulation. Starting from a fixed level of input (1.0), the input was varied by increasing and decreasing steps of 0.1, and the range of speed in which the speed increased monotonically with the tonic input was measured. If the range of speeds included a chosen speed (0.15 m/s), a simulation was performed (at the tonic level corresponding to that speed) with a short period of asymmetric tonic input in order to evaluate the capacity of the controller to induce turning. Turning was evaluated by measuring the deviation angle between the directions of swimming before and after the asymmetric input.

The mathematical definition of the fitness function was the following:

$$fitness = fit\_contortion \cdot fit\_\mathbf{max}\_speed \cdot fit\_speed\_range \cdot fit\_turning$$

where *fit_contortion, fit_max_speed, fit_speed_range* and *fit_turning* were functions which were limited between 0.05 and 1.0 and which varied linearly between these values when their corresponding variables varied between two boundaries, a bad and a good boundary. The variables for each function and their corresponding boundaries are given in Table 2. If the speed range did not include the chosen speed of 0.15 m/s, the turning ability was not measured and fit turning was set to 0.05. The fact that the fitness function was a product rather than a sum ensured that controllers which performed equally in all four aspects were preferred over controllers performing well in some aspects but not in others.

Function	Variable	[bad,good] Boundaries
fit_contortion	Number of contortions	[0,10] contortions
fit_max_speed	Maximum speed	[0.0,1.0] m/s
fit_speed_range	Relative speed range	[0.0,1.0]
fit_turning	Deviation angle	[0.0,0:75ss]

Table 2. Variables and boundaries for the fitness function in the swimming lamprey experiment. A contortion is measured as the passage of the center of the body from one side to the other of the line between head and tail. The speed range is measured relative to the maximum speed.

Ten evolutions were carried out with populations of 50 controllers, each run starting with a different random population and lasting for 100 generations.

5.C. Experimental results

Figure 26 shows the swimming behavior induced by the 10 best controllers that have been obtained. As described in Ijspeert and Kodjabachian (1999), these controllers can be classified into three categories: those which do not produce oscillations (controllers 1, 2, 3, and 4), those which produce chaotic behavior (controllers 5 and 6) and those which produce stable oscillations (controllers 7, 8, 9, and 10).

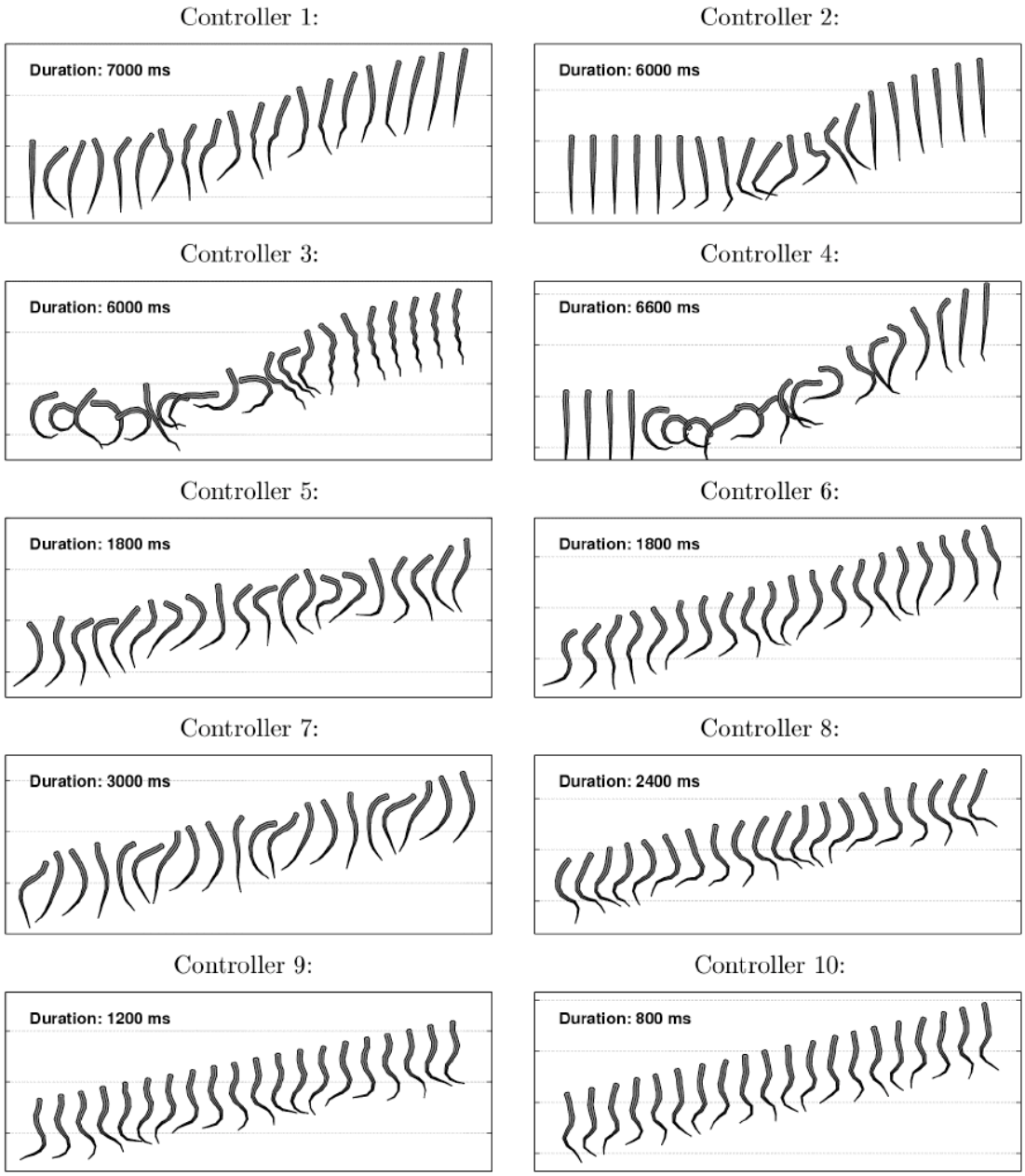
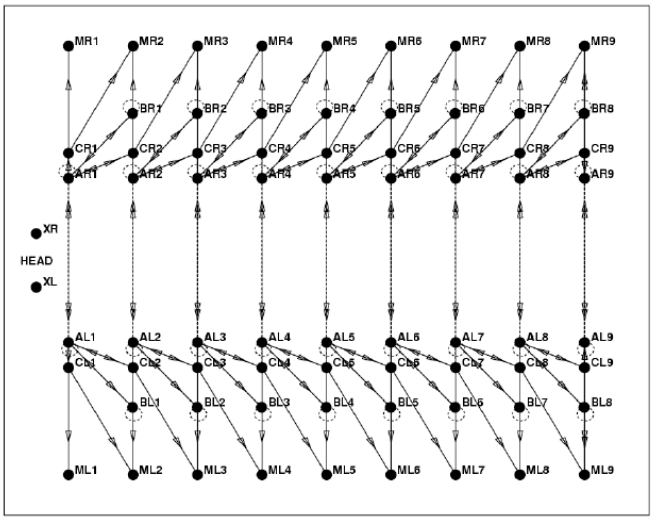


Figure 26. Swimming produced by controllers 1 to 10. The horizontal lines are separated by 200mm.

The architecture of controller 10, which led to the highest fitness, is given in Figure 27. Basically, it implements one oscillator per segment, such oscillators being distributed over both sides of the spinal cord through inhibitory contralateral connections. The segments are interconnected over the spinal cord by closest-neighbor coupling.



Parameters:		Weights:					
Tau:		Segment 1		Segment 5		Segment 9	
A:	1600.0	AL1 → ML1	27.00	CL4 → ML5	31.80	CL8 → ML9	31.80
B:	29.2	XL → AL1	28.45	AL5 → ML5	27.00	AL9 → ML9	27.00
C:	16.5	BL1 → AL1	−6.10	XL → AL5	28.45	CL9 → ML9	31.80
		CL2 → AL1	−24.75	BL5 → AL5	−6.10	XL → AL9	28.45
Bias:		AL1 → AL1	−23.95	CL6 → AL5	−24.75	CL9 → AL9	−6.10
A:	−16.00	XL → BL1	1.00	AL5 → AL5	−23.95	CL9 → AL9	−24.75
B:	−16.00	BL1 → BL1	−16.20	XL → BL5	1.00	AL9 → AL9	−23.95
C:	2.00	AL1 → BL1	24.70	BL5 → BL5	−16.20	XL → CL9	1.00
		XL → CL1	1.00	AL5 → BL5	24.70	CR9 → CL9	−6.95
		CR1 → CL1	−6.95	XL → CL5	1.00	AR9 → CL9	−27.90
		AR1 → CL1	−27.90	CR5 → CL5	−6.95	AL8 → CL9	30.70
		AL1 → CL1	30.70	AR5 → CL5	−27.90		
				AL4 → CL5	30.70		

Figure 27. Architecture of controller 10, one of the two controllers producing anguiliform swimming. Left: Neurons on the substrate. For the sake of clarity, the lateral axis of the substrate has been multiplied by ten. Ml_i and Mr_i represent the left and right muscle nodes of segment i; X_l and X_r are the input nodes. A, B and C are three types of neurons. Excitatory and inhibitory connections are represented through continuous and dashed lines respectively. Each neuron also receives a connection from the input nodes, which is not shown here. Right: Parameters and connection weights. Only the weights of the connections to the left neurons are given, the others are symmetrical.

Such a controller produced anguiliform swimming very similar to that of the real lamprey. The corresponding animat had left and right neurons oscillating out of phase, and phase lags between segments which were almost constant over the spinal cord leading to caudally directed traveling waves (Figure 28). Moreover, it could modulate the speed of swimming when the level of tonic input was varied, this speed increasing monotonically with the excitation in a given range of excitation. It also afforded good control of the direction of swimming such that, when an asymmetrical excitation was applied to the neural network for a fixed duration, the lamprey turned proportionately to the asymmetry of excitation.

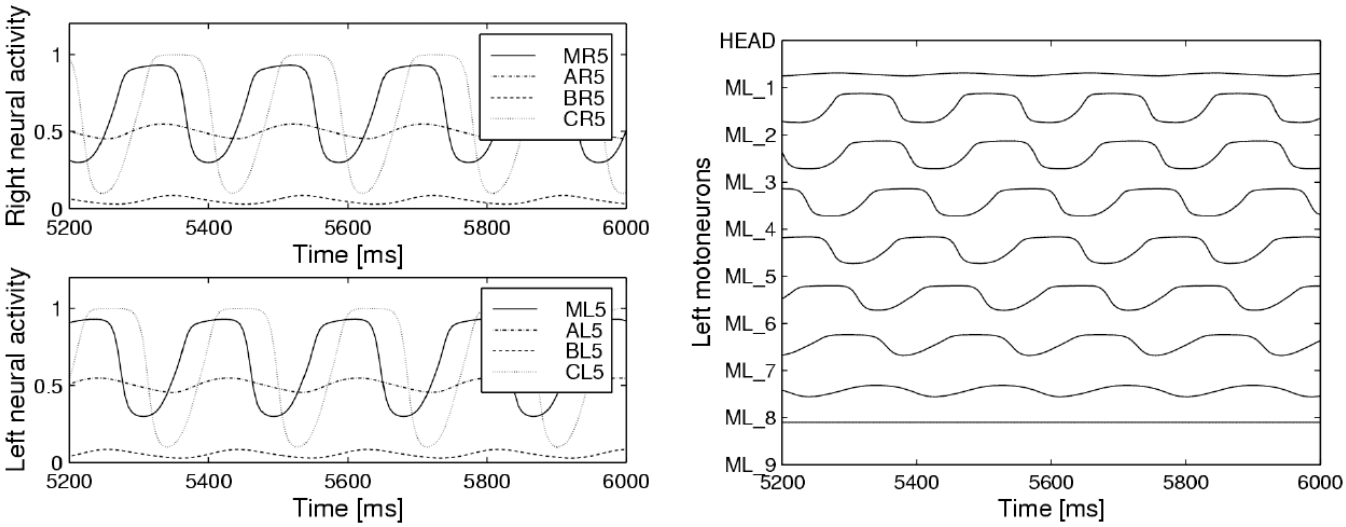


Figure 28. Neural activity in controller 10. Left: Neural activity of left and right neurons and the signals sent to the muscles in segment 5. Right: Signals sent to the left muscles along the nine segments.

The effect of such asymmetry was to change the duration of bursts between left and right muscles leading to a change of direction towards the side with the longest bursts. If the asymmetry was small (for instance ±10% of the excitation level), the lamprey went on propagating a traveling undulation and therefore swam in a circle until the asymmetry was stopped (Figure 29, top). If the asymmetry was large

(for instance ± 30% of the excitation level), the undulations almost stopped as one side became completely contracted. This bending caused the lamprey to turn very sharply, and allowed important changes of direction when the duration of the asymmetry was short (Figure 29, bottom).

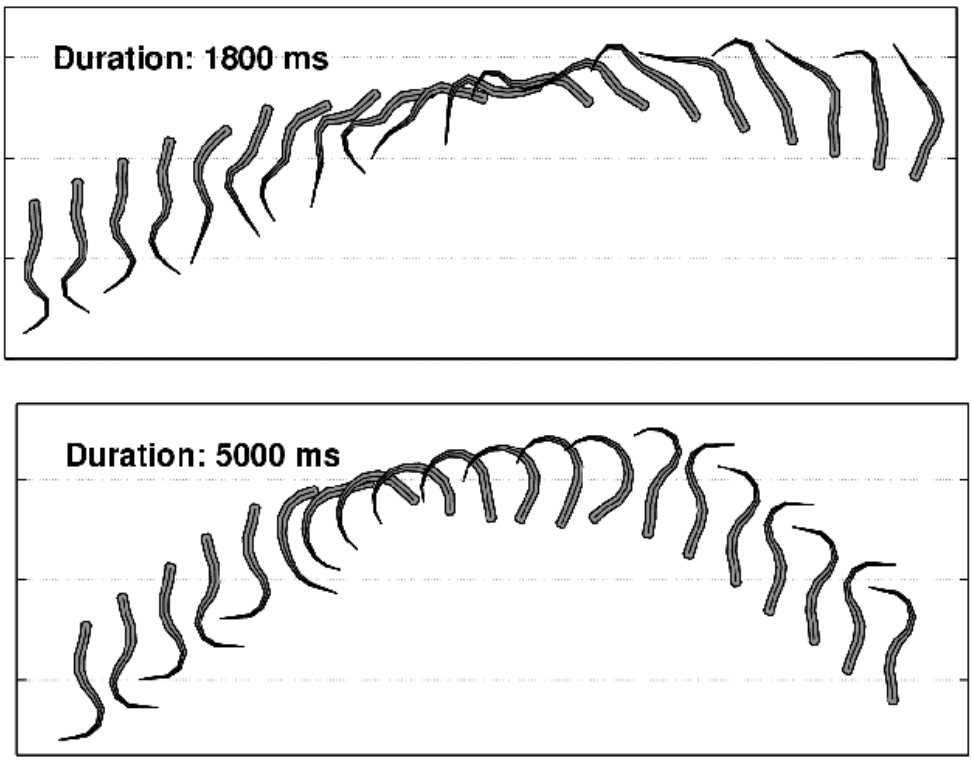


Figure 29. Top: slow turning. Bottom: Sharp turning in a simulated lamprey.

6. A flying animat

6.A. The problem

In this application (Doncieux 1999), neural networks were evolved for flying control in a simulated airship, i.e., an helium tank with a nacelle and some motors. Such a device is very stable and easy to pilot, but is highly sensitive to the wind. The objective was therefore to test the possibility of evolving neural controllers that could combat the effects of wind and maintain a constant flying speed. Ultimately, such methodology could be used to assist the piloting of real airships.

A two-stage approach was used, in which controllers resistant to a wind blowing parallel to the airship's axis (front and rear) were first evolved, and then improved to operate also under more challenging conditions, when sideways winds were affecting the airship's trajectory. Again, comparisons with results obtained under the alternative one-shot strategy did support the above-mentioned intuition about the usefulness of an incremental approach.

6.B. Experimental approach

The simulated airship that was used in this work (Figure 30) was equipped with two motors. It was considered as a point that stayed at a constant altitude and moved in an infinite two-dimensional space. Its dynamics were given by the following equations:

$$\begin{cases} M\ddot{x} + D_v\dot{x} + F_x = 0 \\ M\ddot{y} + D_v\dot{y} + F_y = 0 \\ I\ddot{\theta} + D_w\dot{\theta} + \tau = 0 \end{cases} \quad (4)$$

in which x, y and θ were the airship's coordinates and orientation angle, D_v and D_w were the translation and rotation dragging coefficients, F were the resultant vectors of forces, τ was the torque, and I was the rotational moment of inertia.

According to these equations it is possible, knowing the wind speed, to compute the airship's speed with respect to the ground.

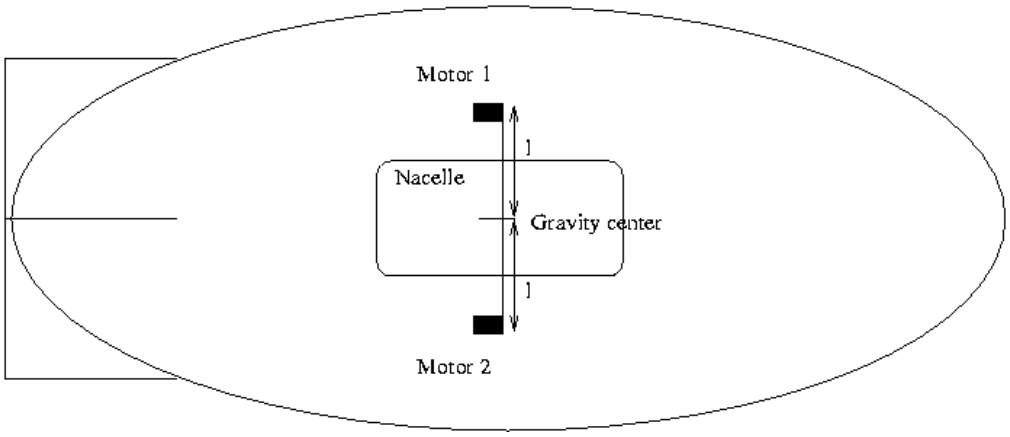


Figure 30. The simulated airship.

The following constants were used:

Weight of the airship: M = 7g

Weight of each motor: M_p = 1g

Distance of each motor from the gravity center: l = 10 cm

Maximum force delivered by each motor: 15N

$D_v = 17\ kg.s^{-1}$

$D_w = 70\ kg.s^{-1}$

In a first evolutionary stage, controllers were sought that could maintain the airship's speed at a target value relative to the ground in the presence of an axial wind. The corresponding fitness was given by the following equation:

$$fitness = \frac{100}{1 + \sum_{t \le T} \left| v_x(t) - \text{vtarget} \right|} \quad (5)$$

in which $v_x(t)$ was the airship speed at instant t, and vtarget was the target speed. The evaluation time T was set to 200 time-steps, and each individual was tested under five different conditions to ensure the controller would be efficient even in case of changing wind speeds. In the first one, the wind speed was permanently set at 0. At the beginning of the four others, the wind speed was set respectively at a small positive value, a high positive value, a small negative value, and a high negative value, and these speeds were suddenly inverted in the middle of each run.

The GRAM-1 grammar of Figure 5 was used, together with a substrate including two symmetrical precursor cells, two sensory cells, and four motoneurons. The activation levels of the sensory cells were respectively proportional to negative and positive differences between the airship and the target's speeds. The motoneurons determined the activity of the motors, which could contribute to pushing the airship either forward or backward.

6.C. Experimental results

Figure 31 shows the best controller obtained after 16000 generations, with a population of 100 individuals. Figure 32 shows the airship's reactions to a varying axial target speed, an experimental setup where changing the target speed was equivalent to changing the wind speed. The corresponding behavior mostly depends on three sub-circuits within the controller – that basically accelerate or slow down the airship in the right timing – and on neurons with very short time constants tailored by evolution – that are only used at the beginning of the simulation to combat the airship's inertia and facilitate a quick return to the target speed. These mechanisms are detailed in Doncieux (2001).

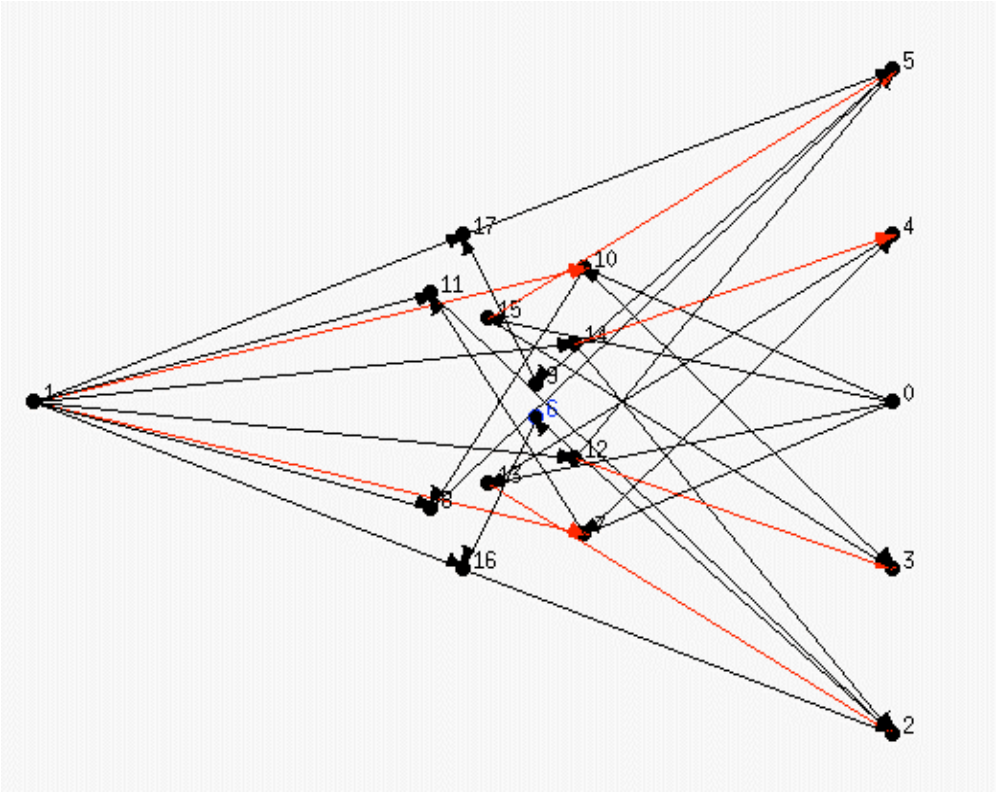


Figure 31. Best neural network of the 16000th generation in the simulated airship experiment. Neuron 0: positive speed. Neuron 1: negative speed. Neuron 5: "move forward" command for motor 1. Neuron 4: "move backward" command for motor 1. Neuron 3: "move backward" command for motor 2. Neuron 2: "move forward" command for motor 2.

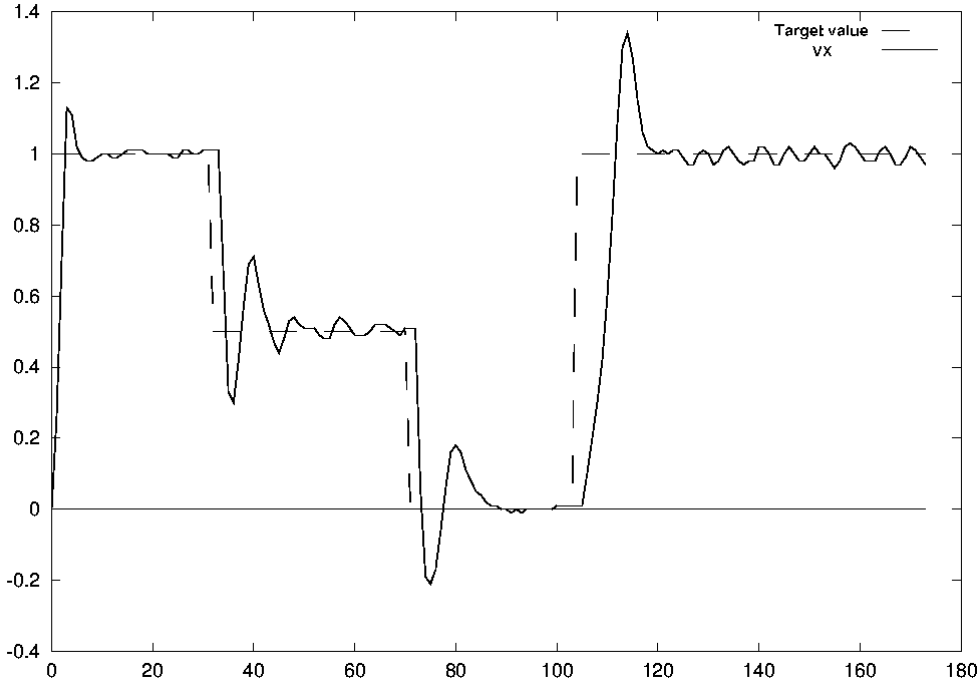


Figure 32. Response of the best neural network of the 16000th generation to a variable target value (X direction) that the airship succeeds in tracking.

At the end of a second evolutionary stage, after 1500 additional generations, a second controller interacting with the preceding one and helping the airship to resist lateral winds was obtained. This controller and the behavior it generates are illustrated on Figures 33 and 34. Here again, evolution resorted to special neurons, some of which reacted to strong stimulations only, while others exhibited more subtle behaviors (Doncieux 2001).

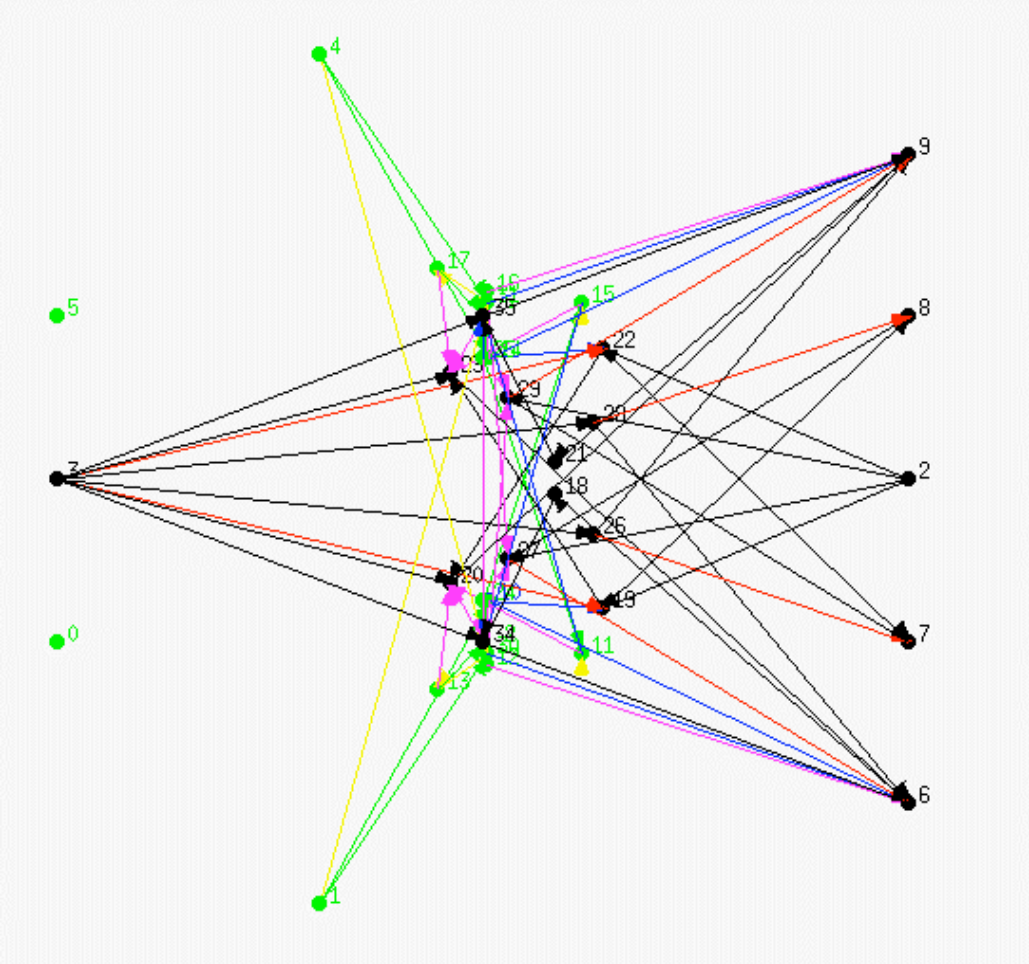


Figure 33. Best neural network after 1500 generations on the problem of variable wind. Neuron 0: negative angle relative to heading. Neuron 1: negative speed perpendicular to heading. Neuron 2: positive speed. Neuron 3: negative speed. Neuron 4: positive speed perpendicular to heading. Neuron 5: positive angle relative to heading. Neuron 6: "move forward" command for motor 2. Neuron 7: "move backward" command for motor 2. Neuron 8: "move backward" command for motor 1. Neuron 9: "move forward" motor 1.

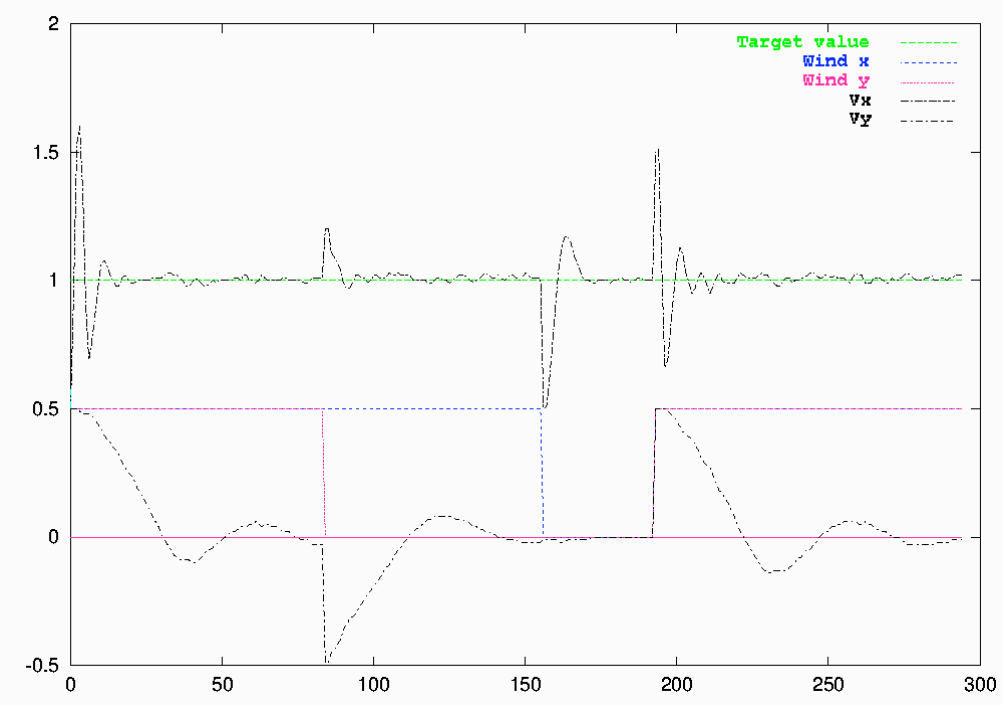


Figure 34. Behavior corresponding to the best neural network after 1500 generations with a lateral wind. The airship is initially blown off its course, but it succeeds in resuming its assigned trajectory by orienting itself towards the direction from which the wind is blowing, thus counterbalancing the effect of this latter.

Finally, to assess the usefulness of the incremental approach that was used here, complementary evolutionary runs were performed where the airship was confronted with lateral winds from the outset. As shown in Doncieux (1999), similar results were obtained with both approaches, but with significant saving of time in the incremental case.

A lenticular airship 10 meters-wide is currently under construction. The SGOCE methodology will be used to generate neural networks that will assist remote piloting, in a first stage, and secure autonomous control, in a second stage.

7. Discussion

The various applications that have been described here and elsewhere demonstrate that evolutionary algorithms are valuable techniques for designing behavioral controllers for animats and robots. Compared to traditional learning algorithms, these techniques are more flexible, notably because the fitness functions they call upon do not need to be differentiable or even continuous, like error functions minimized by most learning algorithms. There is also no need to provide details about the specific behaviors that such controllers are expected to control, because higher-level specifications of both an appropriate fitness function and a minimal performance level are sufficient.

The above-mentioned applications have also demonstrated that neural networks may be efficient at controlling behaviors as diverse as rolling, walking, swimming, and flying.

Finally, these applications exemplified how the SGOCE evolutionary paradigm can be used conveniently to develop neural controllers in simulated animals and real robots with fixed body structures. In the latter case, neural controllers that were evolved in simulation proved to be efficient at controlling real robots in real environments.

Although the fact that such an evolutionary process – which determines both the number and the characteristics of the neurons and connections that are included in a given controller – may lead to a infinite search space, the corresponding search may be channeled in several ways to make the task of the evolutionary algorithm easier. In particular, it is possible to reduce the overall number of parameters to be evolved by selecting the initial positions and orientations of the precursor cells, the sensory cells and the motoneurons in the substrate, thus committing the developed controllers to being symmetrical and segmented. Through the choice of a specific grammar, it is also possible to reduce the size of the search space, notably by limiting the numbers of neurons and connections in the evolved controllers. Such reductions also help to keep the evaluation time of the corresponding neural networks within reasonable bounds. Finally, recourse to an incremental approach also seems to afford possibilities for reducing the evolutionary search space.

However, due to lack of hindsight, it is presently unclear whether every aspect of the SGOCE paradigm is mandatory, useful, or rather has no effect at all. Although many identical implementation details have been used successfully in the above-mentioned applications, one may wonder, for instance, whether another evolutionary procedure – e.g., an evolution strategy (Schwefel 1995) – would not lead to better results. Likewise, it is unclear if the developmental instructions or the constraining grammars that have been used here might not have been replaced by others, and if freezing one controller and letting a second one evolve is a better strategy than evolving both controllers at the same time. Concerning the latter point, however, the results obtained with the rolling Khepera and the flying airship have shown that this was not the case, but any generalization to other behaviors and other robots would be quite premature. Finally, it must be acknowledged that there is at the moment definitely no hint about the convergence properties of the algorithms that have been used here.

Potentially fruitful directions for future research consist in allowing several of the characteristics to evolve that have so far been set arbitrarily by the experimenter. Other interesting paths to explore are those that would lead to more complex control mechanisms than the mere reflexes that were implemented in the various controllers described above. Although neural controllers implementing a rudimentary memory have already been obtained with SGOCE (Kodjabachian and Meyer 1998a), it is presently unclear whether this approach is capable of generating more elaborate cognitive faculties, like other varieties of memory or planning capacities. Still another promising possibility is that of devising new developmental instructions that would confer individual learning capacities on an animat complementing those of evolution and development. For instance, additional developmental instructions could be devised that would allow the synaptic weights of some connections to be changed during an animat's lifetime, thanks to an individual learning process. Similarly, other developmental instructions could be devised that would allow the developmental pathway to be dynamically changed depending upon the specific interactions that the animat experiences with its environment. Besides its operational value, every step in these directions would contribute to theoretical biology and afford a better understanding of the interactions between development, learning and evolution, i.e., the three main adaptive processes exhibited by natural systems.

8. Conclusion

It has been shown here that the current implementation of the SGOCE evolutionary paradigm makes it possible to automatically design the control architectures of rolling, walking, swimming, flying animats or robots. Such results rely upon specific mechanisms implementing the developmental process of a recurrent dynamic neural network in a given substrate, in which precursor cells, sensory cells and motoneurons are initially arranged by the experimenter. There are several ways of improving the corresponding mechanisms, in particular by letting some characteristics evolve that have been arbitrarily set here by the experimenter, or by devising new developmental instructions that would add individual learning capacities to the processes of development and evolution. Such research efforts might be as useful in an engineering perspective as they are in contributing to a better understanding of the mechanisms underlying adaptation and cognition in natural systems.

References

Beer R, Gallagher J (1992) Evolving dynamical neural networks for adaptive behavior. Adaptive Behavior 1(1):91-122

Bongard J C, Paul C (2001). Making Evolution an Offer it Can't Refuse: Morphology and the Extradimensional Bypass. To appear in Proceedings of the 6th European Conference on Artificial Life (ECAL2001). Springer Verlag

Boers E, Kuiper H (1992) Biological Metaphors and the Design of Modular Artificial Neural Networks. Master's Thesis. Department of Computer Science and Experimental and Theoretical Psychology. Leiden University

Brooks R A (1986) A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2:14-23

Cangelosi A, Parisi D, Nolfi S (1994) Cell division and migration in a 'genotype' for neural networks. Network 5:497-515

Chavas J, Corne C, Horvai P, Kodjabachian J, Meyer J A (1998) Incremental evolution of neural controllers for robust obstacle-avoidance in Khepera. In: Husbands P, Meyer J A (eds) Proceedings of The First European Workshop on Evolutionary Robotics - EvoRobot'98. Springer Verlag, pp 224-244

Cliff D, Harvey I, Husbands P (1993) Explorations in evolutionaryrobotics. Adaptive Behavior 2(1) :73-110

Conrad M. (1990) The geometry of evolution. Biosystems 24: 61-81

Cliff D, Miller G F (1996) Co-evolution of pursuit and evasion II: Simulation methods and results. In: P. Maes P, Mataric M J, Meyer J A, Pollack J B, Wilson S W (eds) (1996) Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4. The MIT Press/Bradford Books, Cambridge, MA, pp 506-515

Cruse H, Brunn D E, Bartling C, Dean J, Dreifert M, Kindermann T, Schmitz J (1995) Walking: A complex behavior controlled by simple networks. Adaptive

Behavior 3(4):385-418

Dellaert F, Beer R (1994) Toward an evolvable model of development for autonomous agent synthesis. In: Brooks R A, Maes P (eds) Proceedings of the Fourth International Workshop on Artificial Life. The MIT Press/Bradford Books, Cambridge, MA, pp 393-401

Doncieux S (1999) Evolution d'architectures de contrôle pour robots volants. In: Drogoul A, Meyer J A (eds) Intelligence artificielle située. Hermès, Paris, pp 109-127

Doncieux S (2001) Evolution d'Architectures de Contrôle pour Animats Volants. Mémoire de stage de DEA IARFA. Université Pierre et Marie Curie.

Eggenberger E (1996) Cell-cell interactions as a control tool of developmental processes for evolutionary robotics, In: Maes P, Mataric M J, Meyer J A, Pollack J B, Wilson S W (eds) Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 4, The MIT Press/Bradford Books, Cambridge, MA, pp 440-448

Eggenberger, P. (1997). Evolving Morphologies of Simulated 3D Organisms Based on Differentiable Gene Expression. In Proceedings of the Fourth European Conference on Artificial Life. Husbands and Harvey (Eds.). 205-213. MIT Press

Ekeberg O (1993) A combined neuronal and mechanical model of swimming. Biological Cybernetics 69:363-374

Filliat D, Kodjabachian J, Meyer J A (1999a) Evolution of neural controllers for locomotion and obstacle-avoidance in a 6-legged robot. Connection Science 11: 223-240

Filliat D, Kodjabachian J, Meyer J A (1999b) Incremental evolution of neural controllers for navigation in a 6-legged robot. In: Sugisaka M, Tanaka H (eds) Proceedings of the Fourth International Symposium on Artificial Life and Robotics. Oita University Press, Beppu, Oita, Japan, pp 753-760.

Goldberg D E (1989) Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA

Gruau F (1994) Automatic definition of modular neural networks. Adaptive Behavior 3(2): 151-183

Gruau F (1996) Artificial cellular development in optimization and compilation. In: Sanchez E, Tomassini M (eds) Evolvable Hardware'95. Lecture Notes in Computer Science, Springer Verlag, pp 48-75

Gruau F, Quatramaran K (1997) Cellular Encoding for Interactive Evolutionary Robotics. In: Husbands P, Harvey I (eds) Fourth European Conference on Artificial Life. The MIT Press/Bradford Books, Cambridge, MA, pp 368-377

Harvey, I. (1992). Species Adaptation Genetic Algorithms: The Basis for a Continuing SAGA. In Proceedings of the First European Conference on Artificial Life. Varela and Bourgine (Eds.). The MIT Press Cambridge, MA, pp 346-354

Harvey I., Husbands P, Cliff D (1994) Seeing the light: Artificial evolution, real vision. In: Cliff D, Husbands P, Meyer J A, Wilson S W (eds) Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3. The MIT Press/Bradford Book, Cambridge, MA, pp 392-401

Husbands P, Harvey I., Cliff D, Miller G (1994) The use of genetic algorithms for the development of sensorimotor control systems. In: Nicoud J, Gaussier P (eds) From Perception to Action. Proceedings of the PerAc'94 Conference. IEEE Computer Society Press, Los Alamitos, CA, pp 110-121

Ijspeert A J, Kodjabachian J (1999) Evolution and development of a central pattern generator for the swimming of a lamprey. Artificial Life 5(3): 247-269

Jakobi N (1997a) Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In: Husbands P, Harvey I (eds) Fourth European Conference on Artificial Life. The MIT Press/Bradford Books, Cambridge, MA, pp 348-357

Jakobi N (1997b) Evolutionary robotics and the radical envelope of noise hypothesis. Adaptive Behavior 6(2): 325-367

Jakobi N (1998) Running across the reality gap: octopod locomotion evolved in minimal simulation. In: Husbands P, Meyer J A (eds) Proceedings of The First European Workshop on Evolutionary Robotics: EvoRobot'98. Springer Verlag, pp 39-58

Kodjabachian J, Meyer, J A (1995) Evolution and development of control architectures in animats. Robotics and Autonomous Systems 16: 161-182

Kodjabachian J, Meyer J A (1998a) Evolution and development of modular control architectures for 1-D locomotion in six-legged animats. Connection Science 10: 211-237

Kodjabachian J, Meyer J A (1998b) Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. IEEE Transactions on Neural Networks 9(5): 796-812

Koza J (1992) Genetic Programming: On the programming of computers by means of natural selection. The MIT Press, Cambridge, MA

Koza J (1994) Genetic Programming II: Automatic discovery of reusable sub-programs. The MIT Press, Cambridge, MA

Koza J R, Bennett F H, Andre D, Keane M A (1999). Genetic Programming III. Morgan Kaufmann Publishers

Lee W P, Hallam J, Lund H H (1997) Learning complex robot behaviours by evolutionary approaches. In: Brink A, Demiris J (eds) Proceedings of the 6th European Workshop on Learning Robots. Springer, Brighton, pp 42-51

McClelland J L, Rumelhart D E (eds) (1996) Parallel Distributed Processing. Vol 1. The MIT Press/Bradford Books, Cambridge, MA

Meyer J A (1998) Evolutionary approaches to neural control in mobile robots. In: Proceedings of the IEEE International Conference on Systems and Cybernetics. San Diego, pp 35-40.

Michel O (1995) An artificial life approach for the synthesis of autonomous agents. In: Alliot J, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Proceedings of the European Conference on Artificial Evolution. Springer Verlag, pp 220-231

Michel O, Collard P (1996) Artificial Neurogenesis: An application to autonomous robotics. In: Radle M G (ed) Proceedings of The 8th International Conference on Tools in Artificial Intelligence. IEEE Computer Society Press, Piscataway, NJ, pp 207-214

Mondada F, Franzi E, Ienne P (1993) Mobile robot miniaturization: A tool for investigation in control algorithms. In: Yoshikawa T, Miyazaki F (eds) Proceedings of the Third International Symposium on Experimental Robotics. Springer Verlag, Tokyo, pp 501- 513

Nolfi S, Miglino O, Parisi D (1994) Phenotypic plasticity in evolving neural networks. In: Gaussier P, Nicoud J (eds) From Perception to Action. Proceedings of the PerAc'94 Conference. IEEE Computer Society Press, Los Alamitos, CA, pp 146-157

Rumelhart D E, McClelland J L (eds) (1986) Parallel Distributed Processing. Vol 2. The MIT Press/Bradford Books, Cambridge, MA

Shipman, R. (1999). Genetic Redundancy: Desirable or Problematic for Evolutionary Adaptation? In Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms. 337-344. Springer Verlag

Sims K (1994) Evolving virtual creatures. In: Glassner A (ed) Computer Graphics Proceedings: SIGGRAPH '94, Annual Conferences Series, Association for Computing Machinery, New York, pp 15-22

Spencer G (1994) Automatic generation of programs for crawling and walking In: Kinnear K E Jr (ed) Advances in Genetic Programming, The MIT Press / Bradford Books, Cambridge, MA, pp 335-353

Vaario J (1993) An emergent modeling method for artificial neural networks, Ph.D. thesis, University of Tokyo

Vaario J, Onitsuka A, Shimohara K (1997) Formation of neural structures. In: Husbands P, Harvey I (eds) Proceedings of the Fourth European Conference on Artificial Life. The MIT Press, Cambridge, MA, pp 214-223